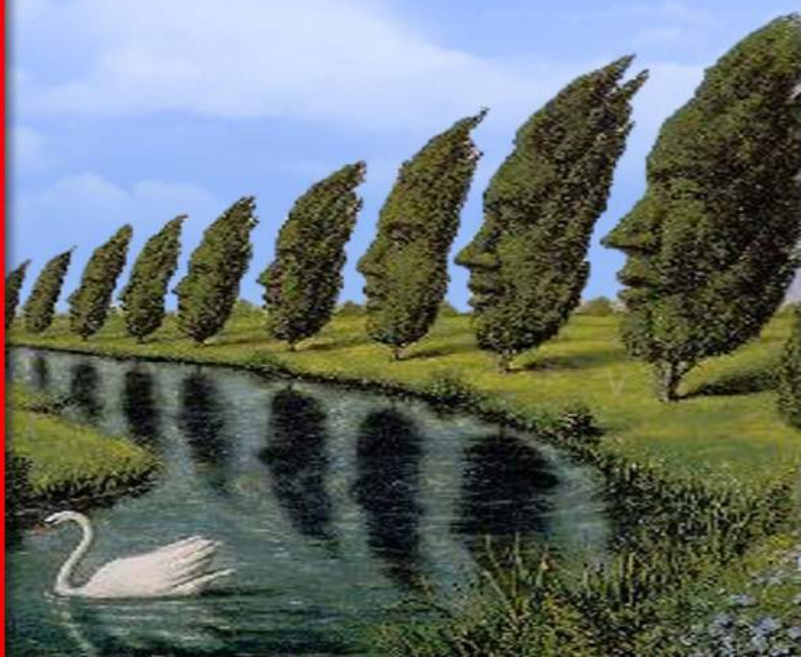
Müasir proqramlaşdırma dilləri

Z.T.Məhərrəmov

PASCALdan DELPHIyə



Z.T. Məhərrəmov

PASCALdan DELPHIyə

ALİ MƏKTƏB TƏLƏBƏLƏRİ ÜÇÜN
DƏRS VƏSAİTİ

(Üçüncü nəşr)

Azərbaycan Respublikası Təhsil Nazirliyi
tərəfindən təsdiq edilmişdir

BAKI – 2014

Rəy verənlər:

Texnika elmləri doktoru, professor **Fazil Həzin oğlu Ələkbərli**
Texnika elmləri namizədi, dosent **Muxtar Feyzi oğlu Bəhmənov**

Elmi redaktor:
Texnika elmləri doktoru **Əli Həsən oğlu Nağıyev**

Məhərrəmov Z.T.
Pascal–dan Delphi–yə. Bakı, 2014. – 412 s.

Kitab ilk növbədə Turbo Pascal və Delphi dillərini öyrənən ali məktəb tələbələri üçün nəzərdə tutulmuşdur. Kitabda Object Pascal dili şərh olunmuş, obyektyönlü proqramlaşdırmanın prinsipləri izah edilmişdir. 300–ə qədər geniş əhatəli çoxlu məsələlər həll edilmiş, müfəssəl izahatlar proqram mətnləri ilə müşayiət olunmuşdur. Əhatə olunmuş mövzular ətraflı izah edilmiş, oxucunun dərin və dərrakəli vərdişlər almasına kömək edir ki, bu da müxtəlif xarakterli məsələləri sərbəst həll etməyə, Turbo Pascal və Delphi–yə aid əlavə kitablardan maneəsiz istifadə etməyə imkan verir. Nəzəri materialların və həll edilmiş məsələlərin sadə izahı Turbo Pascal və Delphi dillərini bilməyən, onu sərbəst öyrənmək istəyənlərin, məsələlərin zənginliyi isə Windows əlavələri yaratmaqla məşğul olan istənilən mütəxəssislərin bu kitabdan istifadə etməsinə imkan verir.

MÜNDƏRİCAT

MÜƏLLİFDƏN

Delphi hal–hazırda bütün dünyada ən geniş yayılmış və ən perspektivli obyektyönlü proqramlaşdırma dilidir. Təəssüflər olsun ki, ölkəmizdə bu dil çox da geniş yayılmamışdır. Hətta, bu azmış kimi, qeyri–peşəkarlar arasında belə fikir formalaşmağa başlamışdır ki, guya bu dil öz ömrünü yaşamış və artıq səhnədən getmək üzrədir. Buna da səbəb kimi, Delphi dilinə alternativ olaraq, C^{++} və Java dillərini qarşı qoyurlar. Hər bir dilin öz təyinat sahələri və üstünlükləri vardır. Söhbət hər hansı bir fiziki qurğuya drayver yazmaqdan gedirsə, onda əlbəttə, Delphi Sizə kömək edə bilməyəcək, bu halda müstəsna olaraq C^{++} dili tətbiq edilməlidir. Bu və ya digər tətbiqi məsələləri həll etdikdə, çoxfunksiyalı əlavələr yaratmaq zərurəti ortaya çıxdıqda isə Delphi əvəzsiz bir dildir. 30 ildən artıq bir müddətdə ən müxtəlif elmi–texniki və mühəndis xarakterli məsələlərin proqramlaşdırılması və proqramlaşdırmanın tədrisi ilə məşğul olan bir mütəxəssis kimi deyə bilərəm ki, Delphi–də proqramlaşdırma ilə məşğul olduqdan sonra, mənim qarşımda yeni bir proqramlaşdırma dünyası açıldı. Delphi–nin meydana gəlməsi proqramçıların illərlə xəyallarında yaşatdıqları bir arzunu reallaşdırdı. Bu dil nəinki, sadəcə olaraq məsələni proqramlaşdıraraq son nəticələri almağa imkan verir, bu dildə proqramlaşdırma prosesi özü proqramçıya həm də mənəvi zövq və həzz verir. Bu dildə proqramlaşdırma ilə yanaşı, ənənəvi proqramlaşdırmaya heç bir aidiyyəti olmayan vizual konstruksiyalaşdırma işləri paralel olaraq yerinə yetirilir. İşin bir hissəsi konstruksiyalaşdırma formasında, digər bir hissəsi proqramlaşdırma formasında yerinə yetirilir. Proqramlaşdırma işlərinin çox böyük bir hissəsini isə tamamilə öz öhdəsinə götürərək, bizim əvəzimizdən, Delphi yerinə yetirir. Mən belə yüksək avtomatlaşdırma səviyyəsinə malik olan ikinci bir dil tanımıram. Delphi–yə aid oxuduğum ədəbiyyatların əksəriyyətinin müəllifləri də bu fikirdədirlər. Məhz Delphi–yə olan bu həvəs və marağım, onun həm də həmvətənlərimə öyrədilməsi arzusu, məni vadar etdi ki, Delphi–yə aid kitab yazım və 2004 – cü ildə Delphi–yə aid ilk dərs vəsaiti nəşr olundu.

Əslində Delphi proqramlaşdırma dili deyil, proqramlaşdırma mühitidir, sistemidir. Delphi–nin proqramlaşdırma dili Pascal dilinin müstəsna olaraq obyektyönlü versiyası olan Object Pascal dilidir. Delphi–yə yol Pascal–dan keçir. Ona görə də, bu dəfə dərs vəsaiti üzərində yenidən işlədikdə qərara gəldim ki, kitaba Turbo Pascal 7.0 dilini əlavə edim. Bu, həm çoxsaylı tələbələrimin arzu və təkliflərinə cavabdır, həm də, digər tərəfdən, Pascal dilini bilmədən Delphi–ni öyrənmək mümkün deyil. Pascal dilini mükəmməl öyrənmək üçün isə, zənnimcə, o ayrılıqda öyrənilməlidir. Bununla da tələbə son nəticədə daha böyük nailiyyət əldə edir – dərhal iki sərbəst dil öyrənir. Kitabda Turbo Pascal dili çox anlaşıqlı misallar üzərində izah edilmiş, proqramlaşdırma üçün xarakterik olan alqoritmlər ümumiləşdirilmişdir. Çoxlu mürəkkəb məsələlərin proqramları bu kitabda həll edilmiş sadə nümunəvi məsələlərin proqramlarından ibarət olur. Object Pascal dili isə kitabın ikinci hissəsində şərh olunmuşdur. Burada əsas diqqət

Turbo Pascal dilindən fərqli cəhətlərə yönəlmiş, obyektyönlü proqramlaşdırmanın prinsip və xüsusiyyətləri çox ətraflı izah edilmişdir. Pascal dilini bilənlər bu bölmələri oxumaya bilər, lakin mən məsləhət görərdim ki, hər kəs bu kitabı, tanışlıq kimi, əvvəldən axıradək bir dəfə oxusun, sonra isə öz seçimini etsin. Şəxsən mən, istənilən kitabı əvvəldən axıradək gözəyarı bir dəfə oxuyuram – sanki, nəyi, harada axtarmağı özüm üçün müəyyənləşdirirəm, sonra isə hər bir paraqrafı dönə–dönə oxuyuram. Bundan başqa, hər hansı bir dili öyrənmək üçün bir kitab heç vaxt kifayət edə bilməz, çoxlu kitablar içərisindən Sizin asan qavradığınız, ürəyinizə yatan 3–4 kitabı seçib onları təkrar–təkrar oxumaq lazımdır.

Kitabda 300–ə qədər misal nümunələri verilmişdir ki, bunlardan təqribən yüzü tam hazır proqramdır. Delphi–də isə 40 tam hazır layihə yerinə yetirilmişdir. Həll edilmiş məsələlərin əhatə dairəsi çox geniş olduğu üçün, Delphi–də proqramlaşdırmanın əsas prinsipləri əks etdirilmişdir. Bu kitab məlumat kitabı deyildir, müəllif, tələbələrə proqramlaşdırmanın prinsiplərini öyrətməyi qarşısına məqsəd qoymuşdur. İndiyədək Delphi–yə aid elə bir kitab yazılmamışdır ki, onun çoxsaylı komponentlərini tam əhatə etmiş olsun. Delphi–də proqramlaşdırma sonu olmayan bir prosesdir və heç kim deyə bilməz ki, mən Delphi dilini tam bilirəm. Kitabda verilmiş izahatlar, həll edilmiş bütün məsələlər Delphi–nin bütün versiyaları üçün doğrudur. Bu məsələlərin bəziləri vaxtilə Delphi 4, əksəriyyəti isə Delphi 7 versiyalarında həll edilmişdir. Həmin məsələlər hətta Delphi 2006 versiyasında da problemsiz olaraq işləyir. Yeri gəlmişkən qeyd edim ki, Delphi yalnız *Windows* əməliyyat sistemi mühitində deyil, həm də *Linux və Kylix* sistemləri mühitində problemsiz işləyir.

Nəhayət dil ilə əlaqədar bəzi məsələlərə toxunmaq istəyirəm. Son zamanlar, doğma dilimizdə informasiya texnologiyasının öyrənilməsi ilə əlaqədar xeyli dərs vəsaiti və kitablar nəşr olunmağa başlamışdır. Əlbəttə, bu sevindirici haldır, hərçənd ki, bunu proqramlaşdırma dillərinə aid etmək olmaz. Lakin, bu kitabların arasında elələrinə rast gəlinir ki, onların elmi səviyyəsi bir yana qalsın, hətta Azərbaycan dilində izahı təəssüflər doğurur. Burada nəinki, terminlərdə anlaşılmazlıq vardır (bu daim mübahisə doğuran problemdir), hətta çoxlu dil və üslub qüsurları vardır. Dərin hörmət bəslədiyim zəngin dilimizi təhrif etməyə heç kimin haqqı yoxdur. Mən bu kitab üzərində işləyərkən çalışmışam ki, onu dilimizin formalaşmasında böyük xidmətləri olan *M.P. Vaqifin, M.Ə. Sabirin, S. Vurğunun, C. Məmmədquluzadənin, C. Cabbarlının, Ü. Hacıbəyovun* və başqalarının danışdıqları dilə yaxın bir dildə yazım. Buna nə dərəcədə nail olduğuma Siz hökm verə bilərsiniz. Terminlərə gəldikdə isə onu deyə bilmərəm ki, bu terminləri mən daha düzgün ifadə etmişəm – mən bu fikirdən tamamilə uzağam – lakin, çalışmışam ki, terminləri onların mahiyyətinə uyğun olaraq istifadə edim. Proqramçılar çox yaxşı bilir ki, proqramlaşdırma dillərinin əlifbası klassik latın dilinin əlifbasından ibarətdir. Bizim əlifbamızda isə bu əlifbaya yeni hərflər daxil edilmişdir. Proqramlaşdırma zamanı kompilyator həmin hərflərdən istifadə etməyə icazə vermir. Doğrudur, mən əksər proqramların məlumat xarakterli mətnlərində, proqram sətirlərinə şərhlər yazdıqda, bu mətnləri dilimizdə olduğu kimi yazmışam, lakin, Siz bu proqramları kompilyatorda yığdıqda, həmin hərfləri digər hərflərlə əvəz etməlisiniz. Bəzi hallarda isə buna əməl etmək mümkün olmamış (bu adətən dəyişənlərin adları ilə əlaqədardır) və belə mətnlər proqramın nəticəsi kimi şəkillər formasında verilmişdir. Bu halda "ş" hərfi əvəzinə "sh", "ç" əvəzinə "ch" və ya "c" və s. yazılmışdır. Belə məcburi xarakterli çatışmazlıqlara görə üzr istəyirəm.

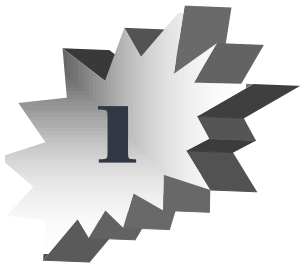
Delphi kimi əhəmiyyətli bir proqramlaşdırma dilinin öyrənilməsində Sizə uğurlar arzulayıram. Kitab haqqında arzu və təkliflərini *zakirmaharramov@rambler.ru* ünvanına göndərən şəxslərə əvvəlcədən dərin minnətdarlığımı bildirirəm.

BIRINCI HISSƏ

Turbo PASCAL 7.0

- **Turbo Pascal dilinin inteqrallaşdırılmış mühiti**
- **Turbo Pascal dilinin əlifbası və strukturu**
- **Verilənlərin tipləri**
- **İfadələr**
- **Operatorlar**
- **Hesablama proseslərinin proqramlaşdırılması**
- **Alt proqramlar**
- **Fayllar**
- **Qrafiklərin proqramlaşdırılması**

Birinci fəsil



TURBO PASCAL DİLİNİN İNTEQRALLAŞDIRILMIŞ MÜHİTİ

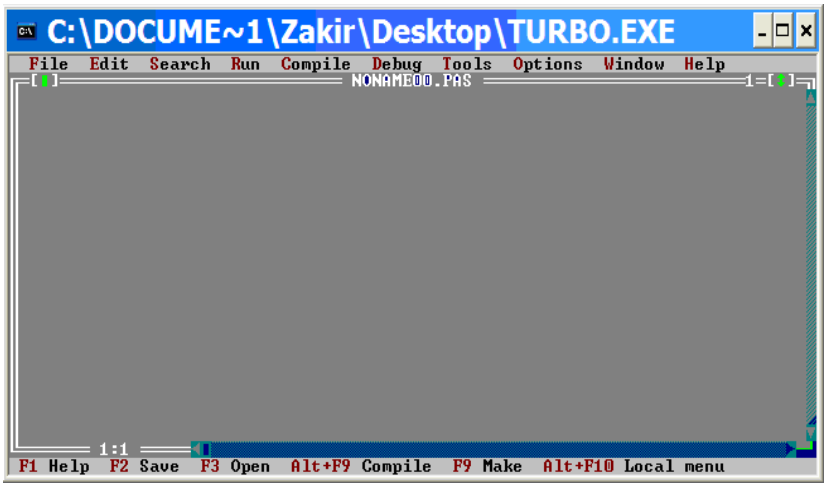
Bu fəsildə Pascal dilinin **Turbo Pascal 7.0, Borland Pascal 7.0, Turbo Pascal for Windows 1.5** və **Borland Pascal for Windows 7.0** versiyalarının inteqrallaşdırılmış mühiti ilə tanış olacaq, yeni proqramların yaradılması, mövcud proqramların açılması, onların sazlanması, kompilyasiyası və yerinə yetirilməsi və habelə inteqrallaşdırılmış mühitin idarə edilməsi ilə əlaqədar əmrləri öyrənəcəyik.

1.1. Turbo Pascal 7.0 dilinin pəncərəsi

Turbo Pascal dilində proqram bu dilin inteqrallaşdırılmış mühitində yaradılır. İnteqrallaşdırılmış mühit dilin kompilyatorundan, proqram mətnini yığmaq və ona düzəlişlər etmək üçün redaktordan və sazlayıcıdan ibarətdir. İnteqrallaşdırılmış mühiti yükləmək üçün **Turbo.exe** proqramını yükləmək lazımdır. Bu zaman ekranda şəkil 1.1 – də göstərilmiş pəncərə açılacaqdır. Bu pəncərə yuxarı hissədə menyu sətrindən, mərkəzi hissədə işçi oblastdan və aşağı hissədə isə vəziyyətlər sətrindən ibarətdir.

Menyu sətri inteqrallaşdırılmış mühitin əmrlərindən ibarətdir. *File* (*Fayl*) menyusu yeni proqram yaratmaq, mövcud proqramları çağırmaq, proqramı yadda saxlamaq və s. kimi əmrlərdən ibarətdir. *Edit* (*Düzəliş*) menyusunda proqrama düzəlişlər etməyə (köçürmək, kəsmək, əlavə etmək və s.) imkan verən əmrlər toplanmışdır. Proqramda olan səhvləri aşkar etmək, onu icra və kompilyasiya etmək üçün *Run* (*Yerinə yetirmək*) və *Compile* (*Kompilyasiya*) menyularından istifadə edilir. Bəzi hallarda menyu sətri görünməyəcəkdir. Belə hallarda menyu sətrini göstərmək üçün ***F10*** klavişini basmaq lazımdır.

İşçi oblastda proqram mətni yığılır. Proqram aşağıdakı qayda ilə yaradılır. İnteqrallaşdırılmış mühit yüklәndikdən sonra, ekranda boş redaktor pəncərəsi açılmalıdır. Əgər bu pəncərə boş deyildirsə, onda yeni proqram mətni yaratmaq üçün *File/New* (*Fayl/Yeni fayl yaratmaq*) əmrini icra etmək lazımdır. Bu zaman pəncərənin yuxarı hissəsində `NONAME00.PAS` adlı yeni boş pəncərə açılacaqdır. Yeni proqram yaratmaq üçün inteqrallaşdırılmış mühit hər dəfə bu adı təklif edir. Təkidlə məsləhət görürük ki, bu adı dərhal dəyişəsiniz. Bunun üçün *File/Save As...* (*Fayl/Faylı necə yadda saxlamaq...*) əmrini icra edərək, yeni açılmış pəncərədə proqrama ad vermək lazımdır. Fayla ad verdikdə, avtomatik olaraq ona *.pas* genişlənmiş hissəsi (kitabın bütün sonrakı fəsillərində biz faylın genişlənmiş hissəsinə *faylın tipi* deyəcəyik) mənimsədiləcəkdir və bu



Şəkil 1.1. Turbo Pascal 7.0 pəncərəsi

fayl kompilyatorun yerləşdiyi qovluqda yerləşəcəkdir.

Bundan sonra, *yeni proqramın mətni yığılır*. Bütün mətn redaktorlarında olduğu kimi, burada da kursordan soldakı simvolu pozmaq üçün *Backspace*, cari simvolu pozmaq üçün *Delete*, yeni sətrə keçmək üçün *Enter*, simvollar registrini dəyişmək üçün *Shift* və baş hərflər rejimini qoşmaq üçün *Caps Lock* klavişlərindən istifadə edilir. Mətnin yığılması zamanı *Edit* (*Düzəlişlər*) menyusunun *Cut* (*Shift+Del – Kəsmək*), *Copy* (*Ctrl+Ins – Köçürmək*), *Paste* (*Shift+Ins –Yerləşdirmək*) və *Clear* (*Ctrl+Del – Pozmaq*) əmrlərindən istifadə etməklə, mətnin yığılması prosesini yüngülləşdirmək olar. Həmçinin hər 10–15 sətirdən bir ***F2*** klavişini basmaqla yığılmış proqram mətnini yadda saxlamaq məsləhət görülür. Bununla da Siz, kompüterdə təsadüfi imtinaların baş verməsi səbəbindən, proqramınızı itirmək təhlükəsindən sığortalanacaqsınız.

İşçi oblastda həm də kompüterdə artıq mövcud olan pascal–proqramların mətninə baxmaq (*.pas* tipli) və onları kompilyasiya etmək olar. Bunun üçün *File/Open...* (*Fayl/Açmaq...*) əmrini icra edib və ya ***F3*** klavişini basıb, açılan pəncərədən faylı seçərək *Open* (*Açmaq*) düyməsini basmaq lazımdır.

Nəhayət, sonuncu *vəziyyətlər sətrində*, proqramçıya müəyyən klavişlər kombinasiyalarının funksiyaları xatırladılır və müəyyən əməllərin yerinə yetirilməsi haqqında məlumatlar təsvir olunur.

İnteqrallaşdırılmış mühitdən çıxmaq üçün *File/Exit* (*Fayl/Çıxmaq*) əmrini icra etmək və ya ***Alt+X*** klavişlərini basmaq lazımdır.

1.2. Proqramın sazlanması

İnteqrallaşdırılmış mühitdə proqram mətni yığıldıqdan sonra, *proqramı sazlamaq* lazımdır. Proqramın sazlanması zamanı proqramda mövcud olan *sintaksis səhvlər* kompilyator tərəfindən aşkar edilir. Proqramı sazlamaq üçün *Compile/Make* əmrini icra etmək və ya ***F9*** klavişini basmaq kifayətdir. Əgər proqramda səhvlər olarsa, kursor həmin mövqedə yerləşəcək və kompilyatorun birinci sətrində həmin səhvin əlaməti haqqında məlumat peyda olacaqdır. Səhvlərin xarakteri haqqında dolğun məlumat almaq üçün inteqrallaşdırılmış mühitin məlumat sisteminə (*Help – Kömək*) müraciət etmək olar.

1.3. Proqramın yerinə yetirilməsi

Proqramda mövcud olan sintaksis səhvlər aradan qaldırıldıqdan sonra, son nəticələr almaq üçün, proqramı icra etmək lazımdır. Bunun üçün *Run* (*Yerinə yetirmək*) menyusundan *Run* (*Yerinə yetirmək*) əmrini icra etmək və ya ***Ctrl+F9*** klavişlərini basmaq lazımdır. Bu zaman ekrana *istifadəçi pəncərəsi* adlanan yeni pəncərə çıxacaqdır. Bu pəncərədə proqram dəyişənlərinə ilkin qiymətlər daxil edilir və nəticələr alınır. Proqramın icrası başa çatdıqdan dərhal sonra kompilyatora qayıdış baş verir, inteqrallaşdırılmış mühit ekrana qayıdır və istifadəçi pəncərəsinin üstünü örtür. Alınmış nəticələri təkrarən görmək və ya onları təhlil etmək üçün istifadəçi pəncərəsini yenidən ekranda göstərmək lazımdır. Kompilyatoru ekrandan müvəqqəti olaraq götürmək üçün *Debug* (*Sazlamaq*) menyusundan *User Screen* (*İstifadəçi pəncərəsi*) əmrini icra etmək və ya ***Alt+F5*** klavişlərini basmaq lazımdır. *Tərtib olunmuş proqram vasitəsilə də istifadəçi pəncərəsini ekranda göstərmək olar.* Bu məqsədlə proqramda sonuncu sətirdən əvvəlki sətirdə, yəni `end.` operatorundan əvvəl `readln` prosedurunu yazmaq lazımdır. Bu halda kompilyator proqrama ilkin verilənin daxil ediləcəyini gözləyir, lakin `readln` prosedurunda heç bir dəyişən yazılmadığı üçün, proqrama heç bir qiymət daxil edilməyəcək və *Enter* klavişi basılana qədər istifadəçi pəncərəsi bağlanmayacaqdır.

Proqramın yerinə yetirilməsi zamanı da kompilyator tərəfindən səhvlər aşkar edilə bilər. Belə səhvlər *semantik* və ya *alqoritm xarakterli* olur, yəni dəyişənlərin aldığı qiymətlər yolverilən həddi aşır, yolverilməz əməliyyatlar (sıfra bölmə, mənfi ədədin loqarifmlənməsi və ya ondan kvadrat kökalma və s.)

baş verir və s. Bu səhvlərə *yerinə yetirmə vaxtının səhvləri* deyilir. Kompilyator yerinə yetirmə vaxtının səhvləri haqqında aşağıdakı məlumatı verir:

```
Run-time error <errnum> at <segment> : <offset>
```

Burada, **<errnum>** – *səhvin kodu*, **:<offset>** – *səhvin baş verdiyi yaddaşın ünvanıdır*.

Turbo Pascal dilinin inteqrallaşdırılmış mühiti proqramın addım–addım yerinə yetirilməsinə də imkan verir. Bu zaman proqramın müxtəlif dəyişənlərinin aldığı real qiymətləri görmək mümkün olur ki, bu da proqramın təhlili üçün əvəzsiz məlumatdır. Proqramı belə seansda icra etmək üçün *Run/Trace into* (*Yerinə yetirmək/Sətirlərlə*) əmrini icra etmək və ya ***F7*** klavişini basmaq lazımdır. Bu seansda proqram sətirləri növbə ilə yerinə yetirilir və hər növbəti sətri yerinə yetirmək üçün ***F7*** klavişini basmaq lazımdır.

Proqramın yerinə yetirilməsi zamanı müşahidə (*Watch*) pəncərəsindən istifadə etmək faydalı ola bilər. *Debug* (*Sazlamaq*) menyusundan *Add Watch* (*Müşahidə üçün əlavə et*) əmrini icra etməklə və ya ***Ctrl+F7*** klavişlərini basmaqla bu pəncərəni ekranda göstərmək olar. Açılan pəncərədə, dəyişənin adını yazmaqla, onun aldığı cari qiyməti görmək olar. Müşahidə pəncərəsinə yeni dəyişənlər əlavə etmək üçün yenidən *Add Watch* əmrini icra etmək lazımdır.

Beləliklə, *Run/Trace into* (*Yerinə yetirmək/Sətirlərlə*) və *Debug/Add Watch* (*Sazlamaq/Müşahidə üçün əlavə et*) əmrləri ilə düzgün işləməyən proqramlarda səhvləri aradan qaldırmaq asanlaşır və proqrama düzgün "diaqnoz" qoymaq mümkün olur.

1.4. Proqramın kompilyasiyası

Proqram sazlandıqdan sonra, onu *kompilyasiya* etmək lazımdır. Kompilyasiya nəticəsində proqramın mətni ikilik verilənlərdən və prosessorun təlimatlarından ibarət maşın kodlarına çevrilir. Kompüter yalnız maşın kodlarından ibarət proqramları icra edir. Proqram mətninin maşın kodlarına çevrilməsi prosesini *kompilyator* adlanan xüsusi proqram yerinə yetirir. Kompilyasiya zamanı `Uses` bölməsində göstərilmiş modullardakı alt proqramlar proqrama əlavə edilir, `Const` və `Var` bölmələrindəki dəyişən və sabitlər üçün yaddaş xanalarına adlar mənimsədilir və s.

Proqramı kompilyasiya etmək üçün *Compile* (*Kompilyasiya*) menyusundan *Compile* (*Kompilyasiya*) əmrini icra etmək və ya ***Alt+F9*** klavişlərini basmaq lazımdır. Proqram uğurla kompilyasiya olunduqdan sonra, ekrana *"Compile successful: press any key"* məlumatı çıxarılacaqdır. Bundan sonra, yeni bir fayl yaranacaqdır ki, bu faylın adı Sizin proqrama verdiyiniz adla eyni olacaq, lakin onun tipi ***.pas*** yox, ***.exe*** olacaqdır. Bu yeni *.exe* tipli faylda proqramın mətnini heç vaxt görə bilməyəcəksiniz və bu fayla düzəlişlər edə bilməyəcəksiniz. Çünki, bu fayl artıq *tam hazır proqram faylıdır* və Turbo Pascal

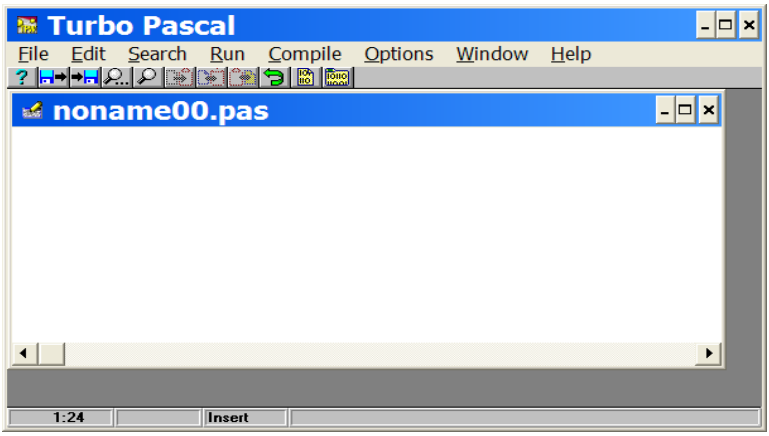
kompilyatorunun mövcud olmasından asılı olmayaraq, o bütün kompüterlərdə yerinə yetirilə biləcəkdir.

Proqramın yerinə yetirilməsini ***Ctrl+Break*** klavişlərini basmaqla dayandırmaq olar.

Qeyd. **Borland Pascal 7.0** inteqrallaşdırılmış mühitinin pəncərəsi Turbo Pascal 7.0 inteqrallaşdırılmış mühitinin pəncərəsi ilə tamamilə eynidir və yuxarıda nəzərdən keçirdiyimiz bütün əmrlər Borland Pascal 7.0 inteqrallaşdırılmış mühitində eyni qayda ilə icra olunur.

1.5. Turbo Pascal for Windows 1.5 dilinin pəncərəsi

Turbo Pascal 7.0 və Borland Pascal 7.0 alqoritmik dilləri, əsasən, *Ms DOS* əməliyyat sistemində işləmək üçün nəzərdə tutulmuşdur. Bununla yanaşı, bu dillər həm də *Ms Windows* əməliyyat sistemində işləyə bilər. Lakin, *Ms DOS* əməliyyat sistemində bu dillər çox dəqiq işlədiyi halda, *Ms Windows* əməliyyat sistemində tam korrekt işləməyə bilər. Ona görə də, Pascal dilinin *Ms Windows* əməliyyat sistemində işləyə bilən **Turbo Pascal for Windows 1.5** (qısaca – **TPW 1.5**) – "*Turbo Pascal Windows üçün*" versiyası yaradılmışdır. TPW 1.5 dilinin inteqrallaşdırılmış mühitinin pəncərəsi şəkil 1.2 – də göstərilmişdir. Yeni proqramın yaradılması, mövcud proqram faylının açılması, proqramın yadda saxlanması, sazlanması, yerinə yetirilməsi və kompilyasiyası və s. bu kimi əmrlər bu mühitdə də eyni qayda ilə yerinə yetirilir. Kəskin fərq ondan ibarətdir ki, bu mühitdə istifadəçi pəncərəsi ayrı bir pəncərə kimi açılır və o adi Windows pəncərələri kimi idarə olunur (zənnimcə, bu mühitin ən böyük üstünlüyü hesab oluna bilər). TPW 1.5 dili ilə işlədikdə unutmayın ki, proqramda `Uses Crt;`

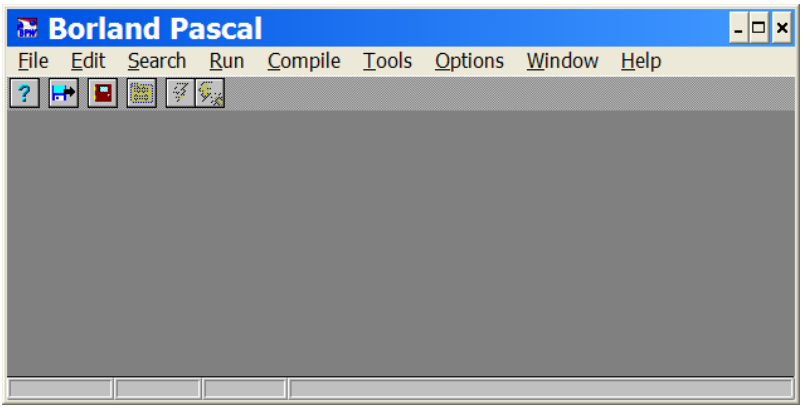


Şəkil 1.2. Turbo Pascal for Windows 1.5 (TPW 1.5) pəncərəsi

əvəzinə **`Uses WinCrt;`** yazılmalıdır və `end.` operatorundan əvvəl `readln` prosedurunu yazmaq lazım deyil.

Turbo Pascal dilində qrafiklərin qurulması ilə əlaqədar yazılmış proqramlar TPW 1.5 mühitində işləməyəcəkdir.

Borland Pascal 7.0 dilinin də Windows versiyası hazırlanmışdır. Bu dil **Borland Pascal for Windows 7.0** (qısaca – **BPW 7.0**) adlanır və onun inteqrallaşdırılmış mühitinin pəncərəsi şəkil 1.3 – də göstərilmişdir. Şəkildən göründüyü kimi bu pəncərə TPW 1.5 pəncərəsindən çox az fərqlənir.



Şəkil 1.3. Borland Pascal for Windows 7.0 pəncərəsi (**BPW 7.0**)

1.6. Delphi–də pascal–proqramların kompilyasiyası

Pascal dilində yazılmış proqramı Delphi mühitində də kompilyasiya etmək olar. Bunun üçün Delphi sistemini yüklədikdən sonra, *File/New* (*Fayl/Yeni fayl yaratmaq*) əmrini seçib, *Other* (*Digər fayllar*) əmri üzərində mausun sol düyməsini basmaq və açılan dialoq pəncərəsində *Console Application (Konsol əlavəsi)* piktoqramı üzərində mausun sol düyməsini yenidən iki dəfə basmaq lazımdır. Bu zaman kod redaktoru pəncərəsində aşağıdakı proqram mətnindən ibarət karkas görəcəksiniz:

```
program Project2;

{$APPTYPE CONSOLE}

uses
     SysUtils;
```

```
Begin

  { TODO -oUser -cConsole Main : Insert code here }
  { Burada proqram sətirləri yazılır }

end.
```

Bu hazır karkasda Siz, istəsəniz, proqramın adını dəyişdirə bilərsiniz. `Uses` bölməsindən sonra `Var`, `Const`, `Type` bölmələrini əlavə edə bilərsiniz. `{$APPTYPE CONSOLE}` *direktivinə isə toxunmayın!* Nümunə üçün aşağıda $y=a+b$ ifadəsinin hesablanması proqramı yazılmışdır.

```
program delphi_pascal;

{$APPTYPE CONSOLE}

uses
  SysUtils;

var
   a,b,y : real;

begin

  Writeln('a,b dəyişənlərinin
         qiymətlərini daxil edin:');
  Readln(a,b);
  y:=a+b;
  Writeln(y);
  Readln;

end.
```

Proqramı kompilyasiya etmək üçün *Run/run* əmrini icra etmək və ya sadəcə olaraq ***F9*** klavişini basmaq lazımdır. Proqramın yadda saxlanması, mövcud proqramın çağrılması əməliyyatları *File* menyusunun əmrləri ilə yerinə yetirilir.

Turbo Pascal dilində modullarla və qrafiklərin qurulması ilə əlaqədar yazılmış proqramlar Delphi mühitində işləməyəcəkdir. Belə proqramların yaradılması kitabın ikinci hissəsində şərh olunmuşdur.

Məsləhət: *Delphi inteqrallaşdırılmış mühitini öyrəndikdən sonra bu bölməyə qayıdın!*

1.7. Turbo Pascal 7.0 inteqrallaşdırılmış mühitində ən çox istifadə edilən əmrlər

Turbo Pascal 7.0 inteqrallaşdırılmış mühitində ən çox istifadə edilən əmrlər cədvəl 1.1 – də göstərilmişdir.

Cədvəl 1.1. Turbo Pascal 7.0 inteqrallaşdırılmış mühitində ən çox istifadə olunan əmrlər

Əmrin adı	Klavişlər	Əmrin yerinə yetirdiyi funksiya
File/New		*Yeni proqram faylı yaratmaq*
File/Open...	**F3**	*Mövcud pascal–proqram faylını açmaq*
File/Save	**F2**	*Proqram mətnini yadda saxlamaq*
File/Save As...		*Proqramı yeni adla yadda saxlamaq*
File/Exit	**Alt+X**	*İnteqrallaşdırılmış mühitdən çıxmaq*
Edit/Cut	**Shift+Del**	*Seçilmiş proqram fraqmentini kəsib buferdə yerləşdirmək*
Edit/Copy	**Ctrl+Ins**	*Seçilmiş proqram fraqmentini buferə köçürmək*
Edit/Paste	**Shift+Ins**	*Buferdə olan proqram fraqmentini kursorla göstərilən mövqeyə yerləşdirmək*
Edit/Clear	**Ctrl+Del**	*Seçilmiş proqram fraqmentini pozmaq*
Compile/Make	**F9**	*Proqramı sazlamaq (səhvləri aşkar etmək)*
Run/Run	**Ctrl+F9**	*Proqramı yerinə yetirmək*
Compile/Compile	**Alt+F9**	*Proqramı kompilyasiya etmək*
Debug/User screen	**Alt+F5**	*İstifadəçi pəncərəsini ekranda göstərmək və ya gizlətmək*
Run/Trace into	**F7**	*Proqramı sətirlərlə yerinə yetirmək*
Run/Program Reset	**Ctrl+F2**	*Proqramın icrasından imtina etmək*
Debug/Add Watch	**Ctrl+F7**	*Dəyişənləri müşahidə pəncərəsinə əlavə etmək*
Ctrl+Break		*Proqramın icrasını dayandırmaq*
	F10	*Menyu sətrini aktivləşdirmək*

Üçüncü fəsil

VERİLƏNLƏRİN TİPLƏRİ

Bu fəsildə Turbo Pascal dilinin tərkibinə daxil olan verilənlərin təsnifatına baxılacaq, sıralı və həqiqi tip verilənlər öyrəniləcəkdir. Tamədədli və həqiqi tiplərin növləri, onların dəyişmə diapazonları və yaddaşda təsviri şərh olunacaq, simvol və məntiqi tiplər izah olunacaqdır. Sadalanan və interval tiplərin proqramda təsviri və bu tiplərin strukturu göstəriləcəkdir. Verilənlərin struktur tiplərinə daxil olan massivlər, sətirlər, çoxluqlar, yazılar və faylların mahiyyəti, onların strukturu, və proqramda təsvirolunma qaydaları ətraflı izah olunacaq və bu izahatlar çoxlu misallar üzərində nümayiş etdiriləcəkdir.

3.1. Tiplərin təsnifatı

İstənilən proqramın əsas elementləri dəyişənlər, sabitlər və operatorlardır. Bu bölmədə biz dəyişən və sabitlərin tiplərini öyrənəcəyik.

Dəyişənlər proqramın yerinə yetirilməsi prosesində müxtəlif qiymətlər alan kəmiyyətlərə deyilir və onlar dəyişənlərin elan edilməsi bölməsində elan edilir. Dəyişənlərdən fərqli olaraq, *sabitlər* proqram boyu öz qiymətlərini dəyişmir və onların qiymətləri sabitlərin elan edilməsi bölməsində təyin olunur. Sabit və dəyişənlərə onların adları ilə müraciət olunur.

Turbo Pascal dilində verilənlərin çoxlu sayda tipləri mövcuddur: bu, şübhəsiz, dilin üstün cəhətidir. Verilənlərin tiplərini öyrənmək üçün onları qruplaşdırmaq məqsədəuyğundur. Hər şeydən əvvəl, verilənlərin tipləri *sadə* və *struktur* tipli olur. *Sadə* tiplər öz növbəsində *sıralı* və *həqiqi* tiplərə bölünür. Turbo Pascal dilində müəyyən edilmiş tiplərin təsnifatı cədvəl 3.1 – də göstərilmişdir.

Tiplər ona görə *sıralı* adlanır ki, əvvəla, dəyişənlərin aldığı qiymətlər sonlu sayda elementlərdən ibarət olur, digər tərəfdən, hər bir elementdən əvvəlki və sonrakı qiymətlər mövcud olur. Başqa sözlə, qonşu qiymətlər bir–birindən bir vahid fərqlənir. Belə ki, misal üçün, *15* tam ədədindən bir vahid çıxdıqda *14*, ona bir vahid əlavə etdikdə isə *16* alınacaqdır. Başqa sözlə, sıralı tip verilənləri nömrələmək mümkündür.

Sıralı tiplərdən fərqli olaraq, *həqiqi* tiplər tam və kəsr hissələrdən ibarət olur, hətta ən məhdud diapazonda yerləşən ədədləri belə nömrələmək mümkün olmur.

Cədvəl 3.1. Turbo Pascal dilində müəyyən edilmiş tiplərin təsnifatı

Qrup	Alt qrup	Adı	İdentifikatoru
Sadə	***Sıralı***	***Qısa tamədədli***	`ShortInt`
		Bayt	`Byte`
		Söz	`Word`
		Tamədədli	`Integer`
		Uzun tamədədli	`LongInt`
		Simvol	`Char`
		Bul	`Boolean`
	Həqiqi	***Həqiqi***	`Real`
		Birqat dəqiqlikli	`Single`
		İkiqat dəqiqlikli	`Double`
		Yüksək dəqiqlikli	`Extended`
		Mürəkkəb	`Comp`
Sətir			`String`
Struktur		***Massiv***	`Array`
		Çoxluq	`Set`
		Fayl	`File`
		Yazı	`Record`
Göstərici			`Pointer`
Prosedur		***Prosedur***	`Procedure`
		Funksiya	`Function`
Obyekt			`Object`

3.2. Verilənlərin sadə tipləri

Bəzi sadə tiplər fiziki (əsas) və ümumi tiplərə bölünür. *Fiziki tiplərin* təməli dil yaradıldıqda qoyulur və kompüterin xüsusiyyətlərindən asılı olmur. *Ümumi tiplər* fiziki tiplərdən hər hansı birinə uyğun gəlir və kompilyator bu tipləri istifadə etdikdə, daha səmərəli kod yaratdığı üçün, onlara daha çox üstünlük verilir.

Bəzi sıralı tipləri proqramçı özü də yarada bilər, ona görə də belə tiplərə istifadəçi tipləri deyilir. İstifadəçi tiplərinə *sadalanan* və *interval* tiplər aiddir.

3.2.1. Tamədədli tiplər

Cədvəl 3.2 – də sadə sıralı tiplərin ala biləcəyi mümkün qiymətlər diapazonu verilmişdir.

Cədvəl 3.2. Tamədədli tiplər

Tipin adı	Dəyişmə diapazonu	Yaddaşda təsviri
ShortInt	(–128) – 127	*işarə ilə 1 bayt*
Integer	(–32 768) – 32 767	*işarə ilə 2 bayt*
LongInt	(–2 147 483 648) – 2 147 483 647	*işarə ilə 4 bayt*
Byte	0 – 255	*işarəsiz 1 bayt*
Word	0 – 65 535	*işarəsiz 2 bayt*

Tam ədədlərin qarşısında "+" və "–" işarələri yazıla bilər. Onlar onluq və onaltılıq say sistemlərində təsvir oluna bilər. Tam ədədləri onaltılıq say sistemində təsvir etmək üçün ədədin qarşısında **$** işarəsi qoyulmalıdır. Belə ədədlər `$00000000 – $FFFFFFFF` diapazonunda yerləşməlidir.

Misal.

```
Var i,j:byte;
    n:word;x6,ss:integer;
```

3.2.2. Həqiqi tiplər

Həqiqi tiplər və onların ala biləcəyi mümkün qiymətlər diapazonu cədvəl 3.3 – də göstərilmişdir:

Cədvəl 3.3. Həqiqi tiplər

Tipin adı	Dəyişmə diapazonu	Yaddaşda təsviri
Real	$(-1{,}7\cdot10^{38}) – (-2{,}9\cdot10^{-39})$, $2{,}9\cdot10^{-39} – 1{,}7\cdot10^{38}$	*6 bayt*
Single	$(-3{,}4\cdot10^{38}) – (-1{,}5\cdot10^{-45})$, $1{,}5\cdot10^{-45} – 3{,}4\cdot10^{38}$	*4 bayt*
Double	$(-1{,}7\cdot10^{308}) – (-5\cdot10^{-324})$, $5\cdot10^{-324} – 1{,}7\cdot10^{308}$	*8 bayt*
Extended	$(-1{,}1\cdot10^{4932}) – (-1{,}9\cdot10^{-4951})$, $1{,}9\cdot10^{-4951} – 1{,}1\cdot10^{4932}$	*10 bayt*
Comp	$(-2^{63}+1) – (2^{63}-1)$	*8 bayt*

Həqiqi ədədlər, bütün dillərdə olduğu kimi, burada da qeyd olunmuş və sürüşkən vergüllü formalarda təsvir olunur və əlavə izahata ehtiyac görmürük.

`Comp` tipi həqiqi tipə aid olsa da, $(-2^{63}+1) - (2^{63}-1)$ aralığında tam ədədləri təsvir edir. Bu tipli dəyişənə həqiqi tipli qiymətlər mənimsətdikdə o, avtomatik olaraq, yaxın tam ədədə qədər yuvarlaqlaşdırılır.

Misal.

```
Var a,b : real;
    d : double; x,s : extended;
```

3.2.3. Simvol tiplər

Simvol tipli dəyişən və sabitlər **ASCII** (*American Standart Code for Information Interchange*) kodunun simvollar çoxluğundan yalnız bir qiymət (işarə, hərf, rəqəm) ala bilər. Bu tip dəyişənlər **`char`** işçi sözü ilə müəyyənləşdirilir:

```
Var a,b,c : char;
```

Simvol tipli dəyişənlər sıralı tiplərə aid olmaqla, yalnız bir simvoldan ibarət olur və proqram daxilində onlara qiymətlər mənimsətmək olar:

```
a:= 't';
b:= '$';
c:= '9';
```

Kompüterdə hər bir simvola *0 – 255* aralığından bir tam ədəd uyğun gəlir. Hər hansı simvolun kodunun qiymətini **`Ord`** funksiyasının köməyi ilə almaq mümkündür. Əks əməliyyat isə **`Chr`** funksiyası ilə yerinə yetirilir.

Misal.

```
Ord('A') = 65;
Ord('a') = 97;
Chr(65) = 'A';
Chr(100) = 'd';
```

`Chr` funksiyasını **#** işarəsi ilə də əvəz etmək olar:

```
#65='A';
#97='a'.
```

3.2.4. Məntiqi tiplər

Turbo Pascal dilində bir neçə məntiqi tipin olmasına baxmayaraq, proqramda **`Boolean`** tipini istifadə etmək məqsədəuyğundur. Bu tip verilənlər yalnız iki qiymət ala bilər: *True* (Doğru) və *False* (Yalan). Məntiqi tiplər də sıralı tiplərə aiddir.

Misal.

```
Var a1, b2 : boolean;
```

```
d7, s44 : boolean;
```

3.3. Sadalanan tiplər

Sadalanan tiplərdə verilənlərin qiymətləri birbaşa sadalanır. Qiymətlər bir–birindən vergül işarəsi ilə ayrılmaqla mötərizə daxilində yazılır. Sadalanan tiplərin ümumi forması belədir:

Type *tipin adı* **=** **(***1–ci ad, 2–ci ad, . . . , n–ci ad***);**

Misal.

```
Type Color = (red, orange, yellow, blue);
Var   c1,c2 : color;
      Heyvanlar : (Fil, At, Aslan);
```

Bu misalda, `Color` tipi aşkar təsvir olunmuş və onun üçün qiymətlər kimi rənglər müəyyənləşdirilmişdir. `c1`, `c2` dəyişənləri bu sadalanan rənglərdən birini ala bilər: onlara digər qiymətlər vermək olmaz. İkinci tip anonim müəyyənləşdirilmişdir (adı yoxdur) və `heyvanlar` dəyişəni `Fil`, `At` və `Aslan` qiymətlərindən birini ala bilər. Qeyd edək ki, sadalanan tiplər sıralı tiplərə aiddir. Belə ki, `red`, `orange`, `yellow` və `blue` qiymətlərinin sıra nömrələri uyğun olaraq *0, 1, 2* və *3* – dür.

Sadalama proqramçıya verilənlərin yeni tiplərini təsvir etməyə imkan verir.

Sadalanan tipin bütün elementlərinin identifikatorları sabit kimi interpretasiya olunur. Bu identifikatorlar sətir sabitləri olmadığı üçün dırnaq işarəsi daxilində yazılmır. Qeyd etmək lazımdır ki, eyni identifikatorun müxtəlif tiplərdə təsviri səhvdir. Məsələn:

```
program tekrar;
type
   Hafta_1=(Mon, Tue, Wed, Thu, Fri, Sat, Sun);
   Hafta_2=(Mon, Tue, Wed, Thu, Fri);
begin
...
end.
```

Bu proqramı icra etsək, kompilyator *"Error: Duplicate identifier* (`Mon`)*"* məlumatı verəcəkdir.

Proqramda sadalanan tipli dəyişəndən istifadə etdikdə aşağıdakıları nəzərə almaq lazımdır:

- sadalanan tiplər sıra tipinə aiddir;
- `Read`, `Readln`, `Write` və `Writeln` prosedurlarında arqument kimi sadalanan tipin qiymətlərindən istifadə etmək olmaz;
- sadalanan tiplərdə yalnız münasibət əməliyyatlarından istifadə etmək olar;
- sadalanan tip dəyişənlərdən mənimsətmə operatorlarında, massivin indekslərində və `for` operatorunun sərhədlərində istifadə etməyə icazə

verilir.

3.4. İnterval tiplər

Dəyişənlər interval tipləri ilə təsvir olunduqda dəyişmə intervalından istifadə olunur. Belə ki, bu dəyişmə intervalı dəyişənin sərhəd qiymətlərini göstərən iki sabitlə müəyyən edilir. İnterval tiplər yalnız sıralı tip dəyişənlərə verilə bilər, başqa sözlə, sərhəd qiymətləri həqiqi tiplər istisna olmaqla, istənilən sadə tiplər ola bilər. Lakin hər iki sabit eyni tipli olmalıdır. İnterval tiplərin ümumi forması belədir:

`Type` *tipin adı* **=** *1–ci sabit* **`..`** *2–ci sabit*;

Misal.

```
Const m = 10; n = 100;
Type otrezok = 1..25;
     Sem = m..n;
     Simvol = 'a'..'z';
Var a,b : otrezok;
     a1,a2 : sem;
     bukva : simvol;
```

`a` və `b` dəyişənləri `otrezok` tipli elan olunur və onlar *1*–dən *25*–ə qədər diapazonda qiymətlər ala bilər; `a1` və `a2` dəyişənləri `sem` tipli elan olunur və *10*–dan *100*–ə qədər qiymətlər ala bilər; `bukva` dəyişəni isə `simvol` tipli elan olunmaqla kiçik latın hərflərini ala bilər.

Misaldan və izahatdan göründüyü kimi, interval tiplər də sıralı tipə aiddir. Proqramın icrası zamanı interval tipli dəyişənlərin qiymətləri göstərilən sərhəd qiymətlərindən kənara çıxarsa, bu barədə məlumat verilməyəcək və onun qiyməti səhv olacaqdır.

3.5. Verilənlərin struktur tipləri

Cədvəl 3.1 – də göstərildiyi kimi, struktur tiplərə aşağıdakılar aiddir:

- *massivlər*;
- *sətirlər*;
- *çoxluqlar*;
- *yazılar*;
- *fayllar*.

Bu tiplərlə tanış olaq.

3.5.1. Massivlər

Massiv eyniadlı və eynitipli indeksli elementlərin yığımına deyilir. Massivin elementləri, struktur tip də daxil olmaqla, istənilən tipli ola bilər, lakin eyni bir massivin elementləri eyni tipli olmalıdır. Massivlər, özlərinin adları və

indeksləri ilə müəyyən olunur. Massivin indeksi onun elementlərinin sıra nömrəsini göstərir. İndekslərin sayı isə massivin ölçüsünü müəyyən edir. Əgər bir indeks yazılarsa, massiv birölçülü, iki indeks yazılarsa ikiölçülü və s. olur. Birölçülü massivlər riyaziyyatda vektorlara, ikiölçülü massivlər isə matrislərə uyğun gəlir. Çoxölçülü paralelepipedləri massivlərin həndəsi obrazı hesab etmək olar. Massivin indeksləri sıralı tip olmalıdır. Eyni bir massivin müxtəlif indeksləri müxtəlif tip ola bilər. Massivlərin indeksləri daha çox tamədədli tipli olur. İndekslər dəyişən və ifadələr də ola bilər. Massivə müraciət etmək üçün onun adını və kvadrat mötərizə daxilində yazılmış indekslərini göstərmək lazımdır, məsələn, `a[5,8]`, `s[16]`, `x[i,j]`, `b[m+n]` və s.

Massivlərin ölçüləri elanetmə bölməsində əvvəlcədən müəyyənləşdirilir. Proqramın yerinə yetirildiyi müddətdə onların ölçüləri dəyişmir. Belə massivlərin ümumi təsvir forması aşağıdakı kimidir:

Massivin adı **: Array [***indekslər***] of** *elementlərin tipi***;**

Burada, *indekslər* interval tip ilə göstərilir.

Misal.

```
Const max = 1000;
Type a = array[1..5,1..8] of integer;
Var x, y : a;
   b : array[1..50] of real;
   c : array[1..40] of char;
   d : array['a'..'z'] of integer;
   z : array[1..max] of boolean;
```

Bu misala əsasən, `x` və `y` dəyişənləri *40* elementdən – *5* sətir və *8* sütundan ibarət ikiölçülü massiv olur; massivin elementləri `integer` – tam tiplidir. `b` dəyişəni *50* həqiqi tipli elementdən, `c` dəyişəni *40* simvol tipli elementdən, `d` dəyişəni *26* tam ədədlərdən, `z` dəyişəni isə *1000* məntiqi tipli elementlərdən ibarət massivlər kimi təyin olunur.

Massivlərin elementlərinə `const` operatoru vasitəsilə də qiymətlər vermək olar. Bu halda massivin elementləri adi mötərizə daxilində, bir–birindən vergüllə ayrılmaqla verilir. Çoxölçülü massivlərdə isə xarici adi mötərizə sol indeksə, daxili adi mötərizə isə növbəti indeksə və s. aid olur.

Misal.

• *həqiqi ədədlərdən ibarət birölçülü massiv*

```
const
     vektor: array[1..7] of real = (0.25,
          3.21,6.37,9.91,71.06,67.9,37.6);
```

• *tam ədədlərdən ibarət ikiölçülü massiv*

```
const
     m2: array[1..3,1..4] of integer =
     ((1,2,3,4),(5,6,7,8),(9,10,11,12));
```

• *tam ədədlərdən ibarət üçölçülü massiv*

```
Const
   m3: array[1..4,1..3,1..2] of Byte =
   (((1,2),(3,4),(5,6)),((7,8),(9,10),(11,12)),
   ((13,14),(15,16),(17,18)),((19,20),(21,22),(23,24)));
```

• *simvollardan ibarət birölçülü massiv*

```
const
     CharVect1: array[1..6] ofchar =
     ('P','A','S','C','A','L');
```

və ya

```
      CharVect2: array [1..6] of char = 'PASCAL'.
```

Bu misallardakı ikiölçülü `m2` və üçölçülü `m3` massivlərinin elementlərinə qiymətlərin mənimsədilməsinin interpretasiyası şəkil 3.1 – də göstərilmişdir.

Massivləri tətbiq etdikdə aşağıdakı məhdudiyyətlərə əməl etmək lazımdır.

Massivlərin indeksləri, `LongInt` tipindən başqa, istənilən sıralı tip ola bilər.

Massivin elementlərinin sayı

$$n=\frac{65520}{p}$$

düsturu ilə təyin olunur. Burada, p – massivin elementlərinin baytlarla ölçüsüdür. $p=1$ olduqda massivin elementlərinin sayı *65520* olacaqdır. Bu düstur verilənlər seqmentinin ölçüsü ilə bağlıdır. Proqramda dəyişən və sabitlər bu seqmentdə saxlanır. Bu seqmentin ölçüsü *64 Kb* və ya *65536* baytdır. Turbo Pascal dilində istənilən tip dəyişənin maksimal uzunluğu *65520* baytdır. Ona görə də, massivin ölçüsünü müəyyən etdikdə, bu məhdudiyyəti nəzərə almaq lazımdır. Məsələn,

```
Var
  a: array[1..200,1..100] of extended;
```

təsviri ilə, `a` massivinin yaddaşda yerləşdirilməsi üçün, *200∗100∗10* bayt yaddaş tələb olunur ki, bu da *64 Kb* – dan çoxdur (çünki, `Extended` tipli dəyişən yaddaşda *10 bayt* yer tutur). Bu Turbo Pascal dilinin nöqsanıdır və bu nöqsanı başqa üsullarla aradan qaldırmaq mümkündür.

```
m2(1,1)= 1  m2(1,2)= 2  m2(1,3)= 3  m2(1,4)= 4
m2(2,1)= 5  m2(2,2)= 6  m2(2,3)= 7  m2(2,4)= 8
m2(3,1)= 9  m2(3,2)=10  m2(3,3)=11  m2(3,4)=12

m3(1,1,1)= 1 m3(1,1,2)= 2 m3(1,2,1)= 3 m3(1,2,2)= 4 m3(1,3,1)= 5 m3(1,3,2)= 6
m3(2,1,1)= 7 m3(2,1,2)= 8 m3(2,2,1)= 9 m3(2,2,2)=10 m3(2,3,1)=11 m3(2,3,2)=12
m3(3,1,1)=13 m3(3,1,2)=14 m3(3,2,1)=15 m3(3,2,2)=16 m3(3,3,1)=17 m3(3,3,2)=18
m3(4,1,1)=19 m3(4,1,2)=20 m3(4,2,1)=21 m3(4,2,2)=22 m3(4,3,1)=23 m3(4,3,2)=24
```

Şəkil 3.1. Massivlərin elementlərinə qiymətlərin mənimsədilməsi

3.5.2. Sətirlər

Sətirlər **String** tipi ilə müəyyənləşdirilir. String tipli sətirlər sətir simvollarından təşkil olunmuş birölçülü massivin xüsusi bir formasıdır və ümumi uzunluğu *255* – dən çox olmayan simvollar ardıcıllığından ibarətdir. Sətirlərin ümumi uzunluğu kvadrat mötərizədə göstərilir.

Misal.

```
var
  S1: string[12];
  S2: string[128];
  Smax: string;
```

Təsvirdə sətrin uzunluğu göstərilməzsə, onda susmaya görə *255* simvola bərabər maksimal uzunluq götürülür.

Sətirlər massivlərə uyğun gəldiyindən, sətrin istənilən simvoluna massivin elementi kimi müraciət etmək olar. Bunun üçün dəyişənin adının yanında, kvadrat mötərizə daxilində, simvolun nömrəsini göstərmək lazımdır. Massivin sıfırıncı elementi idarəedici element olmaqla, sətir tipli dəyişənin faktiki uzunluğunu göstərir. Əgər sətir dəyişəni

```
Var ad: string[4];
```

kimi elan olunarsa, ad dəyişəni *4* simvoldan çox qiymət ala bilməz, ona görə də proqramda

```
ad:='Ada';
```

yazılışı düzgün qəbul olunacaq, lakin

```
ad:='Pascal';
```

yazılışında isə ad dəyişəninin faktiki qiyməti 'Pascal' yox, 'Pasc' olacaqdır.

Əgər, proqramda ad:='abcd'; yazsaq, onda sıfırıncı element ad[0] – sətirdə simvolların ümumi sayını göstərir, yəni ord(ad[0])=4 olacaqdır. Massivin növbəti elementləri isə ad[1]='a', ad[2]='b', ad[3]='c' və ad[4]='d' olacaqdır. Göründüyü kimi, proqramçı sətir tipli dəyişənin istənilən simvoluna müraciət edə bilər.

Misal.

```
Var a,b: string;
  begin
    a:= 'Mən kitab oxuyuram';
    a[5]:= 'q';
    a[6]:= 'ə';
    a[7]:= 'z';
    a[8]:= 'e';
    a[9]:= 't';
    b:= a;
  end.
```

Bu proqrama əsasən, a sətir dəyişəninin qiyməti `Mən kitab oxuyuram` olduğu halda, b dəyişəninin qiyməti `Mən qəzet oxuyuram` olacaqdır.

3.5.3. Çoxluqlar

Proqramlaşdırmada "çoxluq" tipi riyaziyyatdakı çoxluq anlayışına uyğun olaraq istifadə olunur. Fərq ondadır ki, Turbo Pascal dilində çoxluğun elementləri yalnız sıra tipli olmalıdır. Çoxluq əvvəlcədən müəyyən edilmiş qiymətlərdən seçilmiş elementlər yığımıdır. Bundan başqa, çoxluğun yuxarı və aşağı sərhədlərinin qiymətləri *0* ilə *255* intervalında olmalıdır. Buna görə də çoxluqda `ShortInt`, `Integer`, `LongInt` və `Word` tiplərindən istifadə etmək olmaz. Çoxluq tipli dəyişənin aldığı qiymətlər kvadrat mötərizə daxilində göstərilir. Boş çoxluq **`[ ]`** kimi işarə olunur.

Çoxluq tipini təyin etmək üçün **`set`** və **`of`** işçi sözlərindən istifadə olunur və sonra bu çoxluğun elementləri göstərilir. Çoxluq tip dəyişənlərin ümumi təsvir forması belədir:

`Set of` *çoxluğun elementləri*;

Çoxluq tipinin elementləri ya tipə uyğun ayrı sabitlər kimi sadalanır, ya da bir–birindən **`..`** simvolları ilə ayrılan interval qiymətləri ilə təsvir olunur.

Misal.

```
Type
  eded = set of 0..9; { 0–dan 9–a kimi rəqəmlər çoxluğu }
  simvol = set of '0'..'9'; {'0'–dan '9'–a kimi simvollar çoxluğu }

Const
  eded1 : eded = [0, 2, 4, 6, 8];
  eded2 : eded = [1..3, 5..7];
  simvol1: simvol = ['0', '2', '4','6', '8'];
  simvol2: simvol = ['0'..'3','5'..'7'];
  simvollar: Set of Char = ['a'..'z','A'..'Z'];
```

Bu misalda, `type` bölməsində, iki çoxluq tipi müəyyənləşdirilmişdir: `eded` – *0*–dan *9*–a kimi rəqəmlər çoxluğu və `simvol` – *'0'*–dan *'9'*–a kimi simvollar çoxluğu. `Const` bölməsində `eded1` və `eded2` sabitləri `eded` tipli təyin olunmuş və onlara həmin çoxluqdan qiymətlər mənimsədilmişdir. Analoji qayda ilə `simvol1` və `simvol2` sabitləri `simvol` tipli təyin olunmuş və onlara həmin çoxluqdan qiymətlər mənimsədilmişdir. `simvollar` sabiti isə birbaşa `Const` bölməsində çoxluq kimi təyin olunmuş və bu çoxluğun elementləri latın əlifbasının bütün hərflərindən ibarətdir.

Çoxluqlar üzərində əməliyyatlara "*İfadələr*" bölməsində baxacağıq.

3.5.4. Yazılar

Yazılar eyni bir suala aid olan müxtəlif verilənləri sadə üsulla birləşdirməyə imkan verir. Başqa sözlə, yazılar da massivlər kimi verilənlər yığımından ibarətdir, lakin massivlərdən fərqli olaraq, yazılar müxtəlif tipli verilənlərdən təşkil olunur. Massivin isə bütün elementləri həmişə eynitipli olur. Yazılara misal olaraq işə qəbul olunan şəxsin anketini misal göstərmək olar. İşçi xüsusi blankda ad, ünvan, yaş, ailə vəziyyəti və s. kimi suallara cavab verməlidir. Aydındır ki, bu sualların bəzilərinə sözlərlə, bəzilərinə tam ədədlə, bəzilərinə isə məntiqi sabitlərlə (*hə – True*, *yox – False*) cavab vermək lazım gəlir. Əlbəttə, belə müxtəlif tipli verilənləri bir massivin elementləri kimi təsvir etmək qeyri–mümkündür. Digər tərəfdən, bu verilənlər nə qədər müxtəlif tipli olsalar da onlar bir nəfərə aiddir. Ona görə də belə verilənləri təsvir etmək üçün yazılardan istifadə edilir.

Müxtəlif verilənlər massivin elementləri adlandırıldığı halda, yazılar sahələrdən ibarət olur. Massivin ayrı–ayrı elementləri üzərində əməliyyatlar aparmaq mümkün olduğu kimi, yazıların da ayrı–ayrı sahələri üzərində əməliyyatlar aparmaq olar. Hər bir sahənin adı olur və yazı daxilində bu ad təkrarlana bilməz.

Yazılar iki növ olur: qeyd olunmuş hissəli və variantlı.

Qeyd olunmuş hissəli yazılar sonlu sayda sahələrdən ibarətdir. Onun elan edilməsinin ümumi forması belədir:

Record

1–ci sahənin adı **:** *sahənin tipi***;**

...

n–ci sahənin **:** *sahənin tipi***;**

end;

Yazının sahələrinə müraciət etmək üçün, aralarında nöqtə işarəsi qoymaqla, yazının adını və sahənin adını yazmaq lazımdır. Sahə üzərində onun tipinin yol verə biləcəyi əməliyyatları aparmaq olar.

Misal.

```
Var Persone: record
      Name:string;
      Address:string;
      Married:boolean;
      Salary:real;
    end;
  ...
  Persone.Name:='Abdullayev R.K.';
  Persone.Address:='Səməd Vurğun 31';
  Persone.Married:=True;
  Persone.Salary:=500;
```

Bu misalda, `Persone` yazının adıdır və onun dörd sahəsi müəyyənləşdirilmişdir: adı (`Name` – sətir tipli), ünvanı (`Address` – sətir tipli), ailə vəziyyəti (`Married` – məntiqi tipli) və əmək haqqı (`Salary` – həqiqi tipli). Proqramın icrası blokunda isə həmin sahələrə müvafiq qiymətlər mənimsədilmişdir. Sahələrə qiymətlər mənimsətmək üçün yazının öz adı göstərilməlidir, ona görə də proqramda, məsələn,

`Persone.Address:='Səməd Vurğun 31';` yazılmışdır.

Misal.

```
type
 Complex = record
   re: real;
   im: real;
 end;
 Tarix = record
   il: integer;
   ay: 1..12;
   gun: 1..31;
 end;
```

Yazı tipi təsvir edildikdən sonra, bu tipin dəyişənləri və ya tipləşdirilmiş sabitləri verilə bilər. Yazı tipli sabitlərin təsvirində, yazının bütün sahələrinin qiymətləri ilə bərabər, onların identifikatorları da göstərilir. Yazı tipli tipləşdirilmiş sahələrdən istifadə etməyə icazə verilmir.

Yuxarıda təsvir edilmiş yazı tipindən sonra, aşağıdakı dəyişən və sabiti təyin etmək olar:

```
var
 X,Y,Z: complex;
 Tar: tarix;
const
 İngilt: tarix=(il: 1951; ay: 12; gün: 19);
```

Sahələrə müraciət etdikdə, hər dəfə yazının adını təkrarən yazmamaq üçün, `With` operatorundan istifadə olunur. Bu operatoru növbəti bölmələrdə öyrənəcəyik.

Variantlı yazılar da sonlu sayda sahələrdən ibarət olur, lakin yaddaşda sahələrin tutduğu yeri müxtəlif cür interpretasiya etməyə imkan verir. Yazının bütün variantları yaddaşın eyni bir hissəsində yerləşir və onlara müxtəlif adlarla müraciət etmək mümkün olur.

Variantlı yazının ümumi forması belədir:

Record
 Case *əlamət* **:** *əlamətin tipi* **of**
  1–ci variant **:** *variantın təsviri***;**
  . . .
  n–ci variant **:** *variantın təsviri***;**
end;

Qeyd olunmuş hissəli yazılar bir və ya bir neçə sahədən ibarət olur ki, hər bir sahənin adı və tipi onların təsvirində göstərilir. Bunu tələbələrin müvəffəqiyyətini əks etdirən yazıda göstərək.

Misal. Tələbələrin müvəffəqiyyətini əks etdirən yazı.

```
Type
  Str6 = String [6];
  Str20 = String [20];

  Sqiymat = record
     Aliriy:Byte;     { Ali riyaziyyat }
     Tarix :Byte;     { Tarix }
     İnform:Byte;     { İnformatika }
     Fizika:Byte;     { Fizika }
  end;

  Talaba = record
     Soyad:Str20;     { Soyadı}
     Ad   :Str20;     { Adı}
     Atasi:Str20;     { Atasını adı }
     İl   :Integer;   { Doğulduğu il }
     Unvan:Str20;     { Ünvan}
     Grup :Str6;      { Qrupun şifri }
     Qiymat:Sqiymat   { Sonuncu semestrin qiymətləri }
  end;
```

Bu yazıya baxdıqda görürük ki, buraya yalnız tələbənin bir semestrinin qiymətləri daxil edilmişdir. Hər semestrdə fənlər dəyişdiyindən, bu yazıya bütün fənlər daxil edilməlidir. Bu isə lazımsız informasiyanın saxlanmasına gətirir. Bundan başqa, hər bir tələbə üçün yaddaşda bütün fənlər üçün yer ayrılır. Ona görə də belə hallarda variantlı yazılardan istifadə etmək məqsədəuyğundur. Bu yazıda da, qeyd olunmuş hissəli yazılarda olduğu kimi, bütün mümkün sahələr təsvir olunur. Amma, yaddaşda cari halda lazım olan variant üçün yer ayrılır. İki semestrin fənlərini nəzərə alsaq, yuxarıdakı yazının variantlı formasını aşağıdakı kimi yaza bilərik:

Misal. Variantlı yazı.

```
Type
  Str6 = String [6];
  Str20 = String [20];

  Sqiymat1 = record
    Aliriy1 :Byte;    { Ali riyaziyyat }
    Tarix   :Byte;    { Tarix }
    İnform1 :Byte;    { İnformatika }
    Fizika  :Byte;    { Fizika }
```

```
  end;

  Sqiymat2 = record
    Aliriy2 :Byte;    { Ali riyaziyyat }
    Electr  :Byte;    { Elektronika }
    İnform2 :Byte;    { İnformatika }
    İntexn  :Byte;    { İnformasiya texnologiyası }
  end;

  Talaba = record
    Soyad :Str20;     { Soyadı }
    Ad    :Str20;     { Adı }
    Atasi :Str20;     { Atasını adı }
    İl    :Integer;   { Doğulduğu il }
    Unvan :Str20;     { Ünvan }
    Grup  :Str6;      { Qrupun şifri }
```

{ *Yazının variant hissəsi* }

```
    case Semestr:Byte of
    1 : (Qiymat1:Sqiymat1);   { Birinci semestrin qiymətləri }
    2 : (Qiymat2:Sqiymat2);   { İkinci semestrin qiymətləri }
  end;
```

Gördüyümüz kimi, variantlı yazı iki hissədən ibarətdir. Birinci hissə qeyd olunmuş hissəli yazı, ikinci hissə isə bir neçə variantdan ibarət variant hissəsidir. Burada `Talaba` yazısının `Semestr` sahəsi iki – `Qiymat1` və `Qiymat2` kimi alternativ variantlardan təşkil olunmuşdur.

Qeyd. Yazı tip dəyişənlər mənimsətmə operatorlarında iştirak edə bilər, lakin onlar üzərində heç bir əməliyyatlar aparıla bilməz. Hesabi və digər əməliyyatlar yalnız yazı sahələri üzərində yerinə yetirilə bilər.

3.5.5. Fayllar

Fayl eynitipli elementlərin xarici qurğularda – disklərdə tutduğu adlı yerə deyilir. Faylın diskdə nə qədər yer tutmasını əvvəlcədən göstərmək lazım deyildir. Diskdə yerləşən fayl üzərində hər hansı bir əməliyyat aparmaq üçün proqramda fayl dəyişənindən (məntiqi fayldan) istifadə olunur. Fayl dəyişəni proqramda təsvir olunduqdan sonra, əgər ona müraciət olunarsa, diskdəki faylla əlaqə yaranır, fayl üzərində əməliyyat aparılır və ona müvafiq dəyişikliklər edilir. Əməliyyat qurtardıqda faylla əlaqə kəsilir. Bundan sonra, fayl dəyişəni həmin tip başqa faylla əlaqə yarada bilər.

Elementlərin tiplərindən asılı olaraq üç növ fayl mövcuddur:

- *mətn tipli*;
- *tipləşdirilmiş*;
- *tipləşdirilməmiş.*

Mətn tipli fayllar müxtəlif uzunluqlu simvollar sətirlərindən ibarət olur və onların tipi **`Text`** işçi sözü ilə müəyyənləşdirilir. *Tipləşdirilmiş fayllar* proqramda göstərilən tipli (fayl tipi istisna olmaqla) elementlərdən ibarət olur və **`File of`** işçi sözü ilə müəyyənləşdirilir. *Tipləşdirilməmiş fayllar* isə tipləri göstərilməmiş elementlərdən ibarət olur və **`File`** işçi sözü ilə müəyyənləşdirilir. Fayl dəyişəninin tipi faylın elementlərinin tipinə uyğun olmalıdır.

Misal.

```
Var
Metn_fayli:Text;                    {Mətn tipli }
Tam_ededli_fayl:File of integer;    {Tipləşdirilmiş – tamədədli }
Heqiqi_ededli_fayl:File of real;    {Tipləşdirilmiş – həqiqi ədədli }
Tipsiz_fayl:File;                   { Tipləşdirilməmiş }.
```

Fayllarla əlaqədar əməliyyatları səkkizinci fəsildə öyrənəcəyik.

Qeyd. Pascal dilində mətn tipli fayllar `Text` sözü ilə təsvir olunur. Delphi–də isə bir sıra komponentlər `Text` xassəsinə malik olduğundan, çaşqınlıq yaranmaması üçün, mətn tipli faylların təsvirində **`TextFile`** və ya **`System.Text`** yazmaq lazımdır.

Dördüncü fəsil

İFADƏLƏR

Bu fəsildə hesabi və məntiqi ifadələri, münasibət əməliyyatlarını öyrənəcəyik. Bu ifadələr üzərində yerinə yetirilən əməliyyatlar və onların yerinə yetirilmə ardıcıllığı şərh olunacaq, ənənəvi hesab əmələri ilə yanaşı, tamədədli bölmə əməllərini, bitlər üzərində əməliyyatları öyrənəcəyik. İfadələrdə ən çox istifadə olunan funksiyaların Pascal dilində yazılışı, bir sıra hesab əməllərini icra edən standart funksiyalara müraciət, sətir ifadələri və onlar üzərində yerinə yetirilən əməliyyatlar, bu əməliyyatları yerinə yetirən standart funksiya və prosedurlar, ekranla işləmək üçün prosedurlar izah ediləcəkdir. Çoxluqlar üzərində əməliyyatlar öyrəniləcəkdir. Bütün bu məsələlər çoxlu misallar üzərində izah ediləcəkdir.

4.1. İfadələr üzərində əməliyyatlar

İfadələr proqramlaşdırmada ən çox istifadə olunan konstruksiyalardır. Riyaziyyatda ifadələr adətən düsturlarla təsvir olunduğu halda, proqramlaşdırmada hər hansı əməliyyatın təyini üçün istifadə edilir. İfadələr sabit, dəyişən, əməliyyat işarələri, adi mötərizələr və funksiyalardan təşkil olunur.

İstifadə olunan verilənlər və əməliyyatların tiplərindən asılı olaraq ifadələr *hesabi*, *məntiqi* və *sətir* tipli olur. Əməliyyatlar özləri isə aşağıdakı qruplara bölünür:

- *Hesabi əməliyyatlar:*
  `+, –, *, /, div, mod`
- *Münasibət əməliyyatları:*
  `=, <>, <, >, <=, >=`

- *Məntiqi (bul) əməliyyatlar:*
  not, and, or, xor
- *İnformasiya bitləri üzrə əməliyyatlar:*
  not, and, or, xor, shl, shr
- *Sətir əməliyyatı*
  + və ya **konkatenasiya**
- *Çoxluqlar üzrə əməliyyatlar:*
  +, –, *, in, <=, >=

Əməliyyatlar aşağıdakı ardıcıllıqla yerinə yetirilir:

1. **Not**
2. ***, /, div, mod, and, shl, shr**
3. **+, –, or, xor**
4. **=, <>, <, <=, >=, in**

İlk növbədə mötərizədaxili ifadələr hesablanır. Eyni prioritetli bir neçə ardıcıl əməliyyatlar soldan sağa istiqamətdə ardıcıl olaraq yerinə yetirilir.

4.2. Hesabi ifadələr

Hesabi ifadələrdə hamıya məlum olan ənənəvi hesab əməlləri ilə yanaşı, bir sıra xüsusi əməliyyatlar da tətbiq olunur. Bu əməliyyatların bəzilərini nəzərdən keçirək.

Div əməliyyatı iki tam tipli sabit və dəyişənləri tamədədli bölmə əməliyyatı adlanır.

Misal.

```
39 div 4 = 9; a=53 və b=7 olduqda a div b = 7.
```

Mod əməliyyatı iki tam tipli sabit və dəyişənlərin tamədədli bölünmə qalığının tapılması əməliyyatı adlanır.

Misal.

```
39 mod 4 = 3; a=53 və b=7 olduqda a mod b = 4.
```

Hər iki əməliyyat yalnız müsbət tam ədədlərə tətbiq olunur.

Tam tipli verilənlər üzərində daha mürəkkəb çevirmələr aparan əməliyyatlar da tətbiq olunur. Bunlar bitlər üzrə yerinə yetirilən **shl** – *sola sürüşdürmə* və **shr** – *sağa sürüşdürmə* əməliyyatlarıdır. Bu əməliyyatlar nəticəsində verilənlərin bitləri (mərtəbələri) sağa (`shr`) və ya sola (`shl`) sürüşdürülür; bu zaman artıq bitlər atılır, azad olan bitlər isə sıfırlarla doldurulur. Məsələn,

Misal.

```
00000111 shl 3 = 00111000;
00111000 shr 3 = 00000111.
```

Bu əməliyyatları işarəli tam ədədlərə tətbiq etdikdə yadda saxlamaq lazımdır ki, ədədin böyük mərtəbəsi onun işarəsini müəyyən edir. Ona görə də istənilən sağa sürüşdürmədə həmin bit sıfra çevriləcək, yəni alınan ədəd müsbət olacaqdır. Sola sürüşdürmədə isə nəticənin işarəsi ixtiyari ola bilər. Bu əməliyyatlar müsbət ədədlərə tətbiq olunduqda, `shl` əməliyyatının nəticəsi – verilmiş ədədi əsası *2* olan qüvvətüstlü kəmiyyətə vurmaqla, `shr` əməliyyatının nəticəsi isə bölməklə alınan nəticələrlə eyni olur. Başqa sözlə, yuxarıda yazılmış misallarla aşağıdakı əməliyyatlar eyni nəticəli olur:

`7 shl 3`, yəni $7*2^3 = 56$,
`56 shr 3`, yəni $56 \text{ div } 2^3 = 7$.

Qeyd edək ki, `shl` və `shr` əməliyyatları ilə hesablamalar daha tez yerinə yetirilir və daha optimal maşın kodları yaradılır.

Tam tipli verilənlər üzərində daha dörd *bit əməliyyatları* – **`not`, `and`, `or`** və **`xor`** yerinə yetirilə bilər. Xatırladırıq ki, bu əməliyyatlar verilənlərin bitləri üzərində yerinə yetirilir. Həmin əməliyyatlar cədvəl 4.1 – də göstərilmişdir.

Cədvəl 4.1. Bit əməliyyatları

Dəyişənlər		**Əməliyyatların nəticələri**				
a	**b**	**a and b**	**a or b**	**a xor b**	**not a**	**not b**
0	*0*	*0*	*0*	*0*	*1*	*1*
0	*1*	*0*	*1*	*1*	*1*	*0*
1	*0*	*0*	*1*	*1*	*0*	*1*
1	*1*	*1*	*1*	*0*	*0*	*0*

Misal.

```
Not 01111011=10000100;
10111010 and 01100011=100010;
10111010 or 01100011=11111011;
10111010 xor 01100011=11011001;
48 and 36=32;
48 or 36=52;
48 xor 36=20.
```

Pascal dilində ifadələrdə hazır element kimi istifadə olunan əvvəlcədən hazırlanmış alt proqram – funksiyalar mövcuddur. Turbo Pascal dilində isə bunların sayı artırılmış və standart bir modulda yerləşdirilmişdir. Turbo Pascal proqramlarında sabit, dəyişən, prosedur və funksiyalardan istifadə edərkən, onların təyin olunduğu modullar təsvir olunmalıdır. İstifadəçi tərəfindən yaradılan modulların və **`System`** modulunun təsviri vacib deyil. Digər modullar hökmən təsvir edilməlidir. Funksiyalara müraciət edərkən onun adı, sonra isə adi mötərizədə funksiyanın arqumentləri göstərilməlidir. Arqumentlər birdən çox olduqda onlar bir–birindən vergüllə ayrılır.

Hesabi funksiyalar. Cədvəl 4.2 – də **System** modulunun tərkibinə daxil olan, sadə riyazi hesablamaları yerinə yetirən və ən çox istifadə edilən funksiyalar göstərilmişdir.

Cədvəl 4.2. Ən çox istifadə edilən funksiyalar

Riyazi funksiya	Proqramda yazılışı	Nəticənin tipi
$y=\|x\|$	`y:=abs(x)`	*x ilə eynitipli*
$y=x^2$	`y:=sqr(X)`	"_ _ _"
$y=\sqrt{x}$	`y:=sqrt(X)`	*həqiqi*
y=arctgx	`y:=arctan(x)`	"_ _ _"
y=cosx	`y:=cos(x)`	"_ _ _"
y=sinx	`y:=sin(x)`	"_ _ _"
$y=e^x$	`y:=exp(x)`	"_ _ _"
y=lnx	`y:=ln(x)`	"_ _ _"
π ədədi	`Pi`	"_ _ _"

Hesabi ifadələri düzgün yazmaq üçün aşağıdakı qaydalara riayət etmək lazımdır:

– simvollar sətirdə bütün əməliyyat işarələri qoyulmaqla eyni səviyyədə yazılır;

– iki ardıcıl əməliyyat işarələrinə icazə verilmir. Məsələn, `A+-B` yazılışı səhvdir. Düzgün yazılış `A+(-B)` kimi olmalıdır.

Cədvəl 4.2 – dən göründüyü kimi, Turbo Pascal dilində qüvvətə yüksəltmə üçün əməliyyat və ya funksiya yoxdur. $y=x^n$ funksiyasının hesablanmasında aşağıdakılar məsləhət görülür: əgər *n*–tam ədəddirsə, onda qüvvətin hesablanmasında vurma əməliyyatından və ya `sqr` standart funksiyasından istifadə oluna bilər. Məsələn, $y=x^4$ funksiyası `y:=x*x*x*x` və ya `y:=sqr(x)*sqr(x)` kimi yazılır; böyük qüvvətlərdə vurma dövrlərdə hesablanır; əgər *n* – həqiqi ədəddirsə, onda $y = x^n = e^{n\ln(x)}$ riyazi düsturundan istifadə olunur ki, bu ifadə də Turbo Pascal dilində `y:= exp(n*ln(x))` kimi yazılır.

Turbo Pascal dilində bir neçə ifadənin yazılışına baxaq.

Misal.

Riyazi ifadə	*Turbo Pascal dilində yazılışı*
1. $y=ax+b$	`y:=a*x+b;`
2. $y=\frac{a+b}{c+d}\sin(x)$	`y:=(a+b)/(c+d)*sin(x);`
3. $y=\frac{a\ln x+b\sqrt{x}}{\cos x-x^2}$	`y:=(a*ln(x)+b*sqrt(x))/` `(cos(x)-sqr(x));`

4. $y = x^{\frac{1}{7}} + arctg\,\frac{x^2}{\sqrt{a}}$

```
y:=exp((1./7.)*ln(x))+
   arctan(sqr(x)/sqrt(a));
```

5. $y = \pi \cdot e^{ax+b} + \ln|x-a|$

```
y:=Pi*exp(a*x+b)+
   ln(abs(x-a));
```

6. $y = x^6$

```
y:=exp(6*ln(x));
```

7.
$$y = \frac{\cos^3(x-a) + \sin^2(x-a)^3}{\sqrt{|x-c^2 z|} + \pi e^{\frac{a}{b}}}$$

```
y:=(exp(3.0*ln(cos(x-a)))+
   sqr(sin(exp(3.0*ln(x-
   a)))))/(sqrt(abs(x-
   c*c*z))+pi*exp(a/b));
```

8. $y = \cos\varphi + \sin\gamma$

```
y:=cos(f)+sin(g);  və ya
y:=cos(fi)+sin(qamma));
```

4.3. Məntiqi ifadələr

Məntiqi ifadələrdə **and**, **or**, **xor** və **not** məntiqi əməliyyatlarından istifadə olunur ki, onlar üzərində əməliyyatların yerinə yetirilmə qaydası cədvəl 4.3 – də göstərilmişdir.

Cədvəl 4.3. Məntiqi əməliyyatlar

Dəyişənlər		**Əməliyyatların nəticələri**				
a	**b**	**a and b**	**a or b**	**a xor b**	**not a**	**not b**
False	*False*	*False*	*False*	*False*	*True*	*True*
False	*True*	*False*	*True*	*True*	*True*	*False*
True	*False*	*False*	*True*	*True*	*False*	*True*
True	*True*	*True*	*True*	*False*	*False*	*False*

Turbo Pascal dilində bir neçə məntiqi ifadənin yazılışına baxaq.

Misal.

Riyazi ifadə	***Turbo Pascal dilində yazılışı***
1. $y \ge \frac{a+b}{c+d}\sin(x)$	`y>=(a+b)/(c+d)*sin(x);`
2. $y \le a\ln x$	`y<=a*ln(x);`
3. $x \le a$ ***və ya*** $y \ge b$	`(x<=a) or (y>=b);`
4. $a < x \le b$	`(x>a) and (x<=b);`
5. $y=a$ ***və*** $x=c$ ***və*** $z=d$	`(y=a) and (x=c) and (z=d);`

4.4. Münasibət əməliyyatları

Münasibət əməliyyatları tam, həqiqi, məntiqi, simvol, sətir tipli verilənlər və massivin elementləri üzərində yerinə yetirilə bilər. Münasibət əməliyyatlarının nəticəsi məntiqi sabitlərdir (*True, False*). Simvol və sətir tipli sabitlərin müqayisəsi **ASCII** simvollar koduna uyğun olaraq, soldan sağa istiqamətdə, simvollar üzrə həyata keçirilir. Hansı simvolun kodu böyükdürsə, həmin sətir böyük olur.

Turbo Pascal dilində bir neçə münasibət əməliyyatlarının yazılışına baxaq.

Misal.

İfadə	Nəticə
`10>5`	`True`
`6=5`	`False`
`False<>True`	`True`
`'ADNA'< 'BDU'`	`True`
`'ALFA' > 'A'`	`True`
`'AAHM'='AAHM'`	`True`
`'AAHM  '>'AAHM'`	`True`
`'A'>'a'`	`False`

4.5. Sətir ifadələri

Sətir ifadələri üzərində əməliyyatların nəticələri simvollardan ibarət sətirlər olur. Sətirlər üzərində yalnız toplama əməli yerinə yetirilə bilər ki, bu əməliyyat nəticəsində sətirlərin birləşməsi (*konkatenasiya*) baş verir.

Misal.

```
S:='BAKI '+'AZƏRBAYCANIN '+'PAYTAXTIDIR';
```

Bu operatorun icrasından sonra, `S` sətir tipli dəyişənin qiyməti `'BAKI AZƏRBAYCANIN PAYTAXTIDIR'` olacaqdır.

Konkatenasiya əməliyyatı nəticəsində alınan sətir *255* simvoldan çox olmamalıdır. Əgər sətrin uzunluğu *255* – dən çox olarsa, onda *255* – ci simvoldan sonrakı simvollar atılır.

Bundan başqa, yuxarıda göstərdiyimiz kimi, sətirlər müqayisə də oluna bilər. Əgər sətirdə simvolların sayı bərabər, simvollar isə ekvivalent olarsa, onda sətirlər bərabər olur. Sətirlərin müqayisəsi hər simvol üzrə yerinə yetirilir və hansı simvolun kodu böyükdürsə, həmin sətir böyük olur.

Misal.

```
'BAKI'='BAKI';
'A' < 'a';          { A – nın kodu 65, a – nın kodu isə 97 – dir }
'AZƏRBAYCAN' > 'BAKI';
```

`String` tipli sətirlərlə işləmək üçün Turbo Pascal dilində aşağıdakı funksiya və prosedurlardan istifadə olunur:

`Concat` **(***s1, s2, ..., sN***); *funksiyası*** – *s1, s2, ..., sN* sətirlər ardıcıllığının *konkatenasiyasını* (birləşməsini) yerinə yetirir.

`Copy` **(***st, index, count***); *funksiyası*** – verilmiş *st* sətrindən, *index* nömrəli simvoldan başlayaraq *count* sayda simvolların *surətini qaytarır*.

Misal.

```
SS:='Microsoft Word -2003';
NSS:= Copy(SS,11,4);
```

olarsa, nəticədə `NSS` sətir dəyişəninin qiyməti `NSS:='Word'` olacaqdır.

`Delete` **(***st, index, count***); *proseduru*** – verilmiş *st* sətrindən, *index* nömrəli simvoldan başlayaraq *count* sayda *simvolları pozur*.

Misal.

```
SS:='Microsoft Word -2003';
Delete(SS,11,4);
```

olarsa, nəticədə `SS` sətir dəyişəninin qiyməti `SS:='Microsoft -2003'` olacaqdır.

`Insert` **(***subst, st, index***); *proseduru*** – *subst* alt sətrini, *index* nömrəli simvoldan başlayaraq, *st* sətrində *yerləşdirir*.

Misal.

```
SS:='Microsoft -2003';
Insert('Word',SS,11);
```

olarsa, nəticədə `SS` sətir dəyişəninin qiyməti `SS:='Microsoft Word -2003'` olacaqdır.

`Pos`(*subst, st*)**; *funksiyası*** – *st* sətrinə daxil olan *subst* alt sətrinin *başlanğıc mövqeyini* müəyyən edir. Əgər *st* sətrində *subst* alt sətri mövcud olmazsa, onda `Pos` funksiyası sıfra bərabər olur.

Misal.

```
SS:='Microsoft Word -2003';
Pos('Word',SS);
```

olarsa, `Pos` funksiyasının nəticəsi `11` olacaqdır.

`Length`(*st*)**; *funksiyası*** – *st* sətrinin *uzunluğunu qaytarır*. Məsələn, `Length('Turbo Pascal')` funksiyasının qiyməti `12` olacaqdır.

`Str` **(** *x* [**:***m* [**:***n*], *st* **); *proseduru*** – istənilən *x* həqiqi və ya tam tipli ədədi, *st sətir simvollarına (tipinə) çevirir*. *m* və *n* parametrləri çevirmə formatını göstərir. Bu parametrlər yazılmaya da bilər. *m* parametri *x* həqiqi və ya tam tipli ədədinin simvol təsvirindəki ümumi sahənin uzunluğunu, *n* isə kəsr hissədəki

simvolların sayını təyin edir. *x* yalnız həqiqi ədəd olduqda *m* və *n* parametrlərindən istifadə olunur.

Misal.

```
Const X:=2000;
Var s:string;
...
Str(x,s,err);
```

Nəticədə `x` sabitinin qiyməti olan `2000` ədədi, simvollara çevrilərək `s` dəyişəninə mənimsədiləcəkdir, yəni `s='2000'` olacaqdır.

`Val` (*st, x, code***); *proseduru*** – *st* simvollar sətrini, *həqiqi və ya tamədədli x təsvirinə (tipinə) çevirir*. Əgər *code* – nin qiyməti sıfır olarsa, çevirmə müvəffəqiyyətlə başa çatır, əks halda, *st* sətrinin səhv olan simvolunun mövqeyinin nömrəsini qaytarır.

Misal.

```
Const s:='2000';
Var x:integer;
...
Val(s,x,err);
```

Nəticədə, `s` sabitinin simvollardan ibarət `'2000'` sətir qiyməti, `2000` ədədinə çevrilərək `x` dəyişəninə mənimsədiləcəkdir, yəni `x=2000` olacaqdır. Hər iki misalda çevirmə düzgün yerinə yetirilərsə, `err` parametrinin qiyməti sıfra bərabər olacaqdır.

`UPCASE` (*ch***); *funksiyası*** – *kiçik latın hərflərini uyğun böyük hərflərə çevirir*, nəticə `char` tipli olur. `UPCASE('t')` funksiyasının qiyməti `'T'` olacaqdır.

4.6. Standart funksiya və prosedurlar

Turbo Pascal dilində standart funksiya və prosedurlar müxtəlif əməliyyatları çox asanlıqla yerinə yetirmək üçün nəzərdə tutulmuşdur. Məsələn, simvolun tam ədədə, həqiqi ədədin tam ədədə çevrilməsi, həqiqi ədədlərin tam və kəsr hissələrinin tapılması və s. Bu funksiyalardan bəziləri aşağıda göstərilmişdir.

`Chr`(*x***);** – **ASCII** *kodunun simvola çevrilməsi.* Funksiyanın arqumenti *0..255* intervalında olmaqla, tam ədəd olmalıdır. Nəticədə bu koda uyğun simvol alınır. Məsələn, `chr(97)` ifadəsinin nəticəsi `'a'` olacaqdır.

Ord(*x*); – *istənilən sıra tipinin tam tipə çevrilməsi*. Funksiyanın arqumenti ixtiyari sıralı (məntiqi, simvol və sadalanan) tip ola bilər. Nəticədə `longint` tipli kəmiyyət alınır. Məsələn, `Ord('a')` ifadəsinin nəticəsi `97` olacaqdır.

Round(*x*); – *x həqiqi ədədinin qiymətinin bu ədədə yaxın olan tam ədədə qədər yuvarlaqlaşdırılması.* Funksiyanın arqumenti həqiqi, nəticə isə `longint` tipində olur. Məsələn, `Round(7.96)=8; Round(7.06)=7` olur.

Trunc(*x*); – *x həqiqi ədədinin tam hissəsinin tapılması*. Funksiyanın arqumenti həqiqi, nəticə isə `longint` tipində olur. Məsələn, `Trunc(7.96)=7; Trunc(7.06)=7` olur.

Int(*x*); – *tipini dəyişdirmədən x həqiqi ədədinin tam hissəsinin tapılması.* Məsələn, `İnt(75.96)=75.0; İnt(13457.88976)=13457.0` olur.

Frac(*x*); – *x həqiqi ədədinin kəsr hissəsinin tapılması.* Məsələn, `Frac(7.96)=0.96; Frac(7.06)=0.06` olur.

Sıralı tip kəmiyyətlər üçün funksiyalar. Bu funksiyalar əvvəlki və ya sonrakı elementlərin tapılması, ədədin təkliyinin yoxlanılması və s. üçündür. Buraya aşağıdakı funksiyalar aiddir:

Odd(*x*); – *x dəyişəninin təkliyinin yoxlanılması.* Funksiyanın arqumenti `longint` tipli, nəticə isə arqument tək olduqda – *True*, cüt olduqda isə *False* olur. Məsələn, `Odd(32)` funksiyasının nəticəsi *False,* `Odd(33)` funksiyasının nəticəsi isə *True* olacaqdır.

Pred(*x*); – *x dəyişəninin əvvəlki qiymətinin təyini.* Funksiyanın arqumenti sıra tipli ixtiyari kəmiyyət, nəticə isə həmin tipli əvvəlki qiymətdir. Məsələn, `Pred(20)` funksiyasının nəticəsi `19`, `Pred('B')` funksiyasının nəticəsi isə `'A'` olacaqdır.

Succ(*x*); – *x dəyişəninin sonrakı qiymətinin təyini.* Funksiyanın arqumenti sıra tipli ixtiyari kəmiyyət, nəticə isə həmin tipli sonrakı qiymətdir. Məsələn, `Succ(2)` funksiyasının nəticəsi `3`, `Succ('B')` funksiyasının nəticəsi isə `'C'` olacaqdır.

Inc(*x*); – *x dəyişəninin qiymətini 1 vahid artırır.* Məsələn, əgər `a:=4` olarsa, `Inc(a)` icra olunduqdan sonra `a=5` olacaqdır.

Dec(*x*); – *x dəyişəninin qiymətini 1 vahid azaldır.* Məsələn, əgər `a:=4` olarsa, `dec(a)` icra olunduqdan sonra `a=3` olacaqdır.

Random; – *$0 \le s < 1$ aralığında təsadüfi s ədədi yaradır.*

Random(*x*); – *0 ≤ s< x aralığında təsadüfi s ədədi yaradır.*

Ekranla işləmək üçün prosedurlar.

ClrScr; – *ekranı təmizləyir, onu fonun rəngi ilə rəngləyir və kursoru ekranın sol yuxarı küncünə yerləşdirir.*

GotoXY (*x, y* **: Byte);** – *kursoru ekranın x , y dəyişənləri ilə verilmiş mövqeyinə yerləşdirir. x* – sütunu, *y* isə sətri göstərir. Məsələn, `GotoXY(30,25)` o deməkdir ki, kursor *25*–ci sətrin *30*–cu mövqeyində yerləşəcəkdir.

WhereX; – *üfqi koordinat oxu üzrə kursorun cari x koordinatını müəyyən edir.*

WhereY; – *şaquli koordinat oxu üzrə kursorun cari y koordinatını müəyyən edir.*

4.7. Çoxluqlar üzərində əməliyyatlar

Çoxluqlar üzərində əməliyyatlar çoxluqlar nəzəriyyəsinin qaydalarına görə aparılır.

İki çoxluğun birləşməsi, yəni `A+B` əməliyyatının nəticəsi, həm `A` çoxluğunun, həm də `B` çoxluğunun bütün təkrarlanmayan elementlərindən ibarət `C` çoxluğudur.

İki çoxluğun fərqi, yəni `A-B` əməliyyatının nəticəsi, `A` çoxluğunun `B` çoxluğuna daxil olmayan elementlərindən ibarət `C` çoxluğudur.

İki çoxluğun kəsişməsi, yəni `A*B` əməliyyatının nəticəsi, `A` və `B` çoxluqlarının eyni elementlərindən təşkil olunmuş `C` çoxluğudur.

`A` və `B` çoxluqlarının elementləri eyni olduqda `A=B` əməliyyatının nəticəsi *True* və `A<>B` əməliyyatının nəticəsi *False* olur.

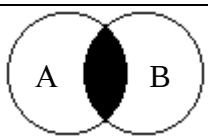
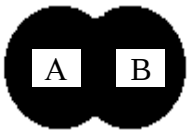
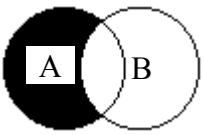
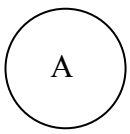
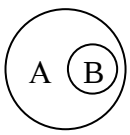
Əgər `A` çoxluğu `B` çoxluğunun alt çoxluğudursa, onda `A<=B` əməliyyatının nəticəsi *True* olur.

Əgər `A` çoxluğu `B` çoxluğunun bütün elementlərini özündə saxlayırsa, onda `A>=B` əməliyyatının nəticəsi *True* olur.

Əgər hər hansı *x* kəmiyyəti `A` çoxluğunun elementidirsə, onda `x in A` əməliyyatının nəticəsi *True* olur.

Cədvəl 4.4 – də çoxluqlar üzərində əməliyyatların həndəsi təsviri göstərilmişdir.

Cədvəl 4.4. Çoxluqlar üzərində əməliyyatların həndəsi təsviri

Riyazi işarəsi	**Proqramda işarəsi**	**Əməliyyatlar**	**Həndəsi təsviri**
$\cap$	`*`	*Kəsişmə*	 C=A∩B `C:=A*B` *Nəticənin tipi – çoxluq*
$\cup$	`+`	*Birləşmə*	 C=A∪B `C:=A+B` *Nəticənin tipi – çoxluq*
\	`-`	*Fərq*	 C=A\B `C:=A-B` *Nəticənin tipi – çoxluq*
$\in$	`in`	*Aiddir və ya çoxluğun elementidir*	 *X*∈*A* `if X in A then` *Nəticənin tipi* – `boolean`
$\subset (\subseteq)$ $\supset (\supseteq)$	`<=` `>=`	*Alt çoxluqdur* *Alt çoxluğu var*	*Çoxluğun operandları*  *A*⊃*B [B*⊃*A]* `A>=B [B<=A]` *Nəticənin tipi* – `boolean`

Misal.

İfadə	**Nəticə**
`[1,2,3,4]+[3,4,5,6]`	`[1,2,3,4,5,6]`
`[1,2,3,4]-[3,4,5,6]`	`[1,2]`
`[1,2,3,4]*[3,4,5,6]`	`[3,4]`
`[1,2,3]=[1,2,3,4]`	`False`
`[1,2,3]<>[1,2,3,4]`	`True`
`[1,2,3]<=[1,2,3,4]`	`True`
`[1,2,3]>=[1,2,3,4]`	`False`
`4 in [3,4,5,6]`	`True`

Misal.

```
Type MonthDays = Set of 1..31;

Var
    color : set of (Red, Blue, White, Black);
    Day : MonthDays;

Begin

  ...

  Color:=[Blue]; { Color dəyişəninə Blue qiyməti mənimsədilir }
  Color:=Color-[blue,red,white]; { Color=[ ] olur }
  Color:=Color+[Black];          { Color=Black olur }
  Day:=[ ];                      { Day dəyişəni boş çoxluqdur }
  Day:=Day-[1];                  { Day=[ ]-[1]=[ ] }
  Day:=[2,4];
  Day:=Day+[5,12,1];   {Day=[2,4]+[5,12,1]=[2,4,5,12,1] }
  Day:=Day-[1];        {Day=[2,4,5,12,1]-[1]=[2,4,5,12] }

end;
```

Beşinci fəsil

OPERATORLAR

Bu fəsildə proqramlaşdırmanın ən vacib konstruksiyaları olan operatorlar bölməsi öyrəniləcəkdir. İlkin verilənləri proqrama daxil etmək və alınmış nəticələri ekranda təsvir etmək üçün verilənləri daxiletmə və xaricetmə prosedurlarını, mənimsətmə və keçid operatorlarını öyrənəcəyik. Strukturlaşdırılmış operatorlar qrupuna daxil olan tərkibli operator, şərti operator, seçim operatoru, üç növ dövr operatorları və `With` operatoru izah olunacaqdır. Bu operatorların mümkün yazılış konstruksiyalarına baxılacaq və onların tətbiqi ilə müxtəlif məsələlər həll ediləcəkdir.

5.1. Verilənləri daxiletmə və xaricetmə prosedurları

Turbo Pascal dilində standart daxiletmə – `Read`, `Readln` və standart xaricetmə isə `Write`, `Writeln` prosedurları vasitəsilə həyata keçirilir. Bu prosedurların ümumi forması belədir:

`Read` (*fayl dəyişəninin adı, daxil ediləcək dəyişənlərin siyahısı*) **;**
`Readln` (*fayl dəyişəninin adı, daxil ediləcək dəyişənlərin siyahısı*) **;**
`Write` (*fayl dəyişəninin adı, xaric ediləcək dəyişənlərin siyahısı*) **;**
`Writeln` (*fayl dəyişəninin adı, xaric ediləcək dəyişənlərin siyahısı*) **;**

`Read` və `readln`, `write` və `writeln` prosedurlarının yazılışlarındakı fərq ondan ibarətdir ki, `read`, `write` prosedurları ilə informasiya bir sətirdə, `readln`, `writeln` prosedurları ilə isə informasiya yeni sətirdə *("read line", "write line")* təsvir olunur.

Fayl dəyişəni ilkin informasiyanın daxil ediləcəyi və ya nəticələrin saxlanacağı faylla əlaqə yaradan *məntiqi fayl dəyişənidir* və onun haqqında "*Fayllar*" bölməsində daha geniş bəhs edəcəyik. Daxiletmə prosedurları fayldan yalnız bir simvol oxuyaraq onu dəyişənə mənimsədir. Ədədlərin oxunması isə birinci probelə, tabulyasiya simvoluna, sətrin sonu işarəsinə və ya faylın sonu işarəsinə kimi həyata keçirilir. Bu dəyişən yazılmadıqda standart daxiletmə (*klaviaturadan*) və standart xaricetmə (*monitora*) yerinə yetirilir. `Writeln` proseduru mətn faylları üçün `write` prosedurunun genişlənmiş variantıdır. Parametrsiz `writeln` proseduru faylın sonuna yalnız sətrin sonu işarəsini yazır.

Standart daxiletmə əvvəlcədən təyin olunmuş, klaviatura ilə əlaqəli `Input` adlı mətn faylından yerinə yetirilir. Standart xaricetmə isə əvvəlcədən təyin olunmuş, monitorla əlaqəli `Output` adlı mətn faylından yerinə yetirilir. Susmaya görə daxiletmə üçün `Input`, xaricetmə üçün isə `Output` götürülür:

```
readln(Input, A, B);
writeln(Output, 'A=', A, 'B=', B);
```

Proqramda `Input` və `Output` sözlərini yazmamaq daha məqsədəuyğundur və bu zaman giriş–çıxış qurğuları kimi klaviatura və monitor başa düşülür, yəni informasiya klaviaturadan daxil edilir və monitora çıxarılır:

```
readln(A,B);
writeln('A=',A, 'B=',B);
```

Standart daxiletmə və xaricetmə prosedurlarından istifadə edərkən aşağıdakılar nəzərə alınmalıdır:

– `read` və `readln` prosedurları ilə yalnız tam, həqiqi, simvol və sətir tipli verilənlər daxil edilir;
– `write` və `writeln` prosedurları ilə yalnız tam, həqiqi, simvol, sətir və məntiqi tipli verilənlər monitora çıxarılır.

5.1.1. Verilənlərin ekrandan daxil edilməsi

Verilənləri ekrandan daxil etdikdə `read` və `readln` prosedurlarının ümumi forması belə olur:

`Read` (*daxil ediləcək dəyişənlərin siyahısı*)**;**
`Readln` (*daxil ediləcək dəyişənlərin siyahısı*)**;**

Daxiletmədə `read` prosedurundan fərqli olaraq `readln` proseduru verilənlərin növbəti sətrin başlanğıcından oxunmasını həyata keçirir. Əgər proqramda sadəcə olaraq

```
Readln;
```

yazılarsa, onda proqramın yerinə yetirilməsi dayandırılır və proqramın işini davam etdirmək üçün *Enter* klavişini basmaq lazımdır. Daxiletmə prosedurunun bu formasından, adətən, Turbo Pascal dilinin inteqrallaşdırılmış mühitində

istifadəçi pəncərəsinin ekranda görünməsini təmin etmək üçün istifadə olunur. Belə ki, bu məqsədlə sonuncu `end.` operatorundan əvvəl `readln` yazmaq lazımdır. İnteqrallaşdırılmış mühitin pəncərəsini yenidən ekranda göstərmək üçün *Enter* klavişini basmaq lazımdır.

Dəyişənlərin qiymətlərini klaviaturadan daxil etdikdə, onları aralarında probel işarəsi qoymaqla bir sətirdən və ya dəyişənlərin hər qiymətini bir sətirdən daxil etmək lazımdır (*Enter* klavişini basmaqla).

Misal. a = 7,8; b = 10,965; s = "BAKI" və k = – 654 qiymətlərinin klaviaturadan daxil edilməsi.

```
program daxil_etme;
uses crt;
var a,b:real;
   k:integer;
   s:string[4];
begin
  read(s,a,b,k);
  writeln(a,b,k,s);
  Readln
end.
```

Bu proqrama əsasən, dəyişənlərin qiymətlərini klaviaturadan

```
BAKI  7.8  10.965  -654
```

kimi, və ya hər qiyməti bir sətirdə yazmaqla, daxil etmək lazımdır. Sətir tipli dəyişənləri daxil etdikdə `read` proseduru, elanedici hissədə sətrin uzunluğu nə qədər göstərilmişdirsə, bir o qədər simvol oxuyur. Bu zaman probel ayırıcı rolunu oynamır. `read` proseduru yeni sətrə keçidi yerinə yetirmir, bu əməliyyatı `readln` proseduru yerinə yetirir. Sonuncu `readln` proseduru inteqrallaşdırılmış mühitin istifadəçi pəncərəsini ekranda ləngitmək üçün yazılmışdır. *Enter* klavişini basdıqdan sonra, yenidən inteqrallaşdırılmış mühitə qayıdacaqsınız.

Misal.

```
program daxil_et;
uses crt;
var A:array [0..5] of Char;
    S1,S2,S3:string [10];
Begin
  Read(A);
  Read(S1);
  Read(S2);
  Readln;
  Read(S3);
  Readln
end.
```

Bu halda, bir sətirdə `A` massivinin elementləri üçün *5* simvol, onun ardınca `S1` və `S2` sətir dəyişənlərinin hər biri üçün *10* simvol yazaraq *Enter* klavişini basdıqdan sonra, növbəti sətirdə `S3` sətir dəyişəni üçün *10* simvol daxil etmək lazımdır. Qeyd edək ki, massivlərin elementlərinin belə daxil edilməsi məqsədəuyğun deyildir və sonrakı bölmələrdə massivlərin elementlərinin daxil edilməsi üçün daha səmərəli üsullar göstərəcəyik.

5.1.2. Verilənlərin ekrana çıxarılması

Verilənləri ekrana çıxardıqda `write` və `writeln` prosedurlarının ümumi forması belə olur:

`Write` (*xaric ediləcək dəyişənlərin siyahısı* **:** m **:** n **);**
`Writeln` (*xaric ediləcək dəyişənlərin siyahısı* **:** m **:** n **);**

Yuxarıda qeyd etdiyimiz kimi, əgər fayl dəyişəni yazılmazsa, onda nəticələr ekrana çıxarılacaqdır. Nəticələri printerdə çap etmək üçün fayl dəyişəninin adı əvəzinə `'prn'` (“*printer*”) yazmaq lazımdır. Məsələn,

```
Writeln('prn',a,b,c);
```

Əgər ekrana çıxarılma prosedurları

`Write` (*xaric ediləcək dəyişənlərin siyahısı* **);**
`Writeln` (*xaric ediləcək dəyişənlərin siyahısı* **);**

kimi yazılarsa, onda həqiqi ədədlər həmişə sürüşkən vergüllü formatda təsvir olunacaqdır.

`Write` və `writeln` prosedurlarının ümumi formalarındakı *m* – xaricedilmə sahəsinin ümumi uzunluğunu, *n* isə xaric edilən dəyişənin onluq hissəsindəki rəqəmlərin sayını göstərən parametrlərdir (*m*>*n*). *n* parametri, çap olunacaq dəyişən yalnız həqiqi tipli olduqda göstərilir. *n* göstərildikdə ədəd qeyd olunmuş vergüllü formatda, göstərilmədikdə isə sürüşkən vergüllü formatda təsvir olunur. Tam və sətir tipli verilənlər üçün *n* yazılmır.

Əgər format `a:m:0` kimi yazılarsa, onda ədədin kəsr hissəsi tam ədədə qədər yuvarlaqlaşdırılacaq və onluq nöqtə ekrana çıxarılmayacaqdır. Bu halda, məsələn, *a=1248.6* olarsa və format `a:6:0` kimi yazılarsa, onda bu ədəd, ekranda qarşısında iki probel işarəsi qoyulmaqla, yuvarlaqlaşdırılaraq *1249* kimi təsvir ediləcəkdir. Əgər *a=150* olduqda, format `a:5:0` kimi yazılarsa, onda ədədin qarşısında iki probel işarəsi qoyulacaqdır. *c=1248.45* ədədi üçün `c:10:2` formatı yazılarsa, onda kompüter bu ədədi, qarşısına üç probel əlavə etməklə, olduğu kimi təsvir edəcəkdir; əgər səhv olaraq `c:2:8` formatı yazılarsa, onda ədəd *1248.45000000* kimi təsvir ediləcəkdir.

Əgər proqramda sadəcə olaraq

```
Writeln;
```

yazılarsa, onda bir boş sətir buraxılır.

Misal.

```
program ekrana_cixarma;
uses Crt;
const
  i: Integer = 12345;
  r: Real = -123.1234567;
  c: Char = '$';
  b: Boolean = True;
  s: string = 'Müasir proqramlaşdırma dilləri';
begin
  ClrScr;
  writeln('formatsız çap');
  writeln(i, r, c, b, s);
  writeln;
  writeln('formatlı çap');
  Writeln(i:10, r:10:3, c:10, b:6, s:35);
  Writeln;
  writeln('Həqiqi ədədlərin qeyd
          olunmuş formatda təsviri');
  writeln(r:3:0);
  writeln(r:6:3);
  writeln(r:13:7);
  writeln(r:25:9);
  writeln;
  writeln('Həqiqi ədədlərin sürüşkən
          formatda təsviri');
  writeln(r:3);
  writeln(r:6);
  writeln(r:13);
  writeln(r:25);
  Readln
end.
```

Bu proqramın nəticəsi şəkil 5.1 – də göstərilmişdir.

```
formatsiz cap
12345-1.2312345670E+02$TRUEMuasir programlashdirma dilleri

formatli cap
     12345  -123.123         $  TRUE    Muasir programlashdirma dilleri

Heqiqi ededlerin qeyd olunmush formatda tesviri
-123
-123.123
 -123.1234567
          -123.123456700

Heqiqi ededlerin surushken formatda tesviri
-1.2E+02
-1.2E+02
-1.231235E+02
        -1.2312345670E+02
```

Şəkil 5.1. Müxtəlif tip verilənlərin ekranda təsvir qaydaları

5.2. Mənimsətmə operatoru

Mənimsətmə operatoru dilin əsas operatorudur. Bu operatorun ümumi forması belədir:

a:= b;

Burada, `b` – sabit, dəyişən, ifadə və massivin elementi, `a` isə dəyişən və ya massivin elementidir. **:=** işarəsi bərabərlik işarəsindən fərqlidir. Belə ki, bu operator icra olunduqda `b` ifadəsi hesablanır və onun nəticəsi `a` dəyişəninə mənimsədilir.

Misal.

```
x:=x+1;
a:=3.8;
b:=8*Pi/sin(x);
s:='TARİX';
y:=(a+b)/(c+d);
```

Misal. Verilmiş *4* rəqəmli tam ədədin rəqəmlərinin tərsinə düzülüşündən alınan ədədin tapılması.

```
program ters_duz;
uses crt;
const n=4658;
var m1qis,m1qal,m2qis:integer;
    m2qal,m3qis,m3qal:integer;
    m:integer;
begin
   m1qis:=n div 1000;
   m1qal:=n mod 1000;
   m2qis:=m1qal div 100;
   m2qal:=m1qal mod 100;
   m3qis:=m2qal div 10;
   m3qal:=m2qal mod 10;
   m:=m3qal*1000+m3qis*100+m2qis*10+m1qis;
   writeln('4 rəqəmli tam ədəd = ',n:4);
   writeln('tərsinə düzülmüş tam ədəd = ',m:4);
   readln
end.
```

Bu məsələnin alqoritmi belədir: verilmiş ədədi *1000* ədədinə tamədədli böldükdə (`div` əməliyyatı – `m1qis` dəyişəni) alınmış qismət axtarılan ədədin ən kiçik mərtəbəsi (*4*–cü rəqəmi – `m1qis` dəyişəni) olur. Bu bölmə nəticəsində alınan qalığı (`mod` əməliyyatı – `m1qal` dəyişəni) *100* ədədinə bölüb, alınmış qisməti *10*–a vursaq, axtarılan ədədin *2*–ci rəqəmini (`m2qis*10`) alarıq. Bu bölmə nəticəsində alınan qalığı *10* ədədinə bölüb, alınmış qisməti *100*–ə vursaq, axtarılan ədədin *3*–cü rəqəmini (`m3qis*100`) alarıq. Axtarılan ədədin birinci rəqəmi isə sonuncu bölmə

əməliyyatından alınmış qalığı *1000*–ə vurmaqla alınır `(m3qal*1000)`. Nəhayət, sonda bu ədədləri toplamaq lazımdır:

```
m:=m3qal*1000+m3qis*100+m2qis*10+m1qis;
```

Proqramın icrasından sonra aşağıdakı nəticələr alınacaqdır:

```
4 rəqəmli tam ədəd = 4658
Tərsinə düzülmüş tam ədəd = 8564.
```

5.3. Keçid operatoru

Bu operator, bütün dillərdə olduğu kimi, hesablama ardıcıllığını dəyişərək idarəetməni hər hansı bir operatora vermək üçündür. Keçid operatorunun ümumi forması belədir:

`goto` *nişan***;**

Burada, *nişan* idarəetməni qəbul edəcək operatorun nişanıdır. Xatırladaq ki, nişan ya identifikator, ya da *0 – 9999* diapazonunda dəyişən işarəsiz tam ədəd ola bilər və o, `Label` operatoru ilə təsvir olunmalıdır. Nişanla operator arasında **`:`** işarəsi qoyulur.

Misal.

```
label nischan, 56;
...
...
    goto nischan;
...

56: y:=a;
...
nischan: x:=x+1;
```

Unutmayın ki, idarəetməni strukturlaşdırılmış operatorların daxilində yerləşən operatorlara vermək olmaz. Ümumiyyətlə, proqramlaşdırmada bu operatorun istifadə edilməsi məsləhət görülmür, çünki bu halda proqramın etibarlılığı, dayanıqlığı azalır. Strukturlaşdırılmış dillər bu operatordan istifadə etmədən də proqramlar tərtib etməyə imkan verir.

5.4. Boş operator

Boş operator sətirdə yalnız bir nöqtəli–vergül işarəsindən ibarət olur və proqramda operatorların yerləşə biləcəyi istənilən hissədə yazıla bilər. Boş operator nişanlana da bilər. Boş operator heç bir əməliyyat yerinə yetirmir və idarəetməni dövrün və ya tərkibli operatorun sonuna vermək üçün istifadə olunur.

5.5. Strukturlaşdırılmış operatorlar

Strukturlaşdırılmış operatorlar müəyyən qayda ilə digər operatorlardan, ifadələrdən və işçi sözlərdən yaradılır. Bu operatorlara aiddir:

- *tərkibli operator*;
- *şərti operator*;
- *seçim operatoru*;
- *dövr operatorları*;
- `With` *operatoru*.

5.5.1. Tərkibli operator

Tərkibli operator **`begin`** və **`end`** mötərizə operatorları daxilində yerləşən və bir–birindən nöqtəli–vergüllə ayrılan ixtiyari sayda operatorlar ardıcıllığından ibarət operatordur. Bu operatorun ümumi forması belədir:

```
Begin
  1–ci operator;
  ...
  n–ci operator;
end;
```

Operatorun tərkibinə daxil olan operatorların sayından asılı olmayaraq, tərkibli operator bir operator kimi qəbul edilir. Bu operator o hallarda istifadə olunur ki, hər hansı bir operatorun konstruksiyasında yalnız bir operator yazmağa icazə verilir, lakin, məsələnin məntiqinə görə isə orada bir neçə operatorun yazılması tələb olunur. Tərkibli operator adətən dövrü və şərti keçid operatorlarında istifadə olunur. Məsələnin məntiqindən asılı olaraq `end` operatorundan sonra nöqtəli–vergül işarəsi yazılmaya da bilər (bu, adətən `if` operatorunda belə olur).

Misal.

```
if m<n then
  begin
    r:=m mod n;
    n:=m;
    m:=r;
  end { burada nöqtəli–vergül işarəsi yazılmır }
else
  begin
    r:=m div n
```

```
    m:= sqr(r);
    n:= r;
  end;
```

Tərkibli operatorlar bir–birinin daxilində də yerləşə bilər.

5.5.2. Şərti operator

Şərti operatorun ümumi forması belədir:

if *şərt* **then** *1–ci operator* **else** *2–ci operator***;**

Operator icra olunduqda məntiqi tipli şərt yoxlanır: onun nəticəsi *True* (doğru) olarsa, *1–ci operator*, *False* (yalan) olduqda isə *2–ci operator* icra olunur. Xüsusi halda, `else` sözü və *2–ci operator* olmaya da bilər. Hər iki operator tərkibli operator ola bilər.

Müxtəlif proqramlarda `if` operatorunun belə yazılış formasına da tez–tez rast gəlinir:

if *şərt* **then** *1–ci operator***;** *2–ci operator***;**

Burada faktiki olaraq iki operator yazılmışdır. Belə ki, nöqtəli–vergül işarəsi `if` operatorunu başa çatdırır və sonra yeni operator icra olunmağa başlayır. Ona görə də, şərtin ödənib–ödənməməsindən asılı olmayaraq, *2–ci opreator* həmişə icra olunacaqdır.

Şərti operatorlarda məntiqi əməliyyatlardan da istifadə etmək olar. Bu halda operatorun ümumi forması belə olacaqdır:

if *məntiqi ifadə* **then** *1–ci operator* **else** *2–ci operator***;**

Bu halda, operator icra olunduqda, məntiqi ifadənin nəticəsi *True* (doğru) olarsa, *1–ci operator*, *False* (yalan) olduqda isə *2–ci operator* icra olunur. Xüsusi halda, `else` sözü və *2–ci operator* olmaya da bilər. Hər iki operator tərkibli operator ola bilər.

Misal.

```
a:=49;
b:=25;
if a>b then y:=sqrt(a) else y:=sqrt(b);
writeln(y);
```

Proqram fraqmentinin nəticəsi `y=7` olacaqdır.

Misal.

```
a:=49;
b:=25;
if a<b then y:=sqrt(a) else y:=sqrt(b);
writeln(y);
```

Proqram fraqmentinin nəticəsi `y=5` olacaqdır.

Misal.

```
a:=49;
b:=5;
if a<b then y:=sqrt(a);y:=sqr(b);
writeln(y);
```

Proqram fraqmentinin nəticəsi `y=25` olacaqdır.

Misal.

```
a:=3;
b:=8;
if (a=3) or (b=4) then y:=sqr(a) else y:=a+b;
writeln(y);
```

Bu proqram fraqmentində `a=3` *və ya* `b=4` (`(a=3) or (b=4)`) olarsa, onda $y=a^2$, digər hallarda isə $y=a+b$ hesablanır, baxdığımız hal üçün proqram fraqmentinin nəticəsi `y=9` olacaqdır.

Misal.

```
a:=3;
b:=8;
if (a=3) and (b=4) then y:=sqr(a) else y:=a+b;
writeln(y);
```

Bu proqram fraqmentində `a=3` *və* `b=4` (`(a=3) and (b=4)`) olarsa, onda $y=a^2$, digər hallarda isə $y=a+b$ hesablanır, baxdığımız hal üçün proqram fraqmentinin nəticəsi `y=11` olacaqdır.

Turbo Pascal dilində bir `if` operatoru daxilində bir neçə `if` operatoru yazmaq mümkündür. Bu isə çox mürəkkəb şərtləri yerinə yetirməyə imkan verir. Bu Pascal dilinin üstün cəhətlərindən biridir. Bir–birinin daxilində yerləşmiş `if` konstruksiyaları aşağıdakı kimidir:

if *1–ci şərt*
  then
    if *2–ci şərt*
      then
        2–ci operator
          else *1–ci operator*;

`if` operatorunun belə yazılışı aşağıdakı iki mənada başa düşülə bilər:

1.
if *1–ci şərt* **then**
  begin
    if *2–ci şərt* **then** *2–ci operator*
    else *1–ci operator*;
  end;

2.
if *1–ci şərt* **then**
  begin
    if *2–ci şərt* **then** *2–ci operator*
  end
else *1–ci operator***;**

Pascal kompilyatoru birinci mənanı daha düzgün hesab edir. Belə ki, hər bir `else` sözünə ən yaxın `if` operatoru uyğun gəlir. Ümumiyyətlə, bir–birinin daxilində yerləşən `if` operatorlarını qarışıq salmamaq üçün, onları `begin` və `end` mötərizə operatorları daxilində yazmaq məsləhət görülür.

Üç və daha çox budaqlanma yaratmaq üçün `if` operatorlarını bir–birinin daxilində yazmaq lazımdır. Bu halda `if` operatorunun aşağıdakı konstruksiyalarına icazə verilir:

1.
if *1–ci şərt*
  then
    if *2–ci şərt* **then**
      if *3–cü şərt* **then**
        ...
          if *n–ci şərt* **then** *n–ci operator* **else** *1–ci operator***;**

2.
if *1–ci şərt* **then** *1–ci operator*
  else
    if *2–ci şərt* **then** *2–ci operator*
      else
        if *3–cü şərt* **then** *3–cü operator*
          ...
            else
              if *n–ci şərt* **then** *n–ci operator***;**

Hər iki konstruksiyada `else` sözü özündən əvvəlki ən yaxın `if` operatoruna uyğun gəlir.

Misal. İki ədədin bölünməsindən alınan qisməti tapmalı.

```
program qismet;
uses crt;
label son;
var
  x,y,nat:integer;
begin
  write('Bölünəni daxil edin');
  readln(x);
  write('Böləni daxil edin');
  readln(y);
```

```
  if y=0 then
  begin
    write('Sıfra bölmə');
    goto son;
  end;
  nat:=x div y;
  writeln('Qismət =' , nat);
  son:  ;  { Boş operator }
  Readln
end.
```

Misal. $ax^2+bx+c=0$ kvadrat tənliyinin köklərinin tapılması.

```
program kvadr_tenlik;
uses crt;
var
  a,b,c,d,x,x1,x2:real;
begin
  writeln(' a,b,c –ni daxil edin');
  read (a,b,c);
  d:=sqr(b)-4*a*c;
  if d>0 then
  begin
    x1:=((-b+sqrt(d))/(2*a);
    x2:=((-b-sqrt(d))/(2*a);
    writeln('x1=',x1,'x2 =',x2);
  end    { Burada ; işarəsi yazmaq olmaz }
  else
  if d=0 then
  begin
    x:=-b/(2*a);
    writeln('köklər eynidir','x1=x2=',x);
  end    { Burada ; işarəsi yazmaq olmaz }
  else
  writeln(' köklər xəyalidir ');
  Readln
end.
```

Bu proqramda kvadrat tənliyin həlli alqoritmi, əslində, yalnız bir `if` operatoru ilə yerinə yetirilmişdir. Belə ki, əvvəlcə `d>0` şərti yoxlanır, şərt ödəndikdə, tənliyin hər iki kökü hesablanır və nəticə ekrana çıxarılır, əks halda `d=0` şərti yoxlanır və şərt ödəndikdə, tənliyin kökü hesablanır və köklərin eyni olması haqqında məlumat ekrana çıxarılır. Şərt ödənmədikdə isə sonuncu hal baş verir, yəni tənliyin həqiqi kökləri mövcud olmur və bu barədə məlumat ekranda təsvir olunur. Kvadrat tənliyin həlli alqoritmini hər üç şərti ayrı–ayrılıqda yoxlamaqla da yerinə yetirmək olar. Bu halda `if` operatorunu üç dəfə yazmaq lazımdır. Alqoritmin bu variantda proqramını belə yaza bilərik:

```
program kvadr_tenlik;
uses crt;
var
  a,b,c,d,x,x1,x2:real;
begin
  writeln(' a,b,c –ni daxil edin');
  read (a,b,c);
  d:=sqr(b) –4*a*c;
  if d>0 then
  begin
    x1:= ((–b+sqrt(d))/(2*a);
    x2:= ((–b–sqrt(d))/(2*a);
    writeln('x1=',x1,'x2 =',x2);
  end;
  if d=0 then
  begin
    x:= –b/(2*a);
    writeln('köklər eynidir','x1=x2=',x);
  end;
  if d<0 then writeln(' köklər xəyalidir ');
  Readln
end.
```

5.5.3. Seçim operatoru

Variantların sayı çox olduqda `if` operatorundan deyil, **`Case`** seçim operatorundan istifadə etmək daha əlverişli olur. Bu operatorun ümumi forması belədir:

`Case` *selektor – ifadə* **`of`**

1–ci variant **:** *1–ci operator***;**

. . .

n–ci variant **:** *n–ci operator***;**

`else` *operator***;**

`end`;

Selektor – ifadə sıralı tip olmalıdır. Hər bir variant sabitlərdən və ondan iki nöqtə (**:**) işarəsi ilə ayrılan operatordan ibarətdir. Sabitlər bir–birindən vergüllə ayrılan ixtiyari sayda qiymətlərlə və ya qiymətlərin dəyişmə diapazonları ilə təsvir olunur. Dəyişmə diapazonları arasında "**..**" işarəsi yazılır. Sabitlərin tipi *selektor – ifadənin* tipinə uyğun olmalıdır və onları `label` operatoru ilə təsvir etmək lazım deyildir.

Operator belə icra olunur. Əvvəlcə *selektor – ifadə* hesablanır və onun qiyməti variantlardakı sabitlərlə müqayisə edilir. Əgər selektorun qiyməti sabitlərin hər hansı birinə bərabər olarsa və ya göstərilmiş diapazona daxil olarsa, onda həmin varianta uyğun operator icra olunur və operator öz işini

dayandırır. Əgər *selektorun* qiyməti heç bir sabitlə üst–üstə düşməzsə, onda `else` sözündən sonrakı operator (əgər varsa) icra olunur.

Misal.

```
program secim_operat;
uses crt;
var
 abituriyent:string;
 bal:word;
begin
 read(bal);
 Case bal of
  700        : abituriyent:= '1-ci yer';
  670,680,690: abituriyent:= '2-ci yer';
  300..650   : abituriyent:= 'Tələbə';
  200..299   : abituriyent:= 'Müsabiqə';
 else
 abituriyent:='Keçmir'
 end;
 writeln('abituriyent=', abituriyent);
 Readln
end.
```

Bu proqrama görə, `bal` dəyişəninin (*selektor – ifadə*) qiyməti *700* olduqda abituriyentə `1-ci yer`, *670*, *680* və ya *690* olduqda `2-ci yer` verilir; bal dəyişəninin qiyməti *300*–dən *650*–yə kimi istənilən tam ədəd aldıqda o, `Tələbə` olur, *200*–dən *299*–a kimi istənilən tam ədəd aldıqda o, `Müsabiqə`–yə göndərilir və nəhayət bütün digər qiymətlərdə, o, `Keçmir` (əlbəttə, bu bir misaldır və Tələbə qəbulu üzrə dövlət komissiyasının qaydalarına heç bir aidiyyəti yoxdur).

Misal.

```
program secim;
uses crt;
var
 simvol: char;
begin
 readln(simvol);
 case simvol of
  '0'..'9'    : writeln('Rəqəm');
  'a'..'z'    : writeln('Kiçik hərf');
  'A'..'Z'    : writeln('Böyük hərf');
 else writeln('Başqa simvol')
 end;
 Readln
end.
```

Bu proqram daxil edilən simvolun tipini müəyyən edir.

Misal.

```
program taqim;
uses crt;
var
  apolet:word;
begin
  readln(apolet);
  case apolet of
    1000..1025  : writeln(' 1 –ci taqım ');
    1026..1050  : writeln(' 2 –ci taqım ');
    1051..1071  : writeln(' 3 –cü taqım ');
    1072..1100  : writeln(' 4 –cü taqım ');
    1101..1130  : writeln(' 5 –ci taqım ');
  end;
  Readln
end.
```

Bu proqram yaxa nömrələrinə görə kursantların hansı taqıma mənsub olmasını müəyyənləşdirir.

5.5.4. Dövr operatorları

Dövr operatorları müəyyən operatorlar qrupunu dəfələrlə yerinə yetirmək üçün istifadə edilir. Təkrar olunan operatorlar qrupu dövrün *gövdəsini* təşkil edir. Üç növ dövr operatorları vardır:

- *parametrli*;
- *ilkin şərtli*;
- *son şərtli.*

Əgər dövrlərin sayı əvvəlcədən məlumdursa, onda *parametrli*, əks halda *ilkin* və ya *son şərtli* dövr operatorları tətbiq olunur.

Goto operatorunu və ya parametrsiz **Break** prosedurunu istifadə etməklə, dövrün işini istənilən zaman dayandırmaq olar.

Continue prosedurunu tətbiq etməklə isə növbəti dövrün işini dayandırıb idarəetməni dövrün sonuna vermək olar.

Dövr operatorları bir–birinin daxilində də yerləşə bilər.

5.5.4.1. Parametrli dövr operatoru

Parametrli dövr operatorunun iki forması mövcuddur:

for *parametr* **:=** *1–ci ifadə* **to** *2–ci ifadə* **do** *operator*;

və

for *parametr* **:=** *1–ci ifadə* **downto** *2–ci ifadə* **do** *operator*;

Burada, *parametr* dövrün parametridir və o, sıralı tip istənilən dəyişən ola bilər. *1–ci ifadə* və *2–ci ifadə* dövrün parametrinin aldığı başlanğıc və son

qiymətlərdir; bu ifadələrin tipi parametrin tipinə uyğun olmalıdır. do sözündən sonrakı *operator* isə dövrün gövdəsində təkrar olunacaq operatordur. Bu operator tərkibli operator da ola bilər.

Parametrli dövr operatoru belə icra olunur. Dövrün parametri özünün başlanğıc qiymətini alır və do sözündən sonra yerləşən operator yerinə yetirilir. Bu proses, dövrün parametrinin qiyməti hər dəfə 1 vahid artırılmaqla, dövrün parametrinin sonuncu qiymətinə kimi təkrar olunur. Operatorun ikinci formasında isə dövrün parametrinin dəyişmə addımı (*–1*) –ə bərabər olur və dövrün parametrinin qiyməti ardıcıl olaraq *2–ci ifadəyə* qədər azalır.

Misal. *100* – ə qədər müsbət tam ədədlərin cəminin hesablanması.

```
Program tam_eded_cemi;
Uses crt;
Const n=100;
var i, cem : integer;
begin
  cem:=0;
  for i:=1 to n do cem:=cem+i;
  writeln('100-ə qədər tam ədədlərin cəmi=',cem:4);
  readln
end.
```

Misal. *10* – a qədər müsbət tam ədədlərin kvadratlarını, kvadrat köklərini və natural loqarifmlərini hesablayıb, nəticələri cədvəl şəklində ekrana çıxarın.

```
Program cedvel;
Uses crt;
Const n=10;
var i:integer;
    x,y,z:real;
begin
  writeln;
  writeln('Kvadratlar, köklər
   və loqarifmlər cədvəli');
  writeln;
  writeln('ədəd','  kvadrat',
  '  kvadrat kök', '  loqarifm');
  writeln;
  for i:=1 to n do
  begin
    x:=sqr(i);
    y:=sqrt(i);
    z:=ln(i);
    writeln(i:2,'    ',x:6:2,
    y:9:2,'      ',z:10:8);
  end;
  readln
end.
```

Proqramın nəticəsi şəkil 5.2 – də göstərilmişdir.

Dövr operatorları bir–birinin daxilində də yerləşə bilər. Bu halda əvvəlcə xarici dövr icra olunur, onun ardınca daxili dövr icra olunmağa başlayır. Daxili dövr tamamilə icra olunduqdan sonra, yenidən xarici dövr icra olunmağa başlayır və s.

Misal.

```
for n:=1 to 3 do     { xarici dövr }
for k:=10 to 20 do   { daxili dövr }
begin
  lines1[n,k]:=k;
  lines2[n,k]:=n;
end;
```

```
Kvadratlar, kokler ve loqarifmler cedveli

eded  kvadrat  kvadrat kok  loqarifm

 1      1.00      1.00      0.00000000
 2      4.00      1.41      0.69314718
 3      9.00      1.73      1.09861229
 4     16.00      2.00      1.38629436
 5     25.00      2.24      1.60943791
 6     36.00      2.45      1.79175947
 7     49.00      2.65      1.94591015
 8     64.00      2.83      2.07944154
 9     81.00      3.00      2.19722458
10    100.00      3.16      2.30258509
```

Şəkil 5.2. Funksiyanın cədvəlləşdirilməsi

Bu misalda, birinci `for` operatoru *xarici dövr,* ikinci `for` operatoru isə *daxili dövr* adlanır. Əvvəlcə `n=1` olur, sonra daxili dövr icra olunur, yəni `k` ardıcıl olaraq *10,11,. . .,20* qiymətlərini alır və hər dəfə `lines1` və `lines2` massivlərinin elementləri hesablanır. `k=20` olduqda daxili dövr sonuncu dəfə hesablanır və bundan sonra, yenidən xarici dövr icra olunmağa başlayır, yəni `n=2` olur, yenidən daxili dövr icra olunur və s. Beləliklə, `lines1` və `lines2` massivlərinin elementləri *3∗20=60* dəfə hesablanır, yəni dövrlər sayı *3∗20=60* olur: `n`–in hər bir qiymətində (`n=1,2,3`) `k` parametri *10*–dan *20*–yə kimi qiymət alır.

Massivlərin elementlərini daxil etmək üçün adətən parametrli dövr operatorundan istifadə edilir. Birölçülü massivlərin elementlərini daxil etmək üçün bu operator

```
for i:=1 to n do readln(a[i]);
```

kimi, ikiölçülü massivlərin elementlərini daxil etmək üçün isə

```
for i:=1 to n do
for j:=1 to m readln(a[i,j]);
```

kimi yazılır.

Qeyd. Dövrün parametrini dövrün gövdəsində dəyişdirmək olmaz.

5.5.4.2. İlkin şərtli dövr operatoru

İlkin şərtli dövrlərin strukturu şəkil 5.3 – də göstərilmişdir. Belə dövrlərdə əvvəlcə şərt yoxlanır, şərt ödəndikdə dövr icra olunur, əks halda dövr yerinə yetirilmir. Bu operatorun ümumi forması belədir:

`While` *şərt* **`do`** *operator*;

İlkin şərtli dövr operatorunu o zaman tətbiq etmək məqsədəuyğundur ki, dövrlərin sayı əvvəlcədən məlum olmur və dövr, ümumiyyətlə, yerinə

yetirilməyə də bilər. Bu operator icra olunduqda əvvəlcə məntiqi tipli şərt yoxlanır və onun nəticəsi *True* (doğru) olarsa, dövrün gövdəsi təkrarlanır. Şərt *False* (yalan) qiyməti alan kimi dövrün yerinə yetirilməsi dayandırılır. Dövrün gövdəsində şərtə təsir edilməlidir ki, dövr sonsuz təkrar olunmasın.

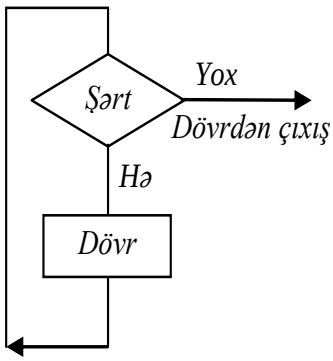
Əgər operator

While `True` **do** *operator*;

şəklində yazılarsa, onda dövr sonsuz təkrarlanacaqdır.

Misal.

```
Program İlkin_devr;
Uses crt;
var i : integer;
    sum :real;
begin
  sum:=0;
  i:=1;
  while i<=100 do
    begin
      sum:=sum+sqrt(i);
      i:=i+1;
    end;
end.
```



Şəkil 5.3. İlkin şərtli dövrlərin strukturu

Burada, `i` dəyişəninə əvvəlcə *1* qiyməti verilir və o, *100*–dən kiçik olduğu üçün, dövrün gövdəsi hesablanır, sonra `i`–nin qiyməti 1 vahid artırılaraq dövr yenidən təkrarlanır. Bu proses `i` > *100* şərti ödənənə qədər davam edir və proqram *1*–dən *100*–ə kimi tam ədədlərin kvadrat kökləri cəmini hesablayır.

5.5.4.3. Son şərtli dövr operatoru

Son şərtli dövrlərin strukturu şəkil 5.4 – də göstərilmişdir. Belə dövrlərdə əvvəlcə dövr yerinə yetirilir, sonra şərt yoxlanılır. Şərt ödənmədikdə, dövr yenidən icra olunur, şərt ödəndikdə isə dövr öz işini dayandırır.

İlkin şərtli dövr operatorundan fərqli olaraq, son şərtli dövr operatorunda şərt dövrün gövdəsindən sonra yoxlandığı üçün, dövrün gövdəsi hökmən bir dəfə yerinə yetirilir. Bu operatorun ümumi forması belədir:

Repeat

1–ci operator;
...
n–ci operator;

until *şərt*;

`Repeat` və `until` sözləri arasında yerləşən operatorlar dövrün gövdəsini təşkil edir. Dövrün gövdəsi bir dəfə yerinə yetirildikdən sonra, məntiqi tipli şərt yoxlanır və onun nəticəsi *False* (yalan) olduqda, dövrün gövdəsi yenidən yerinə

yetirilir. Şərt *True* (doğru) qiyməti alan kimi, dövrün yerinə yetirilməsi dayandırılır. Bu operatorda da dövrün sonsuz təkrar olunmaması üçün dövrün gövdəsində şərtə təsirlər edilməlidir.

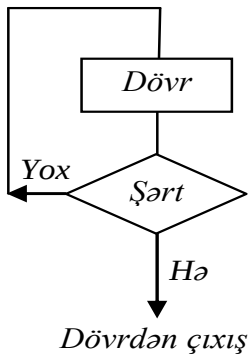
Əgər operatorun sonuncu sətri

```
until False;
```

şəklində yazılarsa, onda dövr sonsuz təkrarlanacaqdır.

Misal. İlkin şərtli dövr operatorunda həll etdiyimiz misalı son şərtli dövr operatoru vasitəsilə həll edək.

```
Program Son_devr;
Uses crt;
var i : integer;
    sum : real;
begin
  sum:=0;
  i:=1;
  Repeat
    sum:=sum+sqrt(i);
    i:=i+1;
  Until i>100;
end.
```



Şəkil 5.4. Son şərtli dövrlərin strukturu

5.5.4.4. Daxilolma operatoru

Daxilolma operatoru adətən tərkibli adların, o cümlədən, yazıların sahələrinə daha asan daxil olmaq üçün istifadə edilir. Bilirik ki, yazı sahələrinə müraciət etmək üçün yazının öz adını və ondan nöqtə ilə ayrılan yazı sahəsinin adını göstərmək lazımdır. Daxilolma operatoru ilə isə yazı sahəsinə birbaşa müraciət etmək olar. Daxilolma operatorunun ümumi forması belədir:

With *obyektin adı* **do** *operator*;

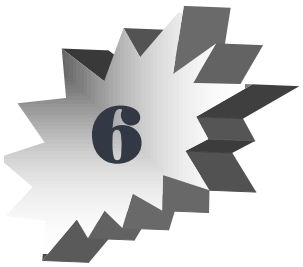
Qeyd olunmuş yazıları öyrəndikdə araşdırdığımız misalı bu operatorun köməyi ilə həll edək.

Misal.

```
With Persone do
  begin
    Name:= 'Abdullayev R.K.';
    Address:= 'Səməd Vurğun küçəsi,31';
    Married:= True;
    Salary:= 500;
  end;
```

Göründüyü kimi, `Persone` yazısının sahələrinə birbaşa müraciət edilmişdir.

Altıncı fəsil



HESABLAMA PROSESLƏRİNİN PROQRAMLAŞDIRILMASI

Bu fəsildə xətti, budaqlanan və dövrü hesablama proseslərinin proqramlaşdırılmasına, simvol və sətirlərin emalına aid ən müxtəlif xarakterli məsələlər proqramlaşdırılacaqdır. Seçilmiş məsələlər proqramlaşdırma üçün səciyyəvi xarakterli məsələlər olduğundan, belə məsələlərin proqramlaşdırılma texnikasının öyrənilməsi gələcəkdə daha mürəkkəb məsələləri proqramlaşdırmaq üçün zəruri vərdişlər əldə etməyə imkan verəcəkdir. Mürəkkəb məsələlərin proqramları əsasən belə proqram konstruksiyalarından ibarət olur.

6.1. Xətti və budaqlanan hesablama proseslərinin proqramlaşdırılması

Misal. v başlanğıc sürəti və α bucağı altında havaya atılmış cismin hündürlüyü

$$H = \frac{v^2}{2g} Sin^2(\alpha)$$

düsturu ilə hesablanır. Hündürlüyü hesablamaq üçün proqram yazın.

İlkin verilənlər kimi v = 10 *m/san* və α = 20 *dərəcə* qəbul edilmişdir. Proqramlaşdırmada triqonometrik funksiyaların arqumentləri dərəcə ilə deyil, radianla verilməlidir. Ona görə də proqram özü bucağı radiana çevirəcəkdir (`Alfa_rad` dəyişəni).

```
Program Hundur;
Uses Crt;
Const g=9.8;
      v=10;
```

```
      Alfa = 20;
Var Alfa_rad, h : real;
begin
 ClrScr;
 Alfa_rad:= Alfa*pi/180;
 H:= sqr(V*sin(Alfa_rad))/(2*g);
 Writeln('Alfa=',Alfa,'dərəcə və V=',V,
 'm/s olduqda h=',H,'m olur');
 readln
end.
```

Misal.

$a = e^{x^2+b}$, $b = \sqrt{1+x^2}$ olduqda, $y = \sin(x) + ax^2 + b\sqrt{|x+1|}$

funksiyasını hesablayın.

```
Program funksiya;
uses crt;
var a,b,x,y : real;
begin
 readln(x);
 a:= exp(x*x+b);
 b:= sqrt(1+sqr(x));
 y:= sin(x)+a*x*x+b*sqrt(abs(x+1));
 write(' ':5,'x=',x:5:2,' ':3,'y=',y:5:2);
 readln
end.
```

Misal.

$$y = \begin{cases} (x-b)/|x-0{,}79| + b^2 & , \quad 0{,}9 \le (x-0{,}79) \\ \sqrt{|(x-b)/[(x-a)^2+b]|} + ab & , \quad 0{,}6 \le (x-0{,}79) < 0{,}9 \\ \ln(x+ab) & , \quad x-0{,}79 < 0{,}6 \end{cases}$$

hesablayın. Burada: $a = e^{x+4}$, $b = \sqrt{ax}$.

```
Program budaq1;
uses crt;
var a,b,x,y,z : real;
begin
 readln(x);
 a:= exp(x+4);
 b:= sqrt(a*x);
 z:= x-0.79;
 if z>=0.9 then y:= (x-b)/abs(z)+sqr(b);
 if (z>=0.6) and (z<0.9) then
 y:= sqrt(abs((x-b)/((sqr(x-a)+b))))+a*b;
 if z<0.6 then y:= ln(x+a*b);
 write(' ':5,'z=',z:5:2,' ':3,'y=',y:5:2);
```

```
  readln
end.
```

Bu məsələdə `x-0.79` ifadəsi bir neçə dəfə hesablanmalıdır. Proqramlaşdırmada çalışmaq lazımdır ki, kompüterin yaddaşından və sürətindən israfçılıqla istifadə edilməsin, ona görə də tərtib etdiyimiz proqramda bu ifadə bir dəfə hesablanaraq `z` dəyişəninə mənimsədilmişdir.

Misal.

$$y=\begin{cases} ax^2+bx+c, & x=1 \\ ax+b, & x=2 \\ bx^2+c, & x=3 \\ cx, & x=4 \\ (ax+b)/(cx+a), & x=5 \\ e^{cx}, & x=6 \quad olduqda \\ \ln\left|ax^2+bx+c\right|, & diger\ hallarda \end{cases}$$

funksiyasını hesablayın. Burada: $a=\sin^2 x,\quad b=\cos x+\sin x,\quad c=e^{ax+b}$.

Bu məsələdə variantların sayı çox olduğu üçün `if` operatorundan istifadə etmək əlverişli deyildir. Ona görə də `case` operatorunu tətbiq edəcəyik.

```
Program funks;
uses crt;
var a,b,c,y:real;
    x:byte;
begin
 readln(x);
 a:=sqr(sin(x));
 b:=cos(x)+sin(x);
 c:=exp(a*x+b);

 Case x of
  1: y:=a*x*x+b+x+c;
  2: y:=a*x+b;
  3: y:=b*x*x+c;
  4: y:=c*x;
  5: y:=(a*x+b)/(c*x+a);
  6: y:=exp(c*x);
  else
  y:=ln(abs(a*x*x+b*x+c));
 end;

 writeln(' ':5,'x=',x:5,' ':3,'y=',y:5:2);
 readln
end.
```

Misal. Klaviaturadan daxil edilən *n* tam ədədinin ($0 \le n \le 15$) onaltılıq say sisteminə çevrilməsi.

```
program cevirme;
uses crt;
var
  n : integer;
  ch : char;
begin
  write('n=');
  readln(n);
  if (n>0) and (n<=15) then
  begin
    if n<10 then
    ch:=chr(ord('0')+n)
    else
    ch:=chr(ord('A')+n-10);
    writeln('n=',ch);
  end
    else
      writeln('Səhvdir');
  readln
end.
```

Proqramın alqoritmi simvolların **ASCII** kodlarına əsaslanır. **ASCII** simvollar cədvəlinə əsasən *0, 1, 2, . . ., 9* ədədlərinin kodları uyğun olaraq *48,49,. . .,57*–dir. Ona görə də proqramda, `n<10` olduqda, `ch:=chr(ord('0')+n)` yazılmışdır, yəni, məsələn, *n=4* üçün `ch=chr`*(48+4)*`=chr`*(52)=4* olacaqdır. *4* isə onaltılıq say sistemində də *4* kimi yazılır. `n>10` olduqda, `ch:=chr(ord('A')+n-10)` yazılmışdır, yəni, məsələn, *n=13* üçün `ch=chr`*(97+13–10)*`=chr`*(100)* olacaqdır ki, *100* ədədinə də uyğun kod `chr`*(100)=D* olacaqdır, *13* ədədi isə onaltılıq say sistemində *D* kimi yazılır.

6.2. Dövrü hesablama proseslərinin proqramlaşdırılması

Proqramlaşdırma texnologiyasında bir sıra hesablama konstruksiyalarından istifadə edilir ki, bu konstruksiyalar hansı proqramlaşdırma dilinin tətbiq edilməsindən asılı olmayaraq, demək olar ki, eyni struktura malik olur. Belə konstruksiyalardan ən vacibləri cəmləmə, hasil və sayğac alqoritmləridir. Bu alqoritmlərə ayrılıqda baxaq.

Cəmləmə alqoritmi. Cəmləmə əməliyyatını yerinə yetirmək üçün, əvvəlcə cəmi yadda saxlayacaq hər hansı bir dəyişənə sıfır qiyməti mənimsədilir, sonra isə dövr təşkil olunaraq, həmin dəyişənin üzərinə cəmlənəcək dəyişən və ya massivin elementləri əlavə edilir:

```
...
s:=0;
For i:=1 to n do
s:=s+a[i];
...
```

Dövrün birinci addımında, yəni `i=1` olduqda, `s` adlı xanada cəmin başlanğıc qiyməti – `a[1]` yerləşəcək. İkinci addımda, `i=2` olduqda, `s` xanasında artıq `a[1]` qiyməti olduğu üçün, onun üzərinə `a[2]` əlavə olunacaqdır. Beləliklə, ikinci addımda `s` xanasında `a[1]+a[2]` cəmi olacaqdır. `n` – ci addımda isə massivin bütün elementləri cəmlənəcəkdir.

Hasil alqoritmi. Cəmləmə alqoritminə analoji olaraq, hasili yadda saxlayacaq hər hansı bir dəyişənə vahid qiyməti mənimsədilir, sonra isə dövr təşkil olunaraq hasili tapılacaq dəyişən və ya massivin elementləri həmin dəyişənə vurulur.

```
...
h:=1;
For i:=1 to n do
h:=h*a[i];
...
```

Bu alqoritmin iş prinsipi cəmləmə alqoritminin işləmə prinsipi ilə eynidir.

Sayğacların yaradılması. Sayğac proqramlaşdırmada ən vacib konstruksiyadır. Demək olar ki, elə bir proqram yoxdur ki, orada sayğaclardan istifadə edilməsin. Sayğac vasitəsilə bu və ya digər elementlərin sayı tapılır. Sayğacların strukturu belədir:

```
...
k:=0;
for i:=1 to n do
  if a[i]>0 then k:=k+1;
...
```

və ya

```
...
k:=0;
for i:=1 to n do
  if a[i]>0 then inc(k);
...
```

Bu sayğacla `a[i]` massivinin müsbət elementlərinin sayı (`k`) tapılır. Mənfi elementlərin sayını hesablayan sayğac belədir:

```
...
k:=0;
for i:=1 to n do
  if a[i]<0 then k:=k+1;
...
```

`a ≤ x[i] ≤ b` aralığında yerləşən elementlərin sayını hesablayan sayğac belədir:

```
...
k:=0;
for i:=1 to n do
  if (x[i] >=a) and (x[i] <=b) then inc(k);
...
```

Misal. $S = \sum_{i=1}^{n} \frac{1}{i^2}$ cəminin hesablanması.

```
program cem;
uses crt;
var
  i, n: word;
  t, add, cem: real;
begin
  write('n –i daxil edin n = ');
  readln(n);
  cem:=0;
  for i:= 1 to n do
  begin
    t:= 1.0/i;
    add:= sqr(t);
    cem:= cem+add;
  end;
  writeln('cəm = ', cem);
  write('Enter klavişini basin:');
  readln;
end.
```

Cəmləmə əməliyyatını yerinə yetirmək üçün dövr təşkil etməzdən əvvəl, hər hansı bir dəyişənə sıfır qiyməti mənimsədərək (`cem:=0;`) dövrün daxilində onun üzərinə cəmlənəcək parametr əlavə edilmişdir (`cem:=cem+add;`).

Misal. $\sum_{i=1}^{1000} \frac{1}{n^5}$ cəmini düz və əks istiqamətlərdə hesablayıb, onların fərqini tapın.

```
program duz_eks_cem;
uses crt;
var x, duzcem, ekscem : real;
    k : word;
begin
  clrscr;
        { düz istiqamətdə cəmləmə }
  duzcem:= 0.0;
  for k:=1 to 1000 do
```

```
  begin
    x:= k;
    duzcem:= duzcem+1.0/(exp(5.0*ln(x)));
  end;
        { əks istiqamətdə cəmləmə }
  ekscem:= 0.0;
  for k:=1000 downto 1 do
  begin
    x:= k;
    ekscem:= ekscem+1.0/(exp(5.0*ln(x)));
  end;
  Writeln('irəliyə cəm    =',duzcem);
  writeln('geriyə cəm   =',ekscem);
  writeln('cəmlərin fərqi  =',duzcem-ekscem);
  readln
end.
```

Bu proqramdan çox qəribə bir nəticə çıxır: məlumdur ki, toplananların yerini dəyişdikdə cəm dəyişmir, lakin, bu proqramdan alınan nəticəyə baxdıqda görürük ki, hesablamanı düz və əks istiqamətlərdə apardıqda cəm dəyişir. Məsələ ondadır ki, kompüter eyni tərtibli dəyişənləri topladıqda daha kiçik xətalara yol verir. Ona görə də bu misalın həllində əks istiqamətdə alınan cavab daha düzgündür. Çünki, məsələn, $n=1,2,3$ qiymətlərində $1+1/2^5+1/3^5$ toplananları bir–birindən çox fərqlənir. Lakin, məsələn, $n=1000, 999, 998$ qiymətlərində $1/1000^5+1/999^5+1/998^5$ toplananları isə bir–birindən çox fərqlənmir.

Misal. $F(x)=x/(1+x)$ funksiyasını cədvəlləşdirin.

```
program cedvel;
uses wincrt;
var x, f: real;
    k: word;
begin
  clrscr;
  x:= 0.0;
  writeln('f(x)=x/(1+x)
  funksiyasının qiymətlər cədvəli');
  writeln;
  Writeln('sıra N','x':10,'f(x)':20);
  writeln;
  for k:=0 to 50 do
   begin
     f:= x/(1.0+x);
     writeln(' ',k,' ',x:12:4, f:20:10);
     x:= x+0.1;
     if k mod 10 = 9 then readln;
   end;
  readln
end.
```

Bu proqram hər *10* sətirdən sonra dayanır. Bunu `if k mod 10=9 then readln;` sətri təmin edir. Parametrsiz `readln;` proseduru *Enter* klavişinin basılmasını gözləyir və bu klaviş basıldıqda hesablama davam etdirilir. Proqramın nəticəsinin bir hissəsi şəkil 6.1 – də göstərilmişdir:

Misal. *n* sayda tam ədədlər içərisindən *3* ədədinə tam bölünməyən ədədləri, onların sayı və cəmini tapmalı.

```
Program uche_bolme;
Uses wincrt;
Const n=300;
var k,i: word; s:longint;
begin
 writeln;
 K:=0;s:=0;
 For i:=1 to n do
 If not(i mod 3 = 0) then
  Begin
    Inc(k);
    s:=s+i;
    Write (' ',i:3);
    if k mod 10 = 9 then writeln;
  end;
 writeln;
 writeln;
 writeln('  3-ə bölünməyən ədədlərin sayı=',k:4);
 writeln('  3-ə bölünməyən ədədlərin cəmi=',s:10);
 readln
end.
```

```
f(x)=x/(1+x) funksiyasinin qiymetler cedveli

sira N        x              f(x)

  0       0.0000       0.0000000000
  1       0.1000       0.0909090909
  2       0.2000       0.1666666667
  3       0.3000       0.2307692308
  4       0.4000       0.2857142857
  5       0.5000       0.3333333333
  6       0.6000       0.3750000000
  7       0.7000       0.4117647059
  8       0.8000       0.4444444444
  9       0.9000       0.4736842105
```

Şəkil 6.1. Funksiyanın cədvəlləşdirilməsi

Nümunə üçün `n`=*300* qəbul edilmişdir. Proqram belə işləyir. *3* ədədinə tam bölünməyən ədədlərin cəmini tapmaq üçün `s` dəyişəninə, ədədlərin sayını tapmaq üçün isə `k` dəyişəninə sıfır qiymətləri mənimsədilmişdir. Sonra, dövr təşkil olunaraq, ədədlərin *3* ədədinə bölünməsindən alınan qalıq tapılır. Əgər qalıq sıfra bərabər deyildirsə, yəni `if not(i mod 3=0)`, onda `k` sayğacının üzərinə vahid, `s` dəyişəninin üzərinə isə həmin ədəd əlavə olunur. İkinci `if` operatoru hər *10* ədəddən bir yeni sətrə keçmək üçün yazılmışdır. Proqramın nəticəsi şəkil 6.2 – də göstərilmişdir.

6.3. Birölçülü massivlər üzərində əməliyyatlar

Massivlərin elementlərini daxil etmək üçün sadəcə olaraq `read` və ya `readln` prosedurundan istifadə etmək mümkün olmur. Çünki bu prosedurla hər dəyişənə yalnız bir qiymət daxil etmək olar, massivlərin isə elementləri çoxdur. Ona görə də massivlərin elementlərini daxil etmək üçün adətən parametrli dövr operatorundan istifadə edilir:

```
for i:=1 to n do readln(a[i]);
```

Misal. *A(15)* massivinin elementlərini ekrandan daxil edin. *B(15)* massivinin elementlərini elə qiymətləndirin ki, cüt nömrəli elementlər *A*–nın uyğun elementlərindən *3* dəfə çox, tək nömrələr isə *5* vahid az olsun.

```
program tek_cut_element;
uses crt;
const n=15;
var
   A,B: array [1..n] of real;
   i:byte;
begin
 writeln('Massivin elementlərini daxil edin');
 for i:= 1 to n do readln(A[i]);
 clrscr;
 for i:= 1 to n do
 if Odd(i) then B[i]:=A[i]+5
 else B[i]:=A[i]*3;
 for i:= 1 to n do
 writeln(B[i]);
 readln
end.
```

Təqdim olunmuş bu proqramda tək və cüt nömrəli elementləri müəyyən etmək üçün `Odd` funksiyasından istifadə edilmişdir. Xatırlayaq ki, arqumenti tək ədəd olduqda, bu funksiyanın nəticəsi *True*, cüt ədəd olduqda isə *False* olur.

```
  1   2   4   5   7   8  10  11  13
 14  16  17  19  20  22  23  25  26  28
 29  31  32  34  35  37  38  40  41  43
 44  46  47  49  50  52  53  55  56  58
 59  61  62  64  65  67  68  70  71  73
 74  76  77  79  80  82  83  85  86  88
 89  91  92  94  95  97  98 100 101 103
104 106 107 109 110 112 113 115 116 118
119 121 122 124 125 127 128 130 131 133
134 136 137 139 140 142 143 145 146 148
149 151 152 154 155 157 158 160 161 163
164 166 167 169 170 172 173 175 176 178
179 181 182 184 185 187 188 190 191 193
194 196 197 199 200 202 203 205 206 208
209 211 212 214 215 217 218 220 221 223
224 226 227 229 230 232 233 235 236 238
239 241 242 244 245 247 248 250 251 253
254 256 257 259 260 262 263 265 266 268
269 271 272 274 275 277 278 280 281 283
284 286 287 289 290 292 293 295 296 298
299

 3-e bolunmeyen ededlerin sayi= 200
 3-e bolunmeyen ededlerin cemi=     30000
```

Şəkil 6.2. *3* ədədinə bölünməyən ədədlər

Misal. *n* sayda elementdən ibarət massivin elementlərinin cəminin hesablanması.

```
program mas_sem;
uses crt;
const n=25;
var
   a:array[1..n] of real;
   s:real;
   i:integer;
begin
  s:=0;
  i:=1;
  while i<=n do
  begin
    s:=s+a[i];
    i:=i+1;
  end;
  writeln('Cəm =', s);
  readln
end.
```

Bu proqramla, artıq bizə məlum olan alqoritm üzrə, cəmləmə əməliyyatı yerinə yetirilmişdir, fərq ondadır ki, burada ilkin şərtli dövr operatorundan istifadə olunmuşdur.

Misal. Tam ədədlərdən təşkil olunmuş massivin birinci mənfi elementinin axtarılması.

```
program menfi_el;
uses crt;
const n=15;
      yes:boolean = False;
var
  mas:array[1..n] of integer;
  i:byte;
begin
  writeln('Massivin elementlərini daxil edin');
  for i:= 1 to n do
  begin
    write('mas[',i,']=');
    readln(mas[i]);
  end;
  for i:= 1 to n do
  begin
    if mas[i]>=0 then continue;
    writeln('Birinci mənfi element=',
           mas[i], 'Nömrə= ',i);
    yes:=True;
    break;
```

```
  end;
  if not yes then
  writeln('Mənfi element yoxdur');
  readln
end.
```

Bu proqramda, dövr daxilində massivin elementlərinin müsbət olması yoxlanır: əgər element müsbətdirsə, onda `continue` proseduru ilə növbəti elementin yoxlanmasına keçilir. Mənfi element rast gəldikdə isə `yes` dəyişəninə *True* qiyməti mənimsədilərək `break` proseduru ilə dövrün işi dayandırılır. Əgər `yes` dəyişəninin qiyməti *False* olarsa, onda proqram massivin mənfi elementinin olmaması haqqında məlumat verir (`if not yes then writeln('Mənfi element yoxdur');`).

Misal. *X(n)* massivinin ən böyük elementinin tapılması.

```
program max_element;
uses crt;
const n=20;
var
   x:array[1..n] of real;
   i: integer; M: real;
begin
  writeln('Massivin elementlərini daxil edin');
  for i:= 1 to n do
    begin
      write('X[',i,']=');
      readln(X[i]);
    end;
  M:=X[1];
  for i:= 2 to n do
  if M < X[i] then M:=X[i];
  writeln('Massivin ən böyük elementi = ', M);
  readln
end.
```

Massivin ən böyük elementinin tapılması alqoritmi belədir: massivin birinci elementi hər hansı bir dəyişənə mənimsədilir (`M:=X[1];`) və dövr daxilində bu element növbəti elementlərlə müqayisə edilir. Hansı element böyük olarsa, o, element həmin dəyişənə mənimsədilir (`if M<X[i] then M:=X[i];`). Bu proses ən böyük element tapılana qədər davam etdirilir.

Kompüter texnologiyasında və proqramlaşdırma texnikasında ən çox istifadə edilən alqoritmlərdən biri massivlərin elementlərinin nizamlanması (elementlərin artma və ya azalma sırası ilə düzülməsi) alqoritmidir. Elementlərin nizamlanması üçün bir neçə metod mövcuddur.

Mövcud metodlardan biri belədir. Verilmiş ilkin ədədlərdən ən kiçiyi tapılır və o, yeni massivin birinci mövqeyinə yazılır, ilkin massivdən isə həmin ədəd yox edilir. İlkin massivin yerdə qalan elementləri içərisindən ən kiçiyi tapılır,

yeni massivin ikinci mövqeyinə yazılmaqla, ilkin massivdən həmin ədəd yox edilir. Növbəti elementlər üçün bu prosesi davam etdirməklə, sonuncu ən böyük ədədi alacağıq ki, bu ədəd yeni massivin sonuncu elementi olacaqdır. Belə alqoritmdə biz faktiki olaraq eyniölçülü iki massivdən – ilkin və yeni massivlərdən istifadə edirik. Proqramlaşdırmada isə belə israfçılığa yol vermək olmaz (massiv çox böyük ölçülü ola bilər). İndi bu alqoritmi təkmilləşdirək. Xətti nizamlama adlanan aşağıdakı alqoritmi tərtib edək.

1. a_i ($i = 1, . . . ,n$) massivinin ən kiçik elementini taparaq onu a_1 elementinə yazaq. Bunun üçün a_1 elementini verilmiş ilkin massivin bütün növbəti elementləri ilə müqayisə edək. Əgər hər hansı element a_1–dən kiçik olarsa, onda onların yerlərini dəyişərək müqayisəetməni sonuncu elementə qədər davam etdirmək lazımdır. Belə müqayisə zamanı, əgər hər hansı element a_1–dən böyük və ya ona bərabər olarsa, onda elementin yerini dəyişmədən növbəti elementlə müqayisəni davam etdirmək lazımdır.

2. Birinci mövqedəki elementi nəzərdən atmaqla, göstərilən prosesi ikinci mövqedən təkrar etmək lazımdır. Başqa sözlə, yerdə qalan elementlər içərisindən göstərilən üsulla ən kiçik elementi tapıb onu a_2 elementinə yazmaq lazımdır.

3. Bu əməlləri, axırıncı element müstəsna olmaqla, bütün elementlər üçün icra etmək lazımdır.

Bu alqoritmin proqramını aşağıdakı kimi yaza bilərik.

```
program xetti_nizam;
uses crt;
const n=4;
var i: word;
     a: array[1..n] of integer;
     yeni,x: integer;
begin
  for i:=1 to n do read(a[i]);
  for i:=1 to n-1 do
  for yeni:=i+1 to n do
  if a[i]>a[yeni] then
    begin
      x:=a[i];
      a[i]:=a[yeni];
      a[yeni]:=x;
    end;
  for i:=1 to n do
    begin
      Write(a[i]:5,' ':3);
      if i=10 then write(#10#13);
    end;
  readln
end.
```

Bu proqramla ekrana hər sətirdə *10* qiymət çıxarılır. Sətirdən–sətrə keçidi

```
if i=10 then write(#10#13);
```

operatoru təmin edir. Burada **#10** və **#13** kodları, uyğun olaraq, *sətrin dəyişdirilməsi* və yazı makinasının *karetkasının qaytarılması* idarəedici simvollarıdır, başqa sözlə, **#10#13** kodları ilə yeni sətrə keçilir.

Tərtib etdiyimiz bu proqram massivin elementlərini yalnız artma sırası ilə düzür. Massivin elementlərini azalma sırası ilə düzmək üçün `if` operatorundakı "`>`" işarəsini "`<`" işarəsi ilə əvəz etmək lazımdır. Lakin proqramı elə tərtib etmək olar ki, o elementlərin artma və ya azalma sırası ilə düzülməsini özü seçsin. Həmin proqram belə olacaqdır:

```
program xetti_nizam;
uses crt;
const n=4;
var i:word;
     a:array[1..n] of integer;
     yeni,x:integer;
     istiqamet:string;
begin
  write('Artma yoxsa azalma?');
  read(istiqamet);
  for i:=1 to n do readln(a[i]);
  if istiqamet='artma'
  then
{ artma sırası ilə düzülüş }
   begin
     for i:=1 to n-1 do
     for yeni:=i+1 to n do
     if a[i]>a[yeni] then
       begin
         x:=a[i];
         a[i]:=a[yeni];
         a[yeni]:=x;
       end;
   end   { ; işarəsi qoymaq olmaz }
  else
{ azalma sırası ilə düzülüş }
   begin
     for i:=1 to n-1 do
     for yeni:=i+1 to n do
     if a[i]<a[yeni] then
       begin
         x:=a[i];
         a[i]:=a[yeni];
         a[yeni]:=x;
       end;
   end;
   for i:=1 to n do
     begin
       Write(a[i]:5,' ':3);
       if i=10 then write(#10#13);
```

```
      end;
  readln
end.
```

Bu proqrama sətir tipli `istiqamet` dəyişəni daxil edilmişdir ki, ona `artma` qiyməti daxil etdikdə massivin elementləri artma sırası ilə, ixtiyari fərqli qiymət daxil etdikdə isə azalma sırası ilə düzüləcəkdir.

Daha bir metoda baxaq. Bu metod "qabarcıq" metodu adlanır. Bu alqoritmdə ən kiçik ədəd elementlər içərisindən yuxarıya qalxır, sanki "üzür", ən böyük elementlər isə aşağıya hərəkət edir, sanki "batır". Metoda ona görə "qabarcıq" adı verilmişdir ki, suda hava qabarcıqlarının hərəkətinə analoji olaraq, massivin kiçik elementləri massivin başlanğıcına ("suyun səthinə") qalxır. Bu metodun mahiyyəti belədir. Məlumdur ki, massivin elementləri artma sırası ilə düzülmüşdürsə, onda hər bir element özündən sonrakı elementdən kiçik olacaqdır. Bu sadə fakt "qabarcıq" metodunun əsas ideyasını təşkil edir. Belə ki, məhz bu qayda massivlərin elementlərini yeni üsulla müqayisə etməyə əsas verir, yəni kifayətdir ki, yalnız bir cüt qonşu ədədləri müqayisə edək. Əgər xətti nizamlama alqoritmində a_1 elementini yerdə qalan bütün elementlərlə müqayisə etmək lazım gəlirdisə, bu alqoritmdə a_1 elementini yalnız a_2 ilə müqayisə etmək kifayətdir. Əgər $a_1<a_2$ olarsa, onda a_2 elementini a_3 ilə, a_3 elementini a_4 ilə və s. müqayisə etmək lazımdır. Belə müqayisə zamanı hər hansı element birbaşa özündən sonra gələn elementdən böyük olarsa, onda onların yerlərini dəyişməklə müqayisəetməni sonadək davam etdirmək lazımdır. Lakin, yerdəyişmə zamanı elə ola bilər ki, a_3 elementi a_2–dən kiçik olsun. Bu isə o deməkdir ki, növbəti dövr başa çatdıqdan sonra, massivin başlanğıcına qayıdaraq elementlərin cüt–cüt müqayisəetmə prosesini təkrarlamaq lazımdır. Beləliklə, "qabarcıq" metodunun alqoritmi belə olacaqdır: massivin qonşu elementlərini, yəni a_1 elementini a_2 ilə, a_2 elementini a_3 ilə, a_3 elementini a_4 ilə və nəhayət a_{n-1} elementini a_n ilə müqayisə etməklə, müqayisəetmə prosesini o qədər təkrar etmək lazımdır ki, elementlərin yeri artıq dəyişməsin.

İlkin vəziyyətdə massivlərin elementləri müəyyən dərəcədə nizamlanmış olduqda "qabarcıq" metodu daha səmərəli olur. Ona görə də bu metodu ilk növbədə əvvəlcədən qismən nizamlanmış massivlər üçün tətbiq etmək daha sərfəlidir. Ümumi halda isə hər iki metod eyni dərəcədə səmərəlidir. "Qabarcıq" alqoritmində dövrlərin sayı əvvəlcədən məlum olmur, xətti nizamlama alqoritmində isə dövrlər sayı həmişə $n-1$ sayda olur.

"Qabarcıq" alqoritminin isə proqramını aşağıdakı kimi yaza bilərik:

```
program qabarciq_nizam;
uses crt;
const n=5;
var i:word;
     a:array[1..n] of integer;
     x:integer;
     sort:boolean;
begin
  for i:=1 to n do read(a[i]);
```

```
  Repeat
    sort:=FALSE;
    for i:=1 to n-1 do
    if a[i]>a[i+1] then
      begin
        x:=a[i];
        a[i]:=a[i+1];
        a[i+1]:=x;
        sort:=TRUE;
      end;
  UNTIL sort=FALSE;
  for i:=1 to n do
    begin
      Write(a[i]:5,' ':3);
      if i=10 then write(#10#13);
    end;
  readln
end.
```

Burada, məntiqi tipli `sort` dəyişəninə `False` (yalan) qiyməti mənimsədilir. Dövr daxilində (`for`) massivin elementlərinin yerdəyişməsi baş verərsə, `sort` dəyişəninin qiyməti `True` (doğru) olur və növbəti elementləri müqayisə etmək üçün son şərtli dövr icra olunmağa başlayır. Massivin elementlərinin yeri dəyişmədikdə isə bu dəyişənin qiyməti `False` olaraq qalır, bu isə massivin elementlərinin nizamlanması deməkdir, ona görə də son şərtli dövr öz işini başa çatdırır.

Misal. *n* sayda təsadüfi ədədlər yaradaraq onların [*a, b*] parçasına daxil olanlarının sayını tapmalı; bütün təsadüfi ədədləri və [*a, b*] parçasına daxil olan ədədləri ekrana çıxarmalı.

```
program TES_EDED;
uses crt;
const a=10;b=30;
      n=20;
var x : array[1..n] of integer;
    i,k :byte;
begin
  k:=0;
  Writeln('Təsadüfi ədədlər generatorunun',
  #10#13,'yaratdığı bütün təsadüfi ədədlər:');
  Randomize;
  for i:=1 to n do
    begin
      x[i]:= random(50);
      write(x[i]:4);
    end;
  writeln(#10#13);
  writeln('intervala daxil olan ədədlər:');
```

```
  for i:=1 to n do
   begin
     if(x[i]>=a) and (x[i]<=b) then
     begin
       inc(k);
       write('   ',x[i]:2);
     end;
   end;
  writeln(#10#13);
  writeln('intervala daxil olan ədədlərin sayı',k:3);
end.
```

Proqramın əsas məqsədi təsadüfi ədədlərin yaradılmasını nümayiş etdirməkdən ibarətdir. Təsadüfi ədədləri yaratmaq üçün `random(50);` funksiyasından istifadə edilmişdir. Bu funksiya $0 \le x < 50$ aralığında təsadüfi ədədlər yaradır. Lakin, bu funksiyaya müraciət etməzdən əvvəl, `Randomize;` prosedurunu çağırmaq lazımdır. Bu prosedur təsadüfi ədədlər generatorunu aktivləşdirir. Siz, proqramı dəfələrlə icra etdikdə hər dəfə tamamilə müxtəlif nəticələr alacaqsınız. Buna səbəb `Randomize;` prosedurudur. Əgər bu proseduru pozsanız, proqram həmişə eyni nəticələr verəcəkdir. Bu proqramın bir fraqmentdə həlli şəkil 6.3 – də göstərilmişdir.

```
Tesadufi ededler  generatorunun
yaratdiqi butun tesadufi ededler:
19  6 48 38 10 37 29 10 34  1  7 37 14 31 12 45  1 41 18  1

intervala daxil olan ededler:
 19   10   29   10   14   12   18

intervala daxil olan ededlerin sayi  7
```

Şəkil 6.3. Təsadüfi ədədlər

6.4. Matrislər üzərində əməliyyatlar

Bildiyimiz kimi, matrislər ikiölçülü massivlərlə ifadə olunur. İkiölçülü massivlərin elementlərini daxil etmək üçün bir–birinin daxilində yerləşən ikiqat dövr təşkil etmək lazımdır. Dövr operatoru kimi adətən parametrli dövr operatoru istifadə edilir. Bu dövr operatorunun birincisinin parametri kimi massivin birinci indeksi, yəni matrisin sətirlərinin nömrələri, ikincisinin parametri kimi massivin ikinci indeksi, yəni matrisin sütunlarının nömrələri istifadə edilir. Daxili dövrün gövdəsində isə `read` və ya `readln` proseduru yazılır:

```
for i:=1 to n do
for j:=1 to m readln(a[i,j]);
```

Belə daxiletmə zamanı massivin hansı elementinin daxil edilməsini görmək mümkün olmur və adətən mexaniki səhvlərə yol verilir.

Aşağıdakı proqram konstruksiyası isə massivin elementlərini sətirbəsətir daxil etməyə imkan verir:

```
for i:=1 to n do
begin
  write(i,'-ci sətir elementlərini daxil edin:');
  for j:=1 to m do read(a[i,j]);
end;
```

Belə daxiletmə zamanı proqram ekrana `1-ci sətir elementlərini daxil edin:` məlumatı verir Buna cavab olaraq massivin birinci sətir elementləri, aralarında bir probel simvolu qoyulmaqla, bir sətirdə yığılır və sətrin sonunda *Enter* klavişi basılır. Bundan sonra, növbəti sətirlərin daxil edilməsi tələb olunduqda, eyni qayda ilə növbəti sətirlərin elementlərini daxil etmək lazımdır.

Matrisin elementlərini sətir və sütunlar şəklində ekrana çıxarmaq üçün aşağıdakı proqram konstruksiyasından istifadə etmək daha məqsədəuyğundur:

```
for i:=1 to n do
begin
  for j:=1 to m do write(c[i,j],'  ');
  writeln;
end;
```

Massivlər üzərində əməliyyatlara aid misallara baxaq.

Misal. *A(n,m)* massivinin sətir elementlərinin hasilləri cəminin hesablanması.

```
program element_hasil_cemi;
uses crt;
const n=3;m=2;
type
   matr=array[1..n,1..m] of real;
var
   A: matr;
   i,j:integer;
   z,s:real;
begin
  for i:=1 to n do
    begin
      write(i,'-ci sətir elementlərini daxil edin:');
      for j:=1 to m do read(a[i,j]);
    end;
  clrscr;
  S:=0.0;
  for i:=1 to n do
    begin
      Z:=1;
      for j:=1 to m do Z:=Z*A[i,j];
```

```
      S:=S+Z;
    end;
  writeln('Nəticə = ',S);
  readln
end.
```

İlkin verilənləri daxil etdikdə, proqram ekrana `1-ci sətir elementlərini daxil edin:` məlumatı verir. Buna cavab olaraq massivin birinci sətir elementləri, aralarında bir probel simvolu qoyulmaqla, bir sətirdə yığılır və sətrin sonunda *Enter* klavişi basılır. Bundan sonra, növbəti sətirlərin daxil edilməsi tələb olunacaqdır. Massivin sətir elementlərinin hasilini tapmaq üçün, dövr təşkil etməzdən əvvəl, hər hansı bir dəyişənə vahid qiyməti mənimsədərək (`Z:=1;`) dövrün daxilində onu hasili tapılacaq parametrə (bizim misalda massivin elementlərinə) vurmaq lazımdır (`Z:=Z*A[i,j];`). Xüsusi diqqət yetirin ki, `Z:=1;` operatoru iki dövr operatorunun arasında yazılmışdır. Bu ona görə belə edilmişdir ki, matrisin bütün elementlərinin hasilini deyil, sətir elementlərinin hasilini tapmaq və onları cəmləmək lazımdır. Əgər `Z:=1;` operatoru `S:=0.0;` operatorundan ya əvvəl, ya da sonra yazılsaydı, onda matrisin bütün elementlərinin hasilləri cəmi hesablanardı. Cəmləmə isə, həmişə olduğu kimi, dövrün daxilində məlum `S:=S+Z;` alqoritmi üzrə yerinə yetirilmişdir.

Misal. *X(n,m)* matrisinin müsbət elementlərinin kvadratları cəminin mənfi elementlərin sayına olan nisbətini tapın.

```
program element;
uses crt;
const n=2;m=2;
var
   X:array[1..n,1..m]of real;
   i,j,k:byte;
   cem,nisbet:real;
begin
  for i:=1 to n do
    begin
      write(i,'-ci sətir elementlərini daxil edin:');
      for j:=1 to m do read(x[i,j]);
    end;
  clrscr;
  cem:=0.0;
  k:=0;
  for i:=1 to n do
  for j:=1 to m do
  if x[i,j]>0 then cem:=cem+x[i,j] { müsbət elementlərin cəmi }
  else k:=k+1;      { mənfi elementlərin sayı }
  if k=0 then
    begin
      writeln('Mənfi element yoxdur');
```

```
      exit;
    end;
  nisbet:=cem/k;
  writeln('Cavab=',nisbet);
  readln
end.
```

Misal. Paskal üçbucağının qurulması.

Paskal üçbucağı *n* sayda sətirdə yerləşən *n* sayda natural ədədlərdən tərtib olunur və bu üçbucağın yan tərəflərindəki ədədlər *1*–ə bərabər olur. Hər bir üçbucağın oturacağındakı ədədlər isə özündən yuxarıdakı sətirdə ona ən yaxın olan iki ədədin cəminə bərabərdir. *n* sayda sətirdən ibarət Paskal üçbucağını qurmalı.

```
program Pascal_uchbucaqi;
uses crt;
const n=10;
var i,k,j: byte;
     s: string;
     a: array[1..n,1..n] of integer;
begin
  writeln;
  a[1,1]:=1;
  for k:=2 to n do
  for i:=1 to k do
  if (i=1) or (i=k) then a[k,i]:=1
  else a[k,i]:= a[k-1,i-1]+a[k-1,i];
  s:=' ';
  for i:=1 to n do s:=s+'  ';
  for i:=1 to n do
    begin
      write(s);
      for j:=1 to i do write(' ',a[i,j]:3);
      writeln;
      delete (s,1,2);
    end;
  readln
end.
```

Proqramın nəticəsi şəkil 6.4 – də göstərilmişdir.

Misal. Matrisin transponirə edilməsi.

```
program matrisin_transp_edilmasi;
uses crt;
var a: array[1..3,1..4] of real;
     b: array[1..4,1..3] of real;
     i,j,k: word;
begin
  for i:=1 to n do
    begin
```

```
      write(i,'-ci sətir elementlərini daxil edin:');
      for j:=1 to m do read(a[i,j]);
    end;
  for j:=1 to 4 do
    begin
      for i:=1 to 3 do
      begin
        b[j,i]:=a[i,j];
        write(b[j,i]:4:2,'  ');
      end;
      writeln;
    end;
  readln
end.
```

Əgər proqrama

$$A=\begin{pmatrix}1 & 2 & 3 & 4\\ 5 & 6 & 7 & 8\\ 9 & 10 & 11 & 12\end{pmatrix}$$

matrisi daxil edilərsə, nəticədə

$$B=\begin{pmatrix}1.00 & 5.00 & 9.00\\ 2.00 & 6.00 & 10.00\\ 3.00 & 7.00 & 11.00\\ 4.00 & 8.00 & 12.00\end{pmatrix}$$

matrisi alınacaqdır.

```
                    1
                  1   1
                1   2   1
              1   3   3   1
            1   4   6   4   1
          1   5  10  10   5   1
        1   6  15  20  15   6   1
      1   7  21  35  35  21   7   1
    1   8  28  56  70  56  28   8   1
  1   9  36  84 126 126  84  36   9   1
```

Şəkil 6.4. Paskal üçbucağı

Misal. Matrislərin vurulması.

```
program matrisin_vurulmasi;
uses crt;
var a: array[1..3,1..4] of real;
    b: array[1..4,1..3] of real;
    c: array[1..3,1..3] of real;
    i,j,k: word;
    s: real;
begin
  for i:=1 to 3 do
    begin
      write(i,'-ci sətir elementlərini daxil edin:');
      for j:=1 to 4 do read(a[i,j]);
    end;
  for i:=1 to 4 do
```

```
    begin
      write(i,'-ci sətir elementlərini daxil edin:');
      for j:=1 to 3 do read(b[i,j]);
    end;
  for k:=1 to 3 do
    begin
      for i:=1 to 3 do
      begin
        s:=0;
        for j:=1 to 4 do
        s:=s+a[i,j]*b[j,k];
        c[i,k]:=s;
      end;
    end;
  for i:=1 to 3 do
    begin
      for j:=1 to 3 do write(c[i,j],'  ');
      writeln;
    end;
  readln
end.
```

Əgər proqrama

A= $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$ və B= $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix}$ matrisləri daxil edilərsə, nəticədə

C= $\begin{pmatrix} 70 & 80 & 90 \\ 158 & 164 & 210 \\ 246 & 288 & 330 \end{pmatrix}$ alınacaqdır.

Matrisin baş və köməkçi diaqonal elementləri. Fərz edək ki,

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n-1} & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n-1} & a_{2n} \\ a_{31} & a_{32} & \dots & a_{3n-1} & a_{3n} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ a_{n1} & a_{n2} & \dots & a_{nn-1} & a_{nn} \end{pmatrix}$$

matrisi verilmişdir. Bu matrisin ixtiyari elementinə `a[i,j]` kimi müraciət etmək olar. Burada i ($i=1,2,\dots,n$) – sətirlərin, j ($j=1,2,\dots,n$) isə sütunların nömrəsidir. a_{11}, a_{22}, $\dots$, a_{n-1n-1}, a_{nn} elementlərinə matrisin *baş diaqonal*

elementləri, a_{1n}, a_{2n-1}, . . . , a_{n-12}, a_{n1} elementlərinə isə *köməkçi diaqonal* elementləri deyilir. Məsələn, yuxarıdakı misalın C matrisində *70, 164, 330* ədədləri matrisin baş diaqonal elementləri, *90, 164, 264* ədədləri isə köməkçi diaqonal elementləridir. Göründüyü kimi, matrisin baş diaqonal elementlərinin indeksləri eynidir. Ona görə də bu elementləri tapmaq üçün dövr yaradıb, dövrün daxilində eyni indekslər yazmaq kifayətdir. Məsələn, *a(n, n)* matrisinin baş diaqonal elementlərini tapmaq üçün proqramda

```
for i:=1 to n do b[i]:=a[i,i];
```

yazmaq kifayətdir. Burada `b[i]` – matrisin baş diaqonal elementlərindən ibarət massivdir.

Matrisin köməkçi diaqonal elementlərini tapmaq üçün *i* və *j* indeksləri arasında asılılıq tapmaq lazımdır. Birinci köməkçi diaqonal elementi a_{1n} elementidir, yəni *i=1* olduqda, *j=n* olur. İkinci $a_{2\,n-1}$ elementi üçün *i=2, j=n–1*, üçüncü $a_{3\,n-2}$ elementi üçün *i=3, j=n–2* və nəhayət *n*–ci a_{n1} elementi üçün *i=n, j=1* olur. Bu asılılığı ümumiləşdirsək, köməkçi diaqonal elementlərinin sətir və sütunları arasında

$i=n-j+1$ və ya $j=n-i+1$

asılılığını alarıq. $i>n-j+1$ olduqda elementlər köməkçi diaqonaldan aşağıda, $i<n-j+1$ olduqda isə elementlər köməkçi diaqonaldan yuxarıda yerləşir. İndi isə bu elementlərin tapılmasına aid misallara baxaq.

Misal. Matrisin baş və köməkçi diaqonal elementlərinin tapılması.

```
program diaqonal_el;
uses crt;
const n=5;
var a:array[1..n,1..n] of integer;
    b,k:array[1..n] of integer;
    i,j:byte;
begin
  for i:=1 to n do
   begin
     write(i,'-ci sətir elementlərini daxil edin:');
     for j:=1 to n do read(a[i,j]);
   end;

   { baş diaqonal elementləri }
  for i:=1 to n do b[i]:=a[i,i];

   { köməkçi diaqonal elementləri }
  for i:=1 to n do
  for j:=n downto n-i+1 do k[i]:=a[i,j];

   { elementlərin çap edilməsi }
  writeln(' baş diaqonal elementləri ');
  for i:=1 to n do write(b[i],'  ');
  writeln;
```

```
 writeln(' köməkçi diaqonal elementləri:');
 for i:=1 to n do write(k[i],'  ');
 readln
end.
```

Daxil edilmiş matris və proqramın nəticəsi şəkil 6.5 – də göstərilmişdir.

```
1-ci setir elementlerini daxil edin:10 20 30 1 2
2-ci setir elementlerini daxil edin:40 50 60 3 4
3-ci setir elementlerini daxil edin:70 80 90 5 6
4-ci setir elementlerini daxil edin:5 6 7 8 9
5-ci setir elementlerini daxil edin:11 22 33 44 55
bash diaqonal elementleri:
10  50  90  8  55
komekchi diaqonal elementleri:
2  3  90  6  11
```

Şəkil 6.5. Matrisin baş və köməkçi diaqonal elementlərinin tapılması

Misal. Matrisin köməkçi diaqonaldan yuxarıdakı və aşağıdakı elementlərinin tapılması.

```
program diaqonal_el;
uses crt;
const n=5;
var a,yu,ya: array[1..n,1..n] of integer;
    b,k: array[1..n] of integer;
    i,j: byte;
begin
 for i:=1 to n do
  begin
    write(i,'-ci sətir elementlərini daxil edin:');
    for j:=1 to n do read(a[i,j]);
  end;

  { köməkçi diaqonaldan yuxarıdakı elementlər }
 for i:=1 to n do
 for j:=1 to n-i do Yu[i,j]:=a[i,j];
 writeln(' köməkçi diaqonaldan
        yuxarıdakı elementlər:');
 for i:=1 to n do
 for j:=1 to n-i do  write(Yu[i,j],'  ');
 writeln;

 { köməkçi diaqonaldan aşağıdakı elementlər }
 for i:=2 to n do
 for j:=n-i+2 to n do Ya[i,j]:=a[i,j];
 writeln('köməkçi diaqonaldan aşağıdakı
        elementlər:');
 for i:=2 to n do
```

```
  for j:=n-i+2 to n do write(Ya[i,j],'  ');
  readln
end.
```

Daxil edilmiş matris və proqramın nəticəsi şəkil 6.6 – da göstərilmişdir.

```
1-ci setir elementlerini daxil edin:10 20 30 1 2
2-ci setir elementlerini daxil edin:40 50 60 3 4
3-ci setir elementlerini daxil edin:70 80 90 5 6
4-ci setir elementlerini daxil edin:5 6 7 8 9
5-ci setir elementlerini daxil edin:11 22 33 44 55
komekchi diaqonaldan yuxaridaki elementler:
10  20  30  1  40  50  60  70  80  5
komekchi diaqonaldan ashaqidaki elementler:
4  5  6  7  8  9  22  33  44  55
```

Şəkil 6.6. Matrisin köməkçi diaqonaldan yuxarıdakı və aşağıdakı elementlərinin tapılması

6.5. Simvol və sətirlər üzərində əməllər

Əvvəlki bölmələrdə biz simvol və sətirlər üzərində yerinə yetirilən əməllərlə, prosedur və funksiyalarla tanış olduq. İndi isə simvol və sətirlərin praktiki tətbiqi ilə məsələlər həll edək.

Misal. *Z* – dən *A* – ya kimi hərflərin ekrana çıxarılması.

```
program Z_A;
uses crt;
var i:char;
begin
  for i:='Z' downto 'A' do
  write(i);
  readln
end.
```

Bu proqram `Z` – dən `A` – ya kimi hərfləri ekranda göstərir.

Misal. Klaviaturadan simvolların daxil edilməsi.

```
Program testread;
uses crt;
var ch:char;
begin
  Writeln('Simvolu daxil et:'); readln(ch);
  ClrScr;
  Writeln(ch,'  simvolu daxil edildi.');
  readln
end.
```

Misal. Simvolların kodlarının tapılması.

```
Program kod;
uses crt;
var ch:char;
begin
 Writeln('Proqram simvolların
        kodlarını tapmaq üçündür');
 Writeln('Proqramdan çıxmaq
        üçün Ctrl+Break klavişlərini bas');
 Writeln;
 Writeln;

  repeat;
    Write('Növbəti klaviş:');
    ch:=ReadKey;
    Writeln('  ord(',ch,')=',Ord(ch));
  Until False;

 readln
end.
```

Bu proqramda `ReadKey` funksiyası tətbiq edilmişdir ki, bu funksiya kursorun yerini dəyişdirmədən simvolu daxil etməyə imkan verir. Simvolu daxil etdikdə, bu funksiya, kursorun yerini dəyişdirmir, ona görə də daxil edilmiş simvolun yerində digər simvol təsvir olunur. Bundan başqa, son şərtli dövr operatorunda `Until False;` konstruksiyası tətbiq edildiyi üçün, proqram sonsuz dövr edəcəkdir. Proqramın işini dayandırmaq üçün ***Ctrl+Break*** klavişlərini birgə basmaq lazımdır.

Misal.

```
Program testread_key;
uses crt;
var ch:char;
begin
 Writeln('Kiçik hərfləri və ya çıxmaq
        üçün z hərfini daxil et:');
  repeat;
    ch:=readKey;
    Write(UpCase(ch));
  Until ch='z';
 readln
end.
```

Bu proqram işləyərkən ekrandan yalnız kiçik hərfləri daxil etmək lazımdır. Proqramın işi kiçik hərfləri baş hərflərə (`UpCase(ch)`) çevirməkdən ibarətdir.

Misal.

```
Program shifreleme;
uses crt;
var ch:char;
begin
 Writeln('Kiçik hərfləri və ya çıxmaq
        üçün Z hərfini daxil et:');
  repeat;
    ch:=ReadKey;
    Write(Char(Ord(ch)+1));
  Until ch='z';
 readln
end.
```

Bu proqramda şifrələmə əməliyyatı aparılır. Klaviaturadan hər hansı bir simvolu daxil etdikdə ekranda tamamilə başqa simvol təsvir olunur (`Write(Char(Ord(ch)+1));`). Bu proqram əsasında adınızı şifrələyən proqramın yazılmasını özünüzə həvalə edirik.

Misal. Daxil edilən simvollar çoxluğunun Azərbaycan dilində ilin ayına uyğunluğunu yoxlamaq. Sadəlik üçün, bu misalda, yalnız böyük hərflərə baxacağıq.

```
program aylar;
uses crt;
const
    Ay: array [1..12] of String [10] =
    ('YANVAR', 'FEVRAL', 'MART', 'APREL', 'MAY',
    'İYUN', 'İYUL', 'AVQUST', 'SENTYABR',
    'OKTYABR', 'NOYABR', 'DEKABR');
var
   Str: string[10];
   i: integer;
begin
 writeln('BÖYÜK HƏRFLƏRLƏ AYIN ADINI DAXİL EDİN');
 Readln(Str);
 for i:= 1 to 12 do
 if Str = Ay[i] then
 writeln(' Ayın adı düzdür ')
 else
 writeln('Bu adda ay yoxdur');
 readln
end.
```

Misal. `Str` və `Val` prosedurlarının tətbiqi.

```
program setir_ve_qiymet;
uses crt;
var i,errcode:integer;
    S:string;
```

```
    a:integer;
begin
  read(A);
  Str(A,S);
  writeln('Sətir qiyməti =',S);
  Val(S,i,errcode);
  if errcode<>0 then
  writeln(errcode,'mövqeyində daxiletmə səhvi')
  else
  Writeln('Ədədi qiymət =',i);
  readln
end.
```

Bu proqramda, tam tipli `A` dəyişəninə ədədi qiymət daxil edilir, lakin o, `Str(A,S);` proseduru ilə `S` sətir tipinə çevrilir. `Val(S,i,errcode);` proseduru ilə isə əks çevirmə yerinə yetirilir: `S` sətir tipi `i` tam tipinə çevrilir.

Misal. "Kalkulyator" proqramı.

```
program kalkulyator;
uses crt;
var emeliyyat:char;
    y:real;
    a,b:real;
begin
  repeat
   writeln('Ədədləri daxil edin');
   write('a=');read(a);
   write('b=');read(b);
   writeln( 'Əməliyyat simvolunu daxil edin',
   'və ya çıxmaq üçün z klavişini basın');
   emeliyyat:=readKey;
     case emeliyyat of
       '+' : y:=a+b;
       '-' : y:=a-b;
       '*' : y:=a*b;
       '/' : y:=a/b;
     else
       exit;
     end;
   writeln(y);
  until emeliyyat='z';
  readln
end.
```

Bu proqramla iki ədəd üzərində sadə hesab əməliyyatları yerinə yetirilir. Proqrama ixtiyari iki ədəd daxil etdikdən sonra, əməliyyat işarəsinin (+ toplama, – çıxma, * vurma və ya / bölmə) daxil edilməsi tələb olunur. Proqramın işi iki halda dayandırılır: `z` klavişini basdıqda və bu əməliyyat işarələrindən fərqli ixtiyari simvol daxil etdikdə `exit` proseduru ilə.

Yeddinci fəsil

ALT PROQRAMLAR

Bu fəsildə alt proqramlar haqqında ümumi məlumatlar veriləcək, lokal və qlobal dəyişənlər, formal və faktik parametrlər araşdırılacaqdır. Prosedur, funksiya və rekursiv tipli alt proqramların strukturunu və əsas proqramdan onlara müraciət qaydalarını öyrənəcəyik. Modulların strukturu ilə tanış olacaq və modulların yaradılması qaydalarını mənimsəyəcəyik. Alt proqramların tətbiqi ilə məsələlər həll ediləcəkdir.

7.1. Alt proqramlar haqqında ümumi məlumatlar

Proqramların tərtibində yaxşı proqramlaşdırma üslubu qaydalarına əməl etmək vacibdir. Yeri gəlmişkən bu qaydaların bir neçəsini xatırlayaq:

- `Program` sözündə (sərlövhədə) müəyyən məna bildirən yığcam proqram adları yazmaq;
- Sabit və dəyişənlərin adlarında müəyyən məna bildirən identifikatorlar istifadə etmək;
- Proqram bloklarının, dəyişən və sabitlərin mahiyyətini dərk etmək üçün qısa şərhlər yazmaq;
- Abzas və boş sətirlər əlavə etməklə proqramı strukturlaşdırmaq.

Bu və digər qaydaların tətbiqi asan başa düşülən proqramlar tərtib etməyə imkan verir.

Ən yaxşı proqramlaşdırma üslublarından biri də prosedur və funksiya tipli alt proqramların əsas proqramlara əlavə edilməsidir.

Alt proqram məntiqi bitkin və xüsusi qayda ilə tərtib edilmiş operatorlar qrupundan ibarətdir. Proqramın müxtəlif hissələrindən alt proqrama dəfələrlə müraciət etmək olar.

Strukturuna görə alt proqram əsas proqrama demək olar ki, tam analojidir. Alt proqram da sərlövhə və blokdan ibarətdir, lakin *alt proqramda modulların qoşulması bölməsi (*`Uses`*) olmur.*

Alt proqramla iş iki əsas mərhələdən ibarətdir:

- alt proqramın təsviri;
- alt proqramın çağrılması.

İstənilən alt proqram əvvəlcə təsvir olunmalıdır. Təsviretmə zamanı alt proqramın adı, parametrlərin siyahısı və onun yerinə yetirəcəyi əməliyyatları icra edən operatorlar ardıcıllığı yazılır. Alt proqramı çağırdıqda isə yalnız onun adı və alt proqrama ötürüləcək faktik parametrlərin siyahısı göstərilir.

Turbo Pascal dilinin müxtəlif modullarında çoxlu standart alt proqramlar vardır ki, onlara müraciət etmək üçün onları təsvir etməyə ehtiyac yoxdur. Belə alt proqramların bir neçəsi ilə biz ifadələr bölməsində tanış olduq. Bundan başqa, proqramçı özü də alt proqram yarada bilər ki, ona istifadəçi alt proqramı deyilir. İstifadəçi alt proqramları isə hökmən təsvir olunmalıdır.

Alt proqramlar *prosedur* və *funksiyalara* bölünür. Bunların bir sıra ümumi cəhətləri ilə yanaşı, fərqli xüsusiyyətləri də vardır. Bu fərqlər əsasən aşağıdakılardır:

- funksiya əsas proqrama bir qayda olaraq yalnız bir qiymət ötürür, məsələn, `Sin(x)`. Prosedur isə alt proqrama müxtəlif qiymətlər, məsələn, massiv ötürə bilər;
- funksiyanın sərlövhəsində onun proqrama ötürəcəyi qiymətin tipi (`real`, `integer` və s.) göstərilir. Məsələn, `function Sin(x:real):real;` Prosedurun sərlövhəsində isə buna ehtiyac yoxdur;
- funksiyanın gövdəsində ən azı bir mənimsətmə operatoru olmalıdır ki, onun sol tərəfində funksiyanın adı yazılmalıdır, yəni funksiyanın adına hökmən qiymət mənimsədilməlidir;
- funksiyaya müraciət etdikdə, onu ifadələrdə identifikator kimi istifadə etmək mümkün olduğu halda, prosedur ifadənin bir hissəsi ola bilməz.

Proqramla alt proqram arasında mübadilə dəyişənlər vasitəsilə yerinə yetirilir. Dəyişənlər qlobal və lokal dəyişənlərə bölünür. Əgər sabit və dəyişənlər əsas proqramda elan olunmuşdursa, belə dəyişənlərə *qlobal dəyişənlər* deyilir. Bu dəyişənlərə kompilyasiya mərhələsində, verilənlər seqmentində yaddaşda (stekdə) yer ayrıldığı üçün, həmin dəyişənləri proqramın istənilən hissəsində, o cümlədən, alt proqramlarda da istifadə etmək olar (bu halda deyirlər ki, həmin dəyişənlər proqramın hər yerindən görünür).

Əgər sabit və dəyişənlər alt proqramda elan olunmuşdursa, onlara *lokal dəyişənlər* deyilir. Qlobal dəyişənlər alt proqramda təsvir olunduqda onlar lokal dəyişənlər olur və bu dəyişənlər yalnız onların elan olunduqları alt proqramlardan görünür. Lokal dəyişənlər üçün də yaddaşda – verilənlər

seqmentində – yer ayrılır. Qlobal və lokal dəyişənlərin adları eyni olsa belə, onlar arasında heç bir əlaqə olmur. Dərhal qeyd edək ki, çalışın alt proqramlarda qlobal dəyişənlərdən istifadə etməyəsiniz. Bunun bir neçə səbəbi vardır. Səbəblərdən biri odur ki, qlobal dəyişənləri istifadə edən alt proqram az universal olur və onu digər proqramlarda istifadə etmək çətinləşir. İkinci səbəb isə belə alt proqramlardan istifadə etdikdə, çətin aşkar olunan səhvlərin artması ehtimalı ilə bağlıdır.

Əsas proqramla alt proqramın qarşılıqlı əlaqəsi, adətən, parametrlər vasitəsilə yerinə yetirilir. *Parametrlər* (*formal* parametrlər) alt proqramın elementləridir və alt proqramın yerinə yetirdiyi alqoritmin təsvirində istifadə edilir.

Arqumentlər (*faktik* parametrlər) alt proqramı çağıran proqramın elementidir. Alt proqrama müraciət etdikdə formal parametrlər faktik parametrlərlə əvəz olunur. Bu zaman formal və faktik parametrlər tipləri, yerləşmə ardıcıllığı və saylarına görə bir–birinə uyğun gəlməlidir. Formal və faktik parametrlər nəinki, müxtəlif adlı ola bilər, hətta çalışmaq lazımdır ki, bu elə belə də olsun.

Bəs prosedurlar necə çağrılır və onlara bu parametrlər necə ötürülür? Çox sadə. Əvvəlcə bu parametrlər stekə qəbul olunur. *Stek* müvəqqəti və ya lokal dəyişənlərin yadda saxlanması üçün yaddaş oblastıdır. Tutaq ki, prosedurun iki parametri vardır. Əvvəlcə birinci parametr, sonra isə ikinci parametr stekə ötürülür və prosedur çağrılır. Yerinə yetirilməyə başlamazdan əvvəl, prosedur həmin parametrləri stekdən əks istiqamətdə çıxarır.

Əgər alt proqramda digər prosedur və funksiyalar varsa, onda dəyişənlərin görünmə oblastı həmin prosedur və funksiyalara da aid olacaq, bu şərtlə ki, onlar eyniadlı olsun.

Alt proqramın işini dayandırmaq üçün `Exit` prosedurundan istifadə etmək olar ki, bu prosedur idarəetməni əsas proqrama verir.

İstənilən növ alt proqramları nəinki əsas proqramdan, həm də digər proqramlardan, parametrlərin eyni və ya müxtəlif qiymətlərində, istənilən qədər çağırmaq olar. Alt proqramları əsas proqramlardan kənarda da kompilyasiya etmək olar.

İndi isə, alt proqramlar haqqında bu vacib, ümumi bilikləri öyrəndikdən sonra, prosedur və funksiyaları öyrənək.

7.2. Prosedurlar

Prosedur sərlövhə və proqram blokundan fərqlənməyən blokdan ibarət olur. Sərlövhə **`procedure`** sözündən, prosedurun adından və onun yanında mötərizə daxilində yazılmış formal parametrlərin siyahısından ibarət olur. Prosedurun ümumi forması belədir:

`Procedure` *ad (formal parametrlər)***;**
  *lokal parametr, prosedur və funksiyaların təsviri və təyini bölmələri***;**

`begin`

1–ci operator;
...
n–ci operator;
end;

Formal parametrlərin qarşısında `Var`, `Const` sözləri yazıla bilər. Bunların mahiyyətini bir az sonra izah edəcəyik. Formal parametrlərin adından sonra, iki nöqtə işarəsi ilə ayrılmaqla, onun tipi göstərilə bilər. Eyni tipli parametrlər olarsa, onlar arasında vergül işarəsi qoyulmalıdır. Formal parametrlər olmaya da bilər.

Misal.

```
procedure format(hour,min,sec,hund : word);
procedure ChangeStr(Var Source : string;
                    const char1,char2 : char);
procedure zaman;
```

Prosedura müraciət üçün prosedurun adını və mötərizə daxilində faktik parametrləri göstərmək lazımdır.

İndi isə prosedura aid misallara baxaq.

Misal. $y = \sqrt{n!} + (m!)^2 + (n-m)!$ hesablayın.

Göründüyü kimi, üç müxtəlif dəyişənin faktorialını hesablamaq lazımdır. Ona görə də biz *n!* hesablanması üçün alt proqram tərtib edərək sonrakı hallarda ona müraciət edəcəyik.

```
Program fact_sum;
uses crt;
Var
  y1,y2,y3:integer;
  n,m:integer;
  y,t1,t2:real;

{ Alt proqram }

procedure fact(l:integer; Var p:integer);
Var
  i:integer;
begin
  p:=1;
  if l<0 then
  begin
    p:=1;
    Exit;
  end;
for i:=1 to l do p:=p*i;
end;
```

{ *Əsas proqram* }

```
begin
  readln(n,m);
  fact(n,y1); t1:=sqrt(y1);
  fact(m,y2); t2:=sqr(y2);
  fact(n-m,y3);
  y:=t1+t2+y3;
  writeln(y);
  readln
end.
```

Alt proqramda *n!* hesablanması üçün proqram yazılmışdır. Burada `l` və `p` alt proqramın formal parametrləridir. `l` – faktorialı hesablanacaq parametr, `p` isə alt proqramın hesablayacağı nəticədir. Məlumdur ki, mənfi ədədin faktorialı mövcud deyil, ona görə də `l<0` olduqda, faktorial vahidə bərabər götürülür (`p:=1;`) və idarəetmə əsas proqrama ötürülür (`Exit;` proseduru), əks halda `l` dəyişəninin faktorialı hesablanır `(for i:=1 to l do p:=p*i;)`. Əsas proqramda isə alt proqrama üç dəfə müraciət olunmuşdur. Hər dəfə müraciət zamanı əsas proqramdan alt proqrama, faktik parametrlər kimi, uyğun olaraq `n`, `m` və `n-m` dəyişənləri ötürülür, alt proqramdan isə əsas proqrama, nəticə kimi, uyğun olaraq `y1`, `y2` və `y3` dəyişənləri qaytarılır.

Misal. $y = [thx + th(x+a)^2] \cdot \sqrt{th(x-a)^2}$ hesablayın.

Hiperbolik tangensi hesablamaq üçün aşağıdakı düsturu tətbiq edək:

$$th\,x = \frac{e^{2x}-1}{e^{2x}+1}.$$

```
program Hyp_tang;
uses crt;
var x,a,y1,y2,y,r,q : real;

{ Alt proqram }
Procedure htan;
var
   s : real;
begin
  s:= exp(2*q);
  r:= (s-1)/(s+1);
end;

 { Əsas proqram }
begin
  read(x,a);
  q:=x;
  htan;
  y1:=r;
```

```
  q:=sqr(x+a);
  htan;
  y2:=r;
  q:=sqr(x-a);
  htan;
  y:=(y1+y2)*sqrt(r);
  writeln('y=',y);
  readln
end.
```

Bu proqramda parametrsiz `htan` prosedur tipli alt proqramından istifadə edilmişdir. Bu alt proqram yalnız hiperbolik tangens funksiyasını hesablayır. Əsas proqramda isə ona *3* dəfə müraciət olunmuşdur: birinci dəfə faktik parametr kimi bu prosedura *x*, ikinci dəfə $(x+a)^2$, üçüncü dəfə isə $(x-a)^2$ arqumentləri ötürülür. Hər müraciətdən sonra isə alt proqramın nəticələri uyğun olaraq `y1`, `y2` və `sqrt(r)` kimi əsas proqrama ötürülür.

7.3. Funksiyalar

Funksiyalar da sərlövhə və blokdan ibarətdir.

Sərlövhə – **`function`** sözündən, funksiyanın adından və onun yanında, mötərizə daxilində yazılmış, vacib olmayan, formal parametrlərin siyahısından və onların tiplərindən ibarət olur. Sərlövhədə funksiyanın özünün də tipi göstərilir.

Funksiyanın *bloku* prosedurun blokuna analojidir, lakin, funksiyanın gövdəsində onun adına hökmən qiymət (ifadə) mənimsədilməlidir. Funksiyanın gövdəsi `begin` sözü ilə başlayır və `end;` ilə qurtarır. Beləliklə, funksiyaların ümumi forması belədir:

`Function F` $(q_1 : t_1;\ q_2 : t_2;\ \ldots ; q_n : t_n) : T$**;**

lokal parametr və alt proqramların təsviri və təyini;

`begin`

1–ci operator;
`...`
n–ci operator;
`F :=` *ifadə*;

`end;`

Burada, **`F`** – funksiyanın adı, q_i – formal parametrlərin adları, t_i – formal parametrlərin tipləri, *T* isə funksiyanın özünün tipidir.

Misal.

```
function Sin(x : real) : real;
function Pi : real;
function Random(x : word) : word;
```

Funksiyaya müraciət, mənimsətmə operatorunun sağ tərəfində funksiyanın adını və mötərizə daxilində faktik parametrləri göstərməklə həyata keçirilir, məsələn, `Sin(y)`, `Pi`, `Random(a)` və s. Faktik və formal parametrlər saylarına, tiplərinə və yerləşmə ardıcıllığına görə uzlaşmalıdır. Funksiyaya müraciət olunduqda faktik parametrlərin qiyməti uyğun mövqedə duran formal parametrlərə mənimsədilir. Sonra isə funksiyanın gövdəsini təşkil edən operatorlar yerinə yetirilir və son nəticə funksiyanın adına mənimsədilir. Əgər belə mənimsətmə bir neçə dəfə olarsa, onda ən sonuncu qiymət əsas proqrama ötürülür.

Misal. $y = \sqrt{n!} + (m!)^2 + (n-m)!$ hesablayın.

Prosedur bölməsində həll etdiyimiz bu məsələni funksiya alt proqramı ilə həll edək.

```
Program Fact_sum;
Uses crt;
var y:real;
    n,m:integer;

{ function tipli alt proqram }
function fact(l:integer):integer;
var p,i:integer;
begin
 p:=1;
 if l<=0 then
 begin
   fact:=1;
   Exit;
 end;
 for i:=1 to l do p:=p*i;
 fact:=p;
end;

{ Əsas proqram }
begin
 readln(n,m);
 y:=sqrt(fact(n))+sqr(fact(m))+fact(n-m);
 write(y);
 readln
end.
```

Burada funksiya tipli `fact` alt proqramının formal parametri faktorialı hesablanacaq `l` arqumentidir. Prosedur tipli alt proqramda olduğu kimi, burada da `l` arqumentinin işarəsi yoxlanır və əgər o mənfidirsə, onda `fact:=1` qəbul olunur və alt proqram öz işini dayandırır (`Exit;` proseduru). `l` arqumentinin qiyməti müsbət olduqda isə alt proqramda onun faktorialı hesablanaraq funksiyanın adına mənimsədilir (`fact:=p;`). Əsas proqramda, `y`–in hesablanması düsturunda, birbaşa `fact` funksiyasına müraciət olunur və hər

müraciət zamanı bu funksiyaya faktik parametrlər kimi, `n`, `m` və `n−m` qiymətləri ötürülür.

Misal. *F(x)=x/(1+x)* funksiyasını cədvəlləşdirin.

Biz bu məsələnin proqramını əvvəlki bölmələrdə tərtib etmişdik. İndi bu məsələni alt proqram vasitəsilə həll edək.

```
program cedvel;
uses crt;
var x:real;
    k:word;

{ function tipli alt proqram }
function F(x:real):real;
begin
  F:=x/(1.0+x);
end;

{ Əsas proqram }
begin
  clrscr;
  x:=0.0;
  writeln('f(x)=x/(1+x)
          funksiyasının qiymətlər cədvəli');
  writeln;
  Writeln('sira N','x':10,'f(x)':20);
  writeln;
  for k:=0 to 50 do
    begin
      writeln(' ',k,' ',x:12:4, F(x):20:10);
      x:=x+0.1;
      if k mod 10=9 then readln;
    end;
  readln
end.
```

Misal. $y=\sum_{i=1}^{10}a_i+\sum_{i-1}^{15}a_i\cdot b_i+\sum_{i=1}^{20}(a_i-b_i)^2$ hesablayın.

```
program Alt_proq;
uses crt;
Type mas=array[1..20] of real;
var a,b,c,c1:mas;
    i:integer;
    y:real;

 { function tipli alt proqram }
function cem(k:integer;z:mas):real;
var y:integer;
```

```
    s:real;
begin
  s:=0.0;
  for y:=1 to k do
  s:=s+z[y];
  cem:=s;
end;

{ Əsas proqram }
begin
  for i:=1 to 20 do read(a[i]);
  for i:=1 to 20 do read(b[i]);
  for i:=1 to 15 do c1[i]:=a[i]*b[i];
  for i:=1 to 20 do c[i]:=sqr(a[i] -b[i]);
  y:=cem(10,a)+cem(15,c1)+cem(20,c);
  write(' ':10,'y=',y:10:2);
  readln
end.
```

7.4. Rekursiv alt proqramlar

Əgər alt proqram özü özünü çağırırsa, belə proqramlara *rekursiv* alt proqramlar deyilir.

Misal. *y=n!* hesablamaq üçün rekursiv alt proqram yazaq.

```
function fact(n:integer):integer;
begin
  if n<=0 then fact:=1 else fact:=n*fact(n-1);
end;
```

`fact` funksiyası `n` qiymətini alaraq onun faktorialını hesablayır. Əgər `n<=0` olarsa, onda faktorial vahidə bərabər olur. Əks halda, `n`–in qiymətini bir vahid azaldaraq (`fact(n-1)`) özü özünə müraciət edir. Bu proses `n=1` olana qədər davam edir.

7.5. Alt proqramlarda parametr və arqumentlər

Parametrlər alt proqramın elementləridir və onun alqoritminin təsvirində istifadə olunur. Arqumentlər isə alt proqram çağrıldıqda göstərilir və proqram icra olunduqda onlar parametrləri əvəz edirlər. Parametrlər istənilən tip ola bilər. Parametrlərin aşağıdakı tipləri mövcuddur:

- *parametr–qiymətlər;*
- *parametr–sabitlər;*
- *parametr dəyişənlər;*
- *tipləşdirilməmiş sabit və dəyişənlər.*

Əgər alt proqramın sərlövhəsində, formal parametrlərin adları qarşısında, hər hansı bir işçi söz (`Var`, `Const`) yazılmazsa, onda həmin formal parametrlər *parametr–qiymətlər* adlanır.

Misal.

```
procedure par_qiy(x : real; y : integer; s : string);
function vaxt(first, second : word) : boolean;
```

Parametr–qiymət kimi dəyişən, sabit və ya riyazi ifadə istifadə oluna bilər. Alt proqram çağrılmazdan əvvəl riyazi ifadə hesablanır, alınmış nəticə yaddaşa köçürülür və yalnız bundan sonra alt proqrama ötürülür. Eyni proses parametr–qiymət sabit və ya dəyişən olduqda da baş verir. Beləliklə, faktik parametrin qiyməti eyni vaxtda iki yerdə yadda saxlanır: verilənlər seqmentində – qiymətin əsli və stekdə onun surəti. Alt proqramda parametr–qiymət dəyişə bilər, lakin bu dəyişikliklər alt proqramlar yerinə yetirildikdə faktik parametrlərin qiymətlərinə təsir etməyəcəkdir, başqa sözlə, əsas proqrama ötürülməyəcəkdir.

Proqramın sərlövhəsində formal parametrlərin qarşısında `Const` sözü yazılarsa, həmin formal parametr *parametr–sabit* adlanır.

Misal.

```
Procedure sabit(Const x, y : integer);
```

Parametr–sabit kimi dəyişən, sabit və riyazi ifadə istifadə oluna bilər. Alt proqramın gövdəsində parametr–sabiti dəyişdirmək olmaz. Hansı parametrlərin ki, alt proqramda dəyişdirilməsi arzu olunmazdır, həmin parametrləri parametr–sabit kimi göstərmək lazımdır. Bundan başqa, sətir və struktur tipli parametr–sabitlər üçün kompilyator daha səmərəli kod yaradır.

Əgər alt proqramın sərlövhəsində, formal parametrlərin qarşısında, `Var` sözü yazılarsa, onda bu parametr *parametr–dəyişən* olacaqdır. Bu parametrə *Var–parametr* də deyirlər.

Misal.

```
Procedure takt(Var param : real; Var d,f : integer);
```

Parametr–dəyişən o hallarda istifadə olunur ki, alt proqramdan əsas proqrama qiymət ötürülməlidir. Parametr–dəyişən tətbiq olunduqda faktik qiymət kimi riyazi ifadə istifadə etmək olmaz. Parametr–dəyişənlərin tətbiqinin üstün cəhəti ondadır ki, böyük ölçülü parametrlərin ötürülməsi zamanı proqram sürətlə işləyir və parametrin nəticəsi, avtomatik olaraq, əsas proqrama ötürülür. Bununla bərabər, bütün formal parametrlərin parametr–dəyişənlər kimi elan olunması məqsədəuyğun hesab olunmur. Çünki, bu əvvəla, faktik parametrlərə riyazi ifadələrin verilməsinə imkan vermir, digər tərəfdən formal parametrin alt proqramda qiyməti təsadüfən itirilə bilər. Parametrlər nə qədər az olarsa, qlobal dəyişənlər də bir o qədər az olar ki, bu da proqramın sadə və asan başa düşülən olmasına gətirər. Elə bu səbəbdən də funksiyalarda parametr– dəyişənlərin istifadə edilməsi məsləhət görülmür: çalışmaq lazımdır ki, funksiyanın nəticəsi yalnız bir qiymət olsun.

Alt proqramların parametrləri *tipləşdirilməmiş sabit və dəyişənlər* də ola bilər. Bu halda alt proqramın sərlövhəsində parametr–sabit və parametr dəyişənlərin tipləri göstərilmir.

Misal.

```
Procedure tipsiz(Var f1; Const t1);
Function tip (Var c2):real;
```

Formal parametrlərə verilən faktik parametrlər istənilən tip ola bilər və bunu proqramçı özü müstəqil müəyyənləşdirir.

7.6. Modullar

Modul anlayışı ilk dəfə *Ada* proqramlaşdırma dilinə daxil edilərək, paket adlanırdı. Niklaus Virt tərəfindən yaradılmış *Pascal* dilinin ilk versiyalarında modul olmamışdır. Amma bir qədər sonra, *Ada* dilində abstrakt tiplərin və paketlərin inkişafı ilə əlaqədar olaraq, Turbo Pascal dilinə modul daxil edilmişdir.

Modul informasiyanın gizlədilməsi ("*information hiding*") prinsipini əsas götürərək, proqramların yaradılmasında istifadə olunur. Turbo Pascal dilində modullar prosedur, funksiya və obyekt kitabxanalarının yaradılmasında istifadə olunur. Modulun köməyi ilə böyük proqramlar nisbətən kiçik proqram fraqmentlərinə parçalanır.

Modullar da proqramlar kimi kompilyasiya olunur, lakin onlardan fərqli olaraq sərbəst icra oluna bilmir.

Modullar iki qrupa bölünür:

- *standart modullar;*
- *istifadəçi modulları.*

Standart modullar Turbo Pascal dilində əvvəlcədən hazırlanmış modullardır. Bu modullardan proqramlarda kompilyasiya olunmuş halda istifadə olunur. Kitabxanada aşağıdakı standart modullar mövcuddur:

- **`System`** –əsas kitabxana;
- **`String`** –ASCIIZ –sətirlərinin emalı üçün kitabxana;
- **`Crt`** –ekran ilə işləmək üçün kitabxana;
- **`Graph`** –qrafiki kitabxana;
- **`Dos`** –*Ms DOS* sisteminin imkanlarından istifadə üçün kitabxana;
- **`WinDos`** –ASCIIZ–sətirlərini nəzərə almaqla *Ms DOS* sisteminin imkanlarından istifadə üçün kitabxana;
- **`Overlay`** –overley strukturun təşkili;
- **`Printer`** –printerlə iş;
- **`Turbo3`** –Turbo Pascal 3.0 proqramları ilə əlaqə;
- **`Grap3`** –Turbo Pascal 3.0 qrafikləri ilə əlaqə;

CRT (*Cathod radio tube – elektron–şüa borusu*) modulu ekranı mətn rejimində idarə edən alt proqramlardan ibarətdir. Modul displeyin *8* iş rejimi sabitlərindən, *17* rəng sabitlərindən, *4* funksiya və *16* prosedurdan ibarətdir. Məsələn, `ClrScr`, `GotoXY`, `Delay`, `ReadKey`, `Keypressed` və s. prosedurlar bu modulun alt proqramlarıdır. Bu modulun qoşulması bütün giriş–çıxış əməliyyatlarını sürətləndirir. Ona görə də, hətta bu modulun alt proqramlarını istifadə etmədikdə də, onun qoşulması məsləhət görülür.

Qeyd. Pascal dilinin **Turbo Pascal for Windows 1.5** versiyası ilə işlədikdə **CRT** əvəzinə **WinCrt** yazmaq lazımdır.

İstifadəçi modulları proqramçı tərəfindən yaradılan modullardır. Bu modullar kompilyasiya olunub, sazlandıqdan sonra proqramlarda istifadə edilə bilər.

Modullar aşağıdakı hissələrdən ibarətdir:

- *modulun sərlövhəsi*;
- *modulun interfeysi*;
- *reallaşdırma bölməsi*;
- *inisiallaşdırma bölməsi.*

Modulun sərlövhəsi. Modulun sərlövhəsi **unit** işçi sözündən və modulun adından ibarətdir.

Misal. **unit** `Modul2.`

Modullar da adi pascal–proqramları kimi inteqrallaşdırılmış mühitdə yığılır. Modulu yadda saxladıqda isə fayla verilən ad hökmən modulun adı ilə (`unit` sözündən sonrakı adla) eyni olmalıdır və bu fayl da *.pas* tipli olacaqdır.

Modulun interfeysi. Bu bölmədə modulun digər istifadəçi və standart modullarla, həmçinin əsas proqramla qarşılıqlı əlaqəsi təsvir olunur. Başqa sözlə, modulun "xarici aləmlə" qarşılıqlı əlaqəsidir. Modulun interfeysi **interface** sözü ilə başlayır və aşağıdakı hissələrdən ibarət ola bilər:

- *istifadə olunan modulların təsviri bölməsi*;
- *sabitlərin təsviri bölməsi*;
- *tiplərin təsviri bölməsi*;
- *dəyişənlərin təsviri bölməsi*;
- *prosedur və funksiyaların təsviri bölməsi.*

Reallaşdırma bölməsi (icraedici hissə). Bu bölmədə cari modulun reallaşdırılması təsvir olunur, başqa sözlə, modulun "daxili mətbəxidir" və digər modul və proqramların buradan istifadəsi mümkün deyildir. Reallaşdırma bölməsi **implementation** sözü ilə başlayır və inisiallaşdırma bölməsinin başlanğıcı və ya **end.** sözü ilə qurtarır.

Bu bölmə aşağıdakı hissələrdən ibarət ola bilər:

- *nişanların təsviri bölməsi*;
- *istifadə olunan modulların təsviri bölməsi*;
- *sabitlərin təsviri bölməsi*;
- *tiplərin təsviri bölməsi*;
- *dəyişənlərin təsviri bölməsi*;
- *prosedur və funksiyaların təsviri bölməsi.*

İnisiallaşdırma bölməsi. Bir çox hallarda, modula müraciətdən əvvəl, onun inisiallaşdırılması tələb olunur, məsələn, `Assign` prosedurunun köməyi ilə bəzi fayllarla əlaqə yaratmaq, hər hansı dəyişənin inisiallaşdırılması və s. həyata keçirilməlidir. Bu əməliyyatları inisiallaşdırma bölməsi həyata keçirir. Bölmə `begin` və `end` sözləri arasındakı icra olunan operatorlardan təşkil olunur. İnisiallaşdırma operatorları tələb olunmursa, `begin` sözü yazılmır.

Modulda əksər hallarda inisiallaşdırma bölməsi olmur, adətən bu bölmədə hər hansı faylla əlaqə yaratmaq üçün kodlar yazılır. Əgər bu bölmə olarsa, o, `begin` və `end.` operatorları arasında yerləşdirilir.

Modulun interfeysində təsvir olunan sabit, dəyişən, prosedur və funksiyalardan əsas proqramda istifadə etmək üçün `uses` işçi sözündən istifadə olunur. Bu təsvirdən sonra, əsas proqramda interfeysdə göstərilən modullardan istifadə etmək mümkündür.

Modul köməkçi obyekt olduğu üçün, onu *Run* (***Ctrl+F9***) əmri ilə icra etmək olmaz. O, yalnız proqram və alt proqramların yaradılmasında iştirak edə bilər.

Modul iki mərhələ üzrə yaradılır. Birinci mərhələdə ayrı bir faylda modulun mətni yığılmalıdır. Bu faylın adı modulun adındakı birinci *8* simvolla eyni olmaqla, *.pas* tipinə malik olmalıdır. Modul və proqram hər ikisi eyni bir qovluqda yerləşməlidir.

Hər bir modul **`TPU`** (*Turbo Pascal Unit*) adlanan xüsusi kitabxanada yerləşməlidir. Modulu `TPU` kitabxanasına yerləşdirmək üçün onu ***F9*** və ya ***Ctrl+F9*** əmri ilə kompilyasiya etmək lazımdır. Bundan sonra, modulun adından ibarət *.pas* tipli fayl yaranacaqdır. Bu faylın *.pas* tipli adi proqram faylından fərqi ondadır ki, burada informasiya maşın kodları ilə təsvir olunduğu üçün, onu oxumaq və başa düşmək mümkün olmur. Sonradan düzəlişlər etmək məqsədilə *.pas* tipli faylı pozmaq məsləhət görülmür.

Misal. Hiperbolik funksiyaların hesablanması üçün modul yaradaq. Bu funksiyaların düsturlarını yada salaq:

$$\sinh(x)=\frac{e^{x}-e^{-x}}{2};\quad \cosh(x)=\frac{e^{x}+e^{-x}}{2};\quad \tanh(x)=\frac{\sinh(x)}{\cosh(x)}.$$

```
{$N+}
Unit hyp_fun;

interface
```

```
type
     deqiq=Extended;
     function sinh(x:deqiq):deqiq;
     function cosh(x:deqiq):deqiq;
     function tanh(x:deqiq):deqiq;

implementation

var t:deqiq;

function sinh(x:deqiq):deqiq;
begin
   t:=exp(x);
   sinh:=0.5*(t -1./t);
end;

function cosh(x:deqiq):deqiq;
begin
  t:=exp(x);
  cosh:=0.5*(t+1./t);
end;

function tanh(x:deqiq):deqiq;
begin
  t:=exp(2.0*x);
  tanh:=(t -1.0)/(t+1.0);
end;

end.
```

Bu modulu ***F9*** və ya ***Ctrl+F9*** əmri ilə kompilyasiya etdikdən sonra, onu `hyp_fun.pas` adı ilə yadda saxlayın. İndi bu moduldan əsas proqramda istifadə edək.

```
{$N+}
program test_hyperbolic_fun;
Uses CRT,hyp_fun;   {hyp_fun hökmən yazılmalıdır }
begin
  clrScr;
  Writeln('Sinh(0.5)=',sinh(0.5));
  Writeln('Cosh(-0.5)=',cosh(0.5));
  Writeln('Tanh(0.5)=',tanh(1.5));
  readln
end.
```

Bu misalda, alt proqramlara və modullara heç bir aidiyyəti olmayan **`{$N+}`** direktivindən istifadə olunmuşdur. `{$N+}` direktivi ona görə yazılmışdır ki, burada yüksək dəqiqlikli `Extended` tipindən istifadə edilmişdir (əks halda kompilyator həmin tipi tanımır).

Səkkizinci fəsil

FAYLLAR

Bu fəsildə proqramın nəticələrinin faylda yadda saxlanılması və ilkin verilənlərin fayllardan oxunması qaydalarını öyrənəcəyik. Mətn faylları, tipləşdirilmiş fayllar və tipləşdirilməmiş fayllar ayrı–ayrılıqda öyrəniləcək, onların müqayisəli təhlili veriləcək, bu fayllarla işləmək üçün zəruri funksiya və prosedurlar nəzərdən keçiriləcəkdir. Qovluq və fayllarla iş üçün ümumi vasitələr şərh olunacaq və fəslin sonunda fayllarla işləmək üçün çoxlu praktiki məsələlər həll ediləcəkdir.

8.1. Fayllar haqqında ümumi məlumatlar

İndiyədək yazdığımız proqramlarda ilkin verilənləri həmişə klaviaturadan daxil edirdik və proqramdan çıxan kimi onun nəticələri ekrandan itirdi. Fayllardan istifadə etdikdə biz, proqrama ilkin verilənləri klaviaturadan deyil, fayllardan daxil edə bilərik və nəticələri fayllarda saxlaya bilərik. Bu isə imkan verir ki, alınmış nəticələri sonradan təhlil edək və ya onları digər proqramlarda, verilənlər bazalarında istifadə edək. Bildiyimiz kimi, fayl xarici yaddaşda adlandırılmış hər hansı bir sahədir. Bu faylın fiziki mövcudluğudur ki, bu da fiziki fayl adlanır. Fiziki faylın strukturu informasiya daşıyıcılarında (disk və ya disket) ardıcıl baytlardan ibarətdir, yəni aşağıdakı kimidir:

bayt	*bayt*	*bayt*	. . .	*bayt*	*bayt*

Digər tərəfdən, fayl proqramlaşdırmada istifadə olunan çoxlu sayda verilənlər strukturundan biridir. Bu halda *məntiqi fayl* anlayışından istifadə olunur. Yəni proqram tərtibində faylın mövcudluğu haqqında bizdə yalnız

məntiqi təsəvvür var. Proqramlarda məntiqi fayllar *fayl dəyişəni* ilə təsvir olunur.

Məntiqi faylın strukturu proqramda faylın başa düşülmə üsuludur. Daha dəqiq desək, faylın fiziki strukturuna baxmaq üçün olan "şablon" və ya "pəncərə"dir. Proqramlaşdırma dilində belə "şablonlara" verilənlərin tipi uyğundur. Turbo Pascal dilində bir neçə "şablonun" təsviri aşağıdakı göstərilmişdir:

`file of Byte`

bayt	*bayt*	*bayt*	. . .	*bayt*	**Eof**

`file of Char`

simvolun kodu	*simvolun kodu*	*simvolun kodu*	. . .	*simvolun kodu*	**Eof**

`file of Integer`

işarəli tam	*işarəli tam*	*işarəli tam*	. . .	*işarəli tam*	**Eof**

`file of P`

bayt	*simvolun kodu*	*işarəli tam*	. . .	*bayt*	*simvolun kodu*	*işarəli tam*	**Eof**

Sonuncu təsvirdə `P` yazı tipli dəyişəndir və onun strukturu belədir:

```
type
   P=record
     a:Byte;
     b:Char;
     c:Integer;
   end;
```

Məntiqi faylın strukturu massivə oxşayır. Amma, fayl və massivlərin bəzi fərqləri mövcuddur. Massivlər operativ yaddaşda təşkil olunur və onların elementlərinin sayı adətən əvvəlcədən məlum olur və nömrələnir. Faylda isə proqramın işi zamanı elementlərin sayı dəyişir və o, xarici informasiya daşıyıcılarında yerləşir. Faylın elementlərinin sayı hər an dəyişə bilər. Amma, faylın sonuna **ASCII** kodu *26* (***Ctrl+Z***) olan xüsusi **`Eof`** simvolu əlavə edilir. Bunlardan başqa, faylın uzunluğunu müəyyən etmək və ya fayllar üzərində digər əməliyyatları yerinə yetirmək üçün, xüsusi standart prosedur və funksiyalardan istifadə olunur.

Üçüncü fəsildə qeyd etdiyimiz kimi, Turbo Pascal dilində *3* növ fayl vardır:

- *Mətn faylları*;
- *Tipləşdirilmiş fayllar*;
- *Tipləşdirilməmiş fayllar.*

Qeyd etmək lazımdır ki, tipləşdirilmiş və tipləşdirilməmiş faylların məzmununa həm ardıcıl həm də birbaşa müraciət üsulu mümkün olduğu halda, mətn fayllarına yalnız ardıcıl müraciət etmək olar.

8.2. Faylların təyini, açılması və bağlanması

Fayl tipinin təyini üçün **`Text, file`** və **`of`** işçi sözlərindən istifadə olunur. Bu işçi sözlərdən sonra faylın tipi göstərilir. Faylların ümumi təsvir forması belədir:

Mətn faylları	**`Var`** *fayl dəyişəni* **`: Text;`**
Tipləşdirilmiş fayllar	**`Var`** *fayl dəyişəni* **`: file of`** *faylın tipi***`;`**
Tipləşdirilməmiş fayllar	**`Var`** *fayl dəyişəni* **`: file ;`**

Burada, *fayl dəyişəni* fiziki faylla əlaqə yaradan məntiqi fayl dəyişəni, *faylın tipi* isə fiziki faylın tipidir. Mətn fayllarında istənilən tip ədədlər, simvol və mətnlər saxlana bilər və sətirlər müxtəlif uzunluqlu ola bilər. Tipləşdirilmiş fayllar tam, həqiqi, ikiqat dəqiqlikli və s. verilənlərdən ibarət (lakin eyni bir faylın məzmunu yalnız eynitipli və eyni uzunluqlu olmalıdır), tipləşdirilməmiş fayllar isə qarışıq tipli verilənlərdən ibarət olur.

Misal.

```
Var metn : file of text;
    f_tam : file of Integer;
    gir_fayl : file of real;
    son_fayl : file;
```

İnformasiya daşıyıcısında yerləşən fiziki faylla işləmək üçün əvvəlcə bu faylla əlaqə yaratmaq lazımdır. Bunun üçün fayl dəyişənindən istifadə edilir. Fiziki faylla əlaqə **`Assign`** proseduru vasitəsilə yaradılır. Bu prosedurun ümumi forması belədir:

`Assign` (*fayl dəyişəni, faylın ünvanı*)**;**

Misal.

```
Assign( f, 'Kafedra.dat' );
```

Bu operatorla, `f` *fayl dəyişəni* aktiv diskin cari qovluğunda yerləşən `Kafedra.dat` fiziki faylı ilə əlaqə yaradır. Faylın tam adını və ona müraciət yolunu göstərməklə prosedurun cari qurğudan asılılığını aradan qaldırmaq olar:

```
Fayl:='d:\AAHM\Kafedra.dat';
Assign(f,Fayl);
```

Fayla yolun ümumi uzunluğu *79*–a qədər simvoldan ibarət ola bilər.

Növbəti mərhələdə, informasiyanı fayldan oxumaq və ya fayla yazmaq üçün, bu fayl hökmən açılmalıdır. İnformasiyanı oxumaq üçün fayl **Reset** proseduru ilə açılır. Bu prosedurun ümumi forması belədir:

Reset (*fayl dəyişəni*)**;**

Fayla informasiyanı yazmaq üçün o, **Rewrite** proseduru ilə açılır. Bu prosedurun ümumi forması belədir:

Rewrite (*fayl dəyişəni*)**;**

İstənilən məqsədlə açılmış fayl sonda hökmən bağlanmalıdır. Faylın bağlanması **Close** proseduru ilə həyata keçirilir. Bu prosedurun ümumi forması belədir:

Close (*fayl dəyişəni*)**;**

`Reset` proseduru *fayl dəyişəni* ilə əlaqədə olan mövcud fiziki faylı açır. Əgər fiziki fayl mətn faylıdırsa, onda onun elementlərini yalnız ardıcıl müraciətlə oxumaq mümkündür. Əgər fiziki fayl tipləşdirilmiş fayldırsa, onda o, həm ardıcıl, həm də birbaşa müraciətlə oxunub–yazıla bilər.

Əgər göstərilən adlı fiziki fayl yoxdursa, onda prosedurun icrasında səhv meydana çıxır. Kompilyatorun **{$I–}** direktivini daxil edərək, `IOResult` funksiyasının köməyi ilə, faylın açılması əməliyyatının nəticəsini təhlil etmək mümkündür. Bu funksiyanın qiyməti *0* olarsa, deməli əməliyyat müvəffəqiyyətlə sona çatmışdır, əks halda səhv var.

`Rewrite` proseduru *fayl dəyişəni* ilə əlaqədə olan yeni fiziki fayl yaradır. Əgər bu adda fiziki fayl mövcuddursa, onda həmin faylın məzmunu pozulacaq və yeni fayl yaradılacaqdır.

`Close` proseduru faylı bağlayır. Yadda saxlayın ki, hər bir açılmış fayl hökmən bağlanmalıdır.

Fayllarla işləyərkən **Eof (***fayl dəyişəni***);** funksiyasından da istifadə olunur. Əgər cari göstərici faylın sonuncu elementinin mövqeyində yerləşərsə və ya fayl boş olarsa, onda **Eof (***fayl dəyişəni***);** funksiyasının qiyməti *True,* əks halda *False* olur.

Misal. *Massiv.dat* faylının elementlərinin cəminin hesablanması.

```
program fayl_oxumaq;
uses crt;
var
   f : file of Integer;
   X : Integer;
   S : Longint;
begin
  ClrScr;
  {$I –}
  Assign (f, 'Massiv.dat'); { faylla əlaqə yaradılır }
```

```
  Reset (f);   { fayl açılır }
  {$I+}
  if IOResult <> 0 then
  begin
    writeln('Faylın açılması səhvdir ');
    Halt(1);
  end;
  S:= 0;
  while not Eof(f) do
  begin
    Read(f,X); { fayl oxunur }
    S:=S+X;
  end;
  writeln('Faylın elementlərinin cəmi = ', S);
  Close (f); { fayl bağlanır }
end.
```

Bu proqramda, `assign` proseduru məzmunu tam ədədlər olan `Massiv.dat` faylı ilə əlaqə yaradır, `reset` proseduru ilə fayl açılır və bundan sonra, faylın bütün elementlərini oxumaq üçün ilkin şərtli dövr təşkil olunur. Faylın məzmunu `Read(f,X);` proseduru ilə oxunub, tam tipli `X` dəyişəninə mənimsədilir. Fayldakı informasiyanın miqdarı əvvəlcədən məlum olmadığı üçün dövr faylın sonu əlaməti rast gələnəcən təkrar olunur (`while not Eof(f) do`). Faylın açılmasında problem yarandıqda bu barədə məlumat verilir, `Halt` proseduru proqramın icrasını dayandırır və idarəetməni əməliyyat sisteminə ötürür.

8.3. Mətn faylları

Mətn fayllarını təsvir etmək üçün əvvəldən təyin olunmuş **`Text`** tipindən istifadə olunur:

`Var` *fayl dəyişəni* **: `Text`;**

Misal.

```
var
  MatnFile:Text;
  A,b:Text;
```

Mətn faylları praktiki olaraq ən çox tətbiq edilən fayllardır. Bu tip faylların üstün cəhəti ondan ibarətdir ki, burada verilənlər maşın kodları ilə deyil (tipləşdirilmiş və tipləşdirilməmiş fayllarda olduğu kimi), adi **ASCII** kodları ilə təsvir olunur. Belə ki, mətn fayllarında istənilən tip ədədlər, simvol və mətnlər saxlana bilər. Digər fayllardan fərqli olaraq, mətn faylları müxtəlif uzunluqlu sətirlərdən ibarət olur. Hər bir sətrin sonunda **`#13`**–karetkanın qaytarılması və **`#10`**–sətrin sonu simvolları olur. Sətrin hər bir simvoluna, birinci simvoldan başlayaraq, yalnız ardıcıl müdaxilə oluna bilər. Fərz edək ki, `f_ilk` faylının *4*

–cü elementini x dəyişəninə mənimsətmək lazımdır. Bunun üçün əvvəlki üç elementi "boş–boşuna" oxumaq lazımdır, yəni:

```
Reset(f_ilk); { f_ilk faylını açır və kursoru 1–ci elementin
                üzərinə yerləşdirir }
For i:=1 to 4 do
Read(f_ilk, x);
Reset(f_ilk); { kursoru yenidən 1–ci elementin üzərinə yerləşdirir }
```

Bu halda x dəyişəninə yalnız dördüncü sətirdəki qiymət mənimsədiləcəkdir.

Mətn tipli fayl yaratdıqda hər bir sətrin sonunda *Enter* klavişini basmaq lazımdır. Bu zaman sətrin sonuna **#13#10** kodları və ya xüsusi **EOLN** (*end of line*) əlaməti əlavə edilir. Hər bir faylın sonu əlaməti olmalıdır, bu zaman EOF (*end of file*) əlaməti yaranır. Faylın sonu əlaməti ***Ctrl+Z*** klavişlərini birgə basmaqla yaradılır.

Mətn fayllarından oxunma və fayla yazma əməliyyatları standart Read, Readln, Write və Writeln prosedurları ilə yerinə yetirilir və onların ümumi forması belədir:

Read (*fayl dəyişəni*, *dəyişənlərin siyahısı*);
Write (*fayl dəyişəni*, *dəyişənlərin siyahısı*);
Readln (*fayl dəyişəni*, *dəyişənlərin siyahısı*);
Writeln (*fayl dəyişəni*, *dəyişənlərin siyahısı*);

Read və Readln prosedurları ilə informasiya fayldan oxunaraq *dəyişənlərin siyahısında* göstərilmiş dəyişənlərə mənimsədilir. Write və Writeln prosedurlarında isə *dəyişənlərin siyahısında* göstərilmiş dəyişənlərin qiymətləri fayla yazılır.

Misal.

```
Read(f,A,B);
Write(g,'A=',A,'B=',B);
Readln(f,C,D);
Writeln(g,'C=',C,'D=',D);
```

Burada, f və g uyğun olaraq oxunan və yazılan fayllarla əlaqə yaradan fayl dəyişənləridir.

Sonrakı izahatları isə aşağıdakı misal üzərində davam etdirək.

Misal. *f(x)=x/(1+x)* funksiyasının cədvəlləşdirilməsi.

```
program text_file;
uses CRT;
var x:real;k:word;
    out_file:text;
```

```
function F(x:real):real;
 begin
    F:=x/(1.+x);
 end;
begin
 assign(out_file,'d:\user\qarayev\table.dat');
 rewrite(out_file);
 x:=0.0;
 writeln(out_file,'F(x)=x/(1+x)
 funksiyasının cədvəl qiymətləti');
 writeln(out_file);
 writeln(out_file,'x':10,'F(x)':25);
 writeln(out_file);
 for k:=0 to 50 do
 begin
  writeln(out_file,x:9:3,F(x):25:10);
  x:=x+0.1;
  if k mod 10=9 then writeln(out_file);
 end;
 close(out_file);
end.
```

Burada, `out_file` dəyişəni `Text` tipli mətn faylı dəyişəni kimi elan edilir. `Assign` proseduru bu dəyişəni `D:` diskindəki `user\qarayev` qovluğunda yerləşən `table.dat` faylı ilə əlaqələndirir. Əgər faylı proqramın yerləşdiyi qovluqda yaratmaq lazımdırsa, onda fayla yolu yazmamaq olar. `Rewrite` proseduru informasiya yazmaq üçün həmin faylı açır. Əgər diskdə eyniadlı fayl artıq mövcud olarsa, onda həmin faylın məzmunu pozulacaq, oraya yeni informasiya yazılacaqdır. Fayl mövcud olmazsa, onda o, yaradılacaqdır. Fayl açıldıqdan sonra, birinci dörd `writeln` prosedurları ilə fayla mətn və boş sətirlər yazılır. *x/(1+x)* funksiyasının qiymətləri isə fayla dövr operatoru daxilində yazılır. Bu zaman hər *10* sətirdən bir faylda bir boş sətir buraxılır. Fayla informasiya yazıldıqdan sonra, fayl `close(out_file)` proseduru ilə bağlanır.

Mətn faylları üçün əlavə olaraq aşağıdakı prosedur və funksiyalardan istifadə etməyə icazə verilir:

- **`Append`** –faylın sonuna elementləri əlavə etmək üçün mövcud faylı açır;
- **`Flush`** –faylın cari ölçüsünü qaytarır;
- **`Readln`** –`Read` proseduru kimi işləyir. Əlavə olaraq cari sətirdə qalan bütün simvolları buraxaraq göstəricini mətn faylının növbəti sətrinə yerləşdirir;
- **`SeekEof`** –mətn faylı üçün `Eof` vəziyyətini qaytarır;
- **`SeekEoln`** –mətn faylı üçün `Eoln` vəziyyətini qaytarır;
- **`SetTextBuf`** –mətn faylı üçün daxiletmə–xaricetmə buferi təyin edir;
- **`Writeln`** –`Write` proseduru kimi işləyir. Əlavə olaraq mətn faylına `Eoln` –"sətrin sonu" işarəsini yazır.

8.4. Tipləşdirilmiş fayllar

Tipləşdirilmiş fayllar belə təsvir olunur:

Var *fayl dəyişəni* **: file of** *faylın tipi***;**

Burada, *fayl dəyişəni* tipləşdirilmiş fayl dəyişəni, *faylın tipi* isə fayl tipi istisna olmaqla, istənilən tip ola bilər (`integer`, `longint`, `real`, `double`, `extended`, *massiv, yazı* və s.).

Misal.

```
type
     Nomr = file of Integer;
     Simv = file of 'A'..'Z';
Var  x: file of real;
     N: nomr;
     S: simv;
```

Tipləşdirilmiş faylların bütün elementləri eynitipli olmaqla bərabər, fayl tipindən başqa, ixtiyari tipli ola bilər. Ədəd tipli verilənləri yadda saxlamaq üçün tipləşdirilmiş fayllardan istifadə etmək əlverişlidir. Çünki, bu halda informasiya daha yığcam şəkildə yadda saxlanır. Tipləşdirilmiş fayllara ardıcıl və birbaşa müraciət etmək mümkündür. Qeyd etmək lazımdır ki, tipləşdirilmiş faylların elementləri həmişə sıfırdan başlayaraq nömrələnir.

Tipləşdirilmiş fayllardan informasiyanı oxumaq yalnız `Read` proseduru ilə, fayla informasiyanı yazmaq isə `Write` proseduru ilə yerinə yetirilir. Bu prosedurların ümumi forması aşağıdakı kimidir:

Read (*fayl dəyişəninin adı, dəyişənlərin siyahısı* **);**
Write (*fayl dəyişəninin adı, dəyişənlərin siyahısı* **);**

Tipləşdirilmiş fayllarla əməliyyatlarda aşağıdakı prosedur və funksiyalar nəzərdə tutulmuşdur:

- **FilePos** –göstəricinin fayldakı cari mövqeyinin nömrəsini qaytarır;
- **FileSize** –faylın cari ölçüsünü (elementlərinin sayını) qaytarır;
- **Seek** –göstəricinin fayldakı cari mövqeyini verilmiş nömrəli elementə dəyişdirir;
- **Truncate** –faylın ölçüsünü göstəricinin cari mövqeyinə qədər qısaldır. Faylın cari mövqedən sonrakı bütün elementləri silinir.

Misal. Mətn faylları ilə tipləşdirilmiş faylları müqayisə etmək üçün aşağıdakı proqrama baxaq. Bu proqramda *Sin(x)* funksiyası *0,001* addımı ilə cədvəlləşdirilir və alınmış *1000* ədəd qiymət həm mətn faylına, həm də tipləşdirilmiş fayla yazılır.

```
{$N+}
program file_of_extended;
uses CRT;
var extfile:file of extended;
    textfile:text;
    x,y:extended;
    i:word;
begin
   { Mətn faylının yaradılması }
 assign(textfile,'table.txt');
 rewrite(textfile);
 x:=0.0;
 for i:=1 to 1000 do
 begin
   y:=sin(x);
   writeln(textfile,y);
   x:=x+0.001;
 end;
 close(textfile);
   { Tipləşdirilmiş faylın yaradılması }
 assign(extfile,'table.ext');
 rewrite(extfile);
 x:=0.0;
 for i:=1 to 1000 do
 begin
   y:=sin(x);
   write(extfile,y);  {writeln yazıla bilməz}
   x:=x+0.001;
 end;
 close(extfile);
end.
```

Proqram icra olunduqdan sonra, proqramın yerləşdiyi qovluqda `table.txt` və `table.ext` faylları yaranacaqdır. `table.txt` faylı – mətn faylı, `table.ext` faylı isə tipləşdirilmiş fayldır. Baxmayaraq ki, hər iki faylın məzmunu eynidir, onlar arasında kəskin fərqlər vardır. Əgər mətn faylının məzmununa baxsaq, görərik ki, burada ədədlər bir sütun üzrə yerləşmişdir. Tipləşdirilmiş faylda isə xaotik yerləşmiş simvollar yığımı görəcəyik. Əgər bu faylların ölçülərini müqayisə etsək, görəcəyik ki, `table.ext` faylı *~10* kilobayt, `table.txt` faylı isə *~25* kilobayt həcmə malikdir. Bu ona görə belədir ki, `Extended` tipli dəyişənin uzunluğu *10* baytdır, ona görə də faylda *1000* həqiqi ədəd *10000* bayt (*~10* kilobayt) yer tutur. Lakin, `Extended` tipli dəyişəni mətn sətirləri kimi yadda saxladıqda, bu sətir *23* simvoldan – *17 rəqəm*, üstəgəl *6* simvoldan ibarət *dərəcə,* yəni $E \pm nnn$ və üstəgəl *sətrin sonu* və *karetkanın qaytarılması* simvollarından (cəmi *25* simvol) ibarət olduğu üçün, fayl *~25* kilobayt olacaqdır. Elə bu səbəbdən də rəqəm tipli verilənləri `Extended` tipli fayllarda yadda saxlamaq daha məqsədəuyğundur.

Tipləşdirilmiş faylda istənilən elementə müraciət etmək üçün

Seek (**Var** *fayl dəyişəni*; *n* : **LongInt**);

proseduru tətbiq edilir. Burada, *n* müraciət ediləcək elementin sıra nömrəsidir (*1*–ci elementin nömrəsi sıfırdan başlayır).

Tipləşdirilmiş faylın ölçüsünü müəyyən etmək üçün

FileSize (**Var** *fayl dəyişəni*):**LongInt;**

funksiyasından istifadə edilir. Bu funksiya *fayl dəyişəni* ilə əlaqələndirilmiş faylın uzunluğunu müəyyən edir, əgər fayl boşdursa, sıfır qiyməti qaytarır.

`Seek` və `FileSize` funksiyaları mətn faylları üçün tətbiq edilə bilməz.

8.5. Tipləşdirilməmiş fayllar

Tipləşdirilməmiş fayllarda müxtəlif tipli dəyişənlərin qiymətləri saxlanır. Belə faylları *ikilik fayllar* da adlandırırlar, çünki bu faylların məzmunu ikilik ədədlərdən ibarət olur. Tipləşdirilməmiş fayllarda onun tipi və strukturundan asılı olmayaraq istənilən elementə birbaşa müraciət etmək olur. Tipləşdirilməmiş faylları təsvir edərkən yalnız **file** işçi sözündən istifadə olunur və onun ümumi forması belədir:

Var *fayl_dəyişəni* : **File;**

Misal.

```
var Y:file;
```

Tipləşdirilmiş fayllar üçün nəzərdə tutulmuş bütün standart alt proqramlar həm də tipləşdirilməmiş fayllar üçün istifadə oluna bilər. Lakin, bu zaman `Read` və `Write` prosedurları müstəsnalıq təşkil edir. Belə ki, tipləşdirilməmiş fayllar üçün `Read` və `Write` prosedurlarının əvəzinə **BlockRead** və **BlockWrite** prosedurlarından istifadə olunur. Bu prosedurların ümumi forması belədir:

BlockRead (**var** *fayl dəyişəni* : **file;**
var *Buf*; *Count* : **word;** *Result* : **word**);

BlockWrite (**var** *fayl dəyişəni* : **file;**
var *Buf*; *Count* : **word;** *Result* : **word**);

Burada, *Buf* – bufer dəyişəni, *Count* – oxunacaq və ya yazılacaq baytların miqdarı, *Result* parametri isə həqiqətən oxunmuş və ya yazılmış informasiyanın faktiki qiymətidir. *Result* parametrinin qiyməti prosedur tərəfindən müəyyən edilir. Digər parametrləri isə proqramçı müəyyən etməlidir. *Buf* parametri fayla yazılacaq və ya fayldan oxunacaq informasiyadır və o, ümumi həcmi *n*Count* olan massivdir (*n* – `Reset` prosedurundakı parametrdir). *Buf* parametri ilə ötürülən informasiyanın maksimal həcmi (*n*Count*) *65520* baytı aşa bilməz.

Tipləşdirilməmiş fayllarla işlədikdə `Reset` və `Rewrite` prosedurlarının ümumi forması belə olur:

Reset (**var** *fayl dəyişəni*; *n* **: Word**);
Rewrite (**var** *fayl dəyişəni*; *n* **: Word**);

Burada, *n* əlavə parametrdir və onu yazmadıqda yazının ölçüsü susmaya görə *128* bayta bərabər götürülür, lakin *n* parametrinin yerində *1* yazmaq məsləhət görülür. Onun digər qiymətlərində yazılar fayla tam yazılmaya bilər. Tipləşdirilməmiş faylların yazılması və oxunmasını başa düşmək üçün disketlərin quruluşu ilə tanış olaq.

Disklərdə informasiya konsentrik yollarda yazılır. Bu yollar sektorlar adlanan ayrı–ayrı bərabər hissələrə bölünür. Belə sektorların hər birinin ölçüsü *512* bayta bərabərdir. Qeyd edək ki, yalnız eyni yollar üçün bu sektorların fəza uzunluqları eynidir. Yol mərkəzdən nə qədər uzaqda yerləşərsə, sektorun uzunluğu bir o qədər böyük olacaqdır. Bu onunla izah olunur ki, disk sabit bucaq sürəti ilə fırlandıqda elementin xətti sürəti onun yerləşdiyi yolun radiusu ilə mütənasibdir. İstənilən fayl klaster adlanan eyni miqdarda informasiyalardan ibarətdir. Diskin bir dövrü ərzində klasterə informasiya yazmaq və ya oxumaq olar. Ona görə də, əgər klasterin ölçüsü informasiyanın miqdarı ilə eyni olarsa, onda verilənlərin köçürülməsi ən yüksək sürətlə baş verir. Diskin klasteri *2* qarışıq sektordan ibarət olduğu üçün, onun ölçüsü *1024* baytdır. Vinçesterin klasteri *4* və ya *8* qarışıq sektordan ibarət olduğu üçün, onun ölçüsü uyğun olaraq *2048* və *4096* bayt olur.

8.6. Qovluq və fayllarla iş üçün ümumi vasitələr

Qovluq və fayllarla işləmək üçün **System** modulunun tərkibinə çoxlu prosedur və funksiyalar daxil edilmişdir. Bu prosedur və funksiyalara müraciət etmək üçün `uses` bölməsinə **System** modulunu qoşmaq vacib deyildir. Bu prosedur və funksiyaları çağırmaq həmişə mümkündür.

Qovluqlarla iş prosedurları. Qovluqlarla iş üçün **System** moduluna *Ms DOS* sisteminin analoji əmrləri olan **ChDir, MkDir, RmDir, GetDir** prosedurları daxil edilmişdir. Bu prosedurlardan istifadə etməklə proqram tərtib edək.

Misal.

```
program karaloqla_ish;
uses crt;
var
   S: String;
begin
  Clrscr;

  { D: diskində cari qovluğu təyin etmək }
```

```
  ChDir('D:\');

  { Cari disk və qovluğu göstərmək }
  GetDir(0,S);
  Writeln('Cari disk və qovluq:', S);

  {Kat_azal alt qovluğunun yaradılması }
  MkDir('Kat_azal');

  { Kat_azal alt qovluğuna keçid}
  ChDir('Kat_azal');

  { Cari disk və qovluğu göstərmək }
  GetDir(0,S);
  Writeln('Cari disk və qovluq:', S);

  { D: diskində cari qovluğu təyin eməkt }
  ChDir('\');

  { Kat_azal alt qovluğunun silinməsi }
  RmDir('Kat_azal');

  { Cari disk və qovluğu göstərmək }
  GetDir(0,S);
  Writeln('Cari disk və qovluq:', S);
end.
```

Faylların adının dəyişdirilməsi və pozulması. **Rename** prosedurundan fiziki faylların adının dəyişdirilməsində, **Erase** prosedurundan isə faylların pozulmasında istifadə olunur. Qeyd edək ki, bu prosedurlar hər hansı fiziki faylla əlaqəli olan məntiqi dəyişənlər üçün yerinə yetirilir. Aşağıdakı proqramda əvvəl Ilkfile.Dat faylının adı dəyişərək Deifile.Dat olur, sonra isə bu fayl pozulur.

Misal.

```
program fayl_ad;
uses crt;
var
   f: file;
begin
  Assign(f, 'Ilkfile.Dat');

  { Faylın adının dəyişdirilməsi }

  Rename(f, 'Deifile.Dat');
  Erase(f)
  Writeln('Cari disk və qovluq:', S);
end.
```

Fayllarla iş üçün prosedur və funksiyalar. Aşağıda təsvir olunan prosedur və funksiyalardan istənilən növ fayllarda istifadə etmək olar.

GetFTime proseduru. Faylın yaradılma və ya sonuncu yeniləşdirmə vaxtını qaytarır. Müraciət forması aşağıdakı kimidir:

GetFTime (*fayl dəyişəni*, *vaxt*)**;**

Burada, *fayl dəyişəni* – fayl dəyişəni kimi proqramda elan olunan identifikator, *vaxt* – isə `Longint` tipində olan dəyişəndir.

SetFTime proseduru. Faylın yaradılma və ya sonuncu yeniləşdirmə vaxtını təyin edir. Müraciət forması belədir:

SetFTime (*fayl dəyişəni*, *vaxt*)**;**

Burada, *vaxt* – yığcam formatdadır.

Qeyd edək ki, **DOS.TPU** modulunda aşağıdakı kimi `DateTime` tipi elan olunmuşdur:

```
type
   DateTime = record
     year :Word; { 19XX formatlı il }
     month:Word; { ay 1..12 }
     day  :Word; { gün 1..31 }
     hour :Word; { saat 0..23 }
     min  :Word; { dəqiqə 0..59 }
     sec  :Word; { saniyə 0..59 }
   end;
```

Yığcam formatdakı vaxtı `Longint` tipli dəyişənə çevirmək üçün aşağıdakı prosedurdan istifadə olunur:

PackTime (**var** `T`: **DateTime**; **var** `Time` : **Longint**)**;**

`Longint` tipli vaxtı geniş formatlı tipli dəyişənə çevirmək üçün aşağıdakı prosedurdan istifadə olunur:

UnpackTime (**var** `Time` : **Longint**; **var** `T`: **DateTime**)**;**

FExpand funksiyası. Faylın adına tam spesifikasiya, yəni disk, qovluq və faylın tipi əlavə edir. Müraciət forması belədir:

Fexpand (*faylın adı*)**;**

Burada, *faylın adı* sətir tipli ifadədir.

Fsearch funksiyası. Qovluqlar siyahısından faylın axtarışını təşkil edir. Müraciət forması belədir:

Fsearch (*faylın adı, qovluqların siyahısı*)**;**

Qeyd edək ki, faylın adında onun yolu da göstərilə bilər.

FindFirst proseduru. Göstərilən qovluqda qeyd olunan fayllardan birincisinin atributunu qaytarır. Müraciət forması:

FindFirst (*sətir tipli ifadə, atribut, ad*)**;**

Burada, *sətir tipli ifadə* – fayl və ya fayllar qrupu, *atribut* – `Byte` tipli ifadədir. Fayl atributları **DOS.TPU** modulunda aşağıdakı kimi elan olunur:

```
const
   ReadOnly   = $01;    { yalnız oxumaq üçün }
   Hidden     = $02;    { gizlədilmiş fayl }
   SysFile    = $04;    { sistem fayl ı}
   VolumeID   = $08;    { diskin identifikatoru }
   Directory  = $10;    { alt qovluğun adı }
   Archive    = $20;    { arxiv faylı }
   AnyFile    = $3F;    { istənilən fayl }
```

Qeyd edək ki, bu baytdakı bitlərin kombinasiyasından istifadə etməklə, müxtəlif variantları göstərmək olar. Məsələn, `$06` – bütün gizli və sistem fayllarını seçir.

`FindFirst` prosedurunun yerinə yetirilməsi nəticəsində `SearChrec` tipli dəyişən qaytarılır. Bu tip **DOS.TPU** modulunda aşağıdakı kimi təyin olunur:

```
type
   SearChrec = record
     Fill : array [1..21] of Byte;
     Attr : Byte;
     Time : LongInt;
     Size : LongInt;
     Name : String [12]
   end;
```

Burada, `Attr` – faylın atributu, `Time` – faylın yaradılmasının və ya sonuncu yeniləşdirilməsinin yığcam formatdakı vaxtı, `Size` – faylın baytlarla uzunluğu, `Name` – faylın adı və tipidir.

FindNext proseduru. Göstərilən qovluqdakı növbəti faylın adını qaytarır. Müraciət forması:

FindNext (*növbəti fayl*)**;**

Misal. `FindFirst` və `FindNext` prosedurlarından istifadə üsulları.

```
program fayl_siyahi;
uses crt,dos;
var
```

```
    S:SearchRec;
begin
  FindFirst('*.pas',AnyFile,S);
  while DosError = 0 do
  begin
    with S do
    writeln(Name:12,Size:12);
    FindNext(S);
  end
end.
```

Proqram cari qovluqdakı bütün *.pas* tipli faylların siyahısını ekrana çıxarır.

GetFAttr proseduru. Faylın atributunu – adını qaytarır. Müraciət forması:

GetFAttr (*fayl dəyişəni*, *atribut*)**;**

Burada, *atribut* – `Word` tipli dəyişəndir.

SetFAttr proseduru. Faylın atributunun təyininə imkan verir. Müraciət forması:

SetFAttr (*fayl dəyişəni*, *atribut*)**;**

DiskFree funksiyası. Göstərilən diskdə boş sahənin baytlarla ölçüsünü qaytarır. Müraciət forması:

DiskFree(*disk*) **:** `LongInt`**;**

Funksiyaya müraciətdə `Byte` tipli *disk* dəyişəni diskin nömrəsini təyin edir.

DiskSize funksiyası. Göstərilən diskin baytlarla tam ölçüsünü qaytarır. Müraciət forması:

DiskSize (*disk*)**:**`LongInt`**;**

Mövcud olmayan disk göstərilərsə, onda funksiya (*–1*) qiymətini qaytarır.

8.7. Fayllarla praktiki iş

Misal. Simvolları `Char` tipli fayla yazmalı və bu simvolların **ASCII** kodlarını ekrana çıxarmalı.

```
program Char_tipli_fayl;
uses crt;
var
   FC : file of Char;
   FB : file of Byte;
   Ch : Char;
   B  : Byte;
```

```
Begin
  ClrScr;
  { Fayl yaradılır }
  Assign (FC, 'D:\Test.dat');
  Rewrite (FC);
  for Ch := '0' to '9' do Write (FC, Ch);
  for Ch := 'A' to 'K' do Write (FC, Ch);
  Close (FC);

  { Fayl oxunur}
  Assign (FB, 'D:\Test.dat');
  Reset (FB);
    while not Eof(FB) do
      begin
        Read (FB,Ch);
        Write (b:8);
      end;
  Close (FB);
end.
```

Bu proqramda, əvvəl `Char` tipli fayla bir sıra simvollar yazılır, sonra bu `Byte` tipli fayl kimi açılır. Nəticədə bu fayla yazılan simvolların **ASCII** kodu çap olunur.

Misal. Faylın `A:` disketindən `D:` diskinə köçürülməsi (tipləşdirilməmiş fayl).

```
Program N_T_File;
uses crt;
const
     Razmer=1024; { Disk klasterinin ölçüsü }
     Ilk_file='A:\ilk_file' ;
     Son_file='d:\son_file' ;
var
     ilk, son:File;
     Bufer : array[1.. Razmer] of Byte;
     Rezultat:word;
begin
  Assign(ilk, ilk_file);
  Reset(ilk, Razmer);      { fayl oxunmaq üçün açılır }
  Assign(son, son_file);
  Rewrite(son, Razmer);    { fayl yazılmaq üçün açılır }
  While not EOF(ilk) do
    Begin
    { ilk faylından informasiya porsiyasını buferə oxuyuruq }
      BlockRead(ilk,Bufer,1,Rezultat);

    { son faylına buferdən fayl porsiyası yazırıq }
      BlockWrite(son,Bufer,1);
```

```
    end;
  Close(ilk);
  Close(son);
end.
```

Misal. Mətn faylındakı rəqəmlərin və latın hərflərinin ayrılıqda saylarının tapılması.

```
program Rqem_herf;
uses crt;
type
     Coxlug = set of Char;
const
     Nomr : Coxlug = ['0'..'9'];
     Harf : Coxlug = ['a'..'z', 'A'..'Z'];
var
   Snomr,
   Sharf : Word;
   Fp    : Text;
   Ch    : Char;
begin
  ClrScr;
  Assign (Fp, 'D:\NIZAMI.TXT');
  Reset (Fp); { fayl oxunmaq üçün açılır }
  Snomr:=0;Sharf:=0;
  while not eof (Fp) do
    begin
      Read(Fp, Ch);
      if Ch in Nomr then inc(Snomr);
      if Ch in Harf then inc(Sharf);
    end;
  close(Fp);
  writeln('Rəqəmlərin sayı = ', Snomr : 5);
  writeln('Hərflərin sayı', Sharf : 5);
end.
```

Bu proqramı icra etməzdən əvvəl, *Notepad* (*Bloknot*) mətn redaktoru vasitəsilə `Nizami.txt` adlı fayl yaratmaq lazımdır (çünki, Turbo Pascal yalnız **ASCII** simvollarını tanıyır). Proqramda iki çoxluq tipi müəyyənləşdirilmişdir: `Nomr` adlı `['0'..'9']` rəqəm formalı simvollar çoxluğu və `Harf` adlı `['a'..'z', 'A'..'Z']` kiçik və baş hərflər çoxluğu. `fp` məntiqi fayl dəyişəni `Nizami.txt` faylı ilə əlaqə yaradır. Faylın bütün simvollarını oxumaq üçün ilkin şərtli dövr operatorunda şərt kimi faylın sonu əlaməti yazılmışdır. Hər dövrdə fayl oxunaraq onun simvolları simvol tipli `ch` dəyişəninə mənimsədilir. Bu dəyişənin aldığı qiymətlərin `Nomr` və `Harf` çoxluğuna mənsub olması (`in` əməliyyatı) yoxlanır. Əgər simvol çoxluğa mənsubdursa, rəqəm və hərf sayğaclarının üzərinə vahid əlavə olunur, yəni
(`if Ch in Nomr then inc(Snomr);` və
`if Ch in Harf then inc(Sharf);`).

Misal. Mətn faylındakı bəzi simvolların sayının tapılması.

```
program Simvollar_sayi;
uses crt;
var
   gir_fayl:text;
   s: string;
   i, vergul, cumle, probel, setir:integer;
begin
  clrscr;
  vergul:=0; cumle:=0;
  probel:=0; setir:=0;
  Assign (gir_fayl,'D:\NIZAMI.TXT');
  Reset (gir_fayl);
  while not eof(gir_fayl) do
    begin
      Readln(gir_fayl, s);
      for i:=1 to Length(s) do
        begin
          case s[i] of
            ',' : inc(vergul); { Vergüllərin sayı }
            '.' : inc(cumle);  { Cümlələrin sayı }
            ' ' : inc(probel); { Probellərin sayı }
          end;
        end;
      inc(setir); { Sətirlərin sayı }
    end;
  close(gir_fayl);
  GotoXY(1,3);
  Writeln('Vergullərin sayı :', vergul);
  Writeln('Cumlələrin sayı  :', cumle);
  Writeln('Probellərin sayı :', probel);
  Writeln('Sətirlərin sayı  :', setir);
  Writeln(s);
end.
```

Bu proqramda, `assign` proseduru ilə `Nizami.txt` adlı fayl ilə əlaqə yaradılır, `reset` proseduru ilə fayl açılır və bundan sonra, faylın bütün elementlərini oxumaq üçün ilkin şərtli dövr təşkil olunur. Fayldakı informasiyanın miqdarı əvvəlcədən məlum olmadığı üçün, dövr faylın sonu əlaməti rast gələnəcən təkrar olunur (`while not Eof(gir_fayl) do`). Fayldan informasiya sətirbəsətir oxunur və sətirdəki hər bir simvolu təhlil etmək üçün `for` operatoru vasitəsilə yenidən dövr təşkil olunur. Dövr sətirlərdəki simvolların sayı qədər təkrar olunur (`for i:=1 to Length(s) do`). Burada, sətirlərin uzunluğunu `Length(s)` funksiyası hesablayır. `Case` operatoru vasitəsilə simvolların vergül, nöqtə və ya probelə uyğunluğu yoxlanır və sayğac vasitəsilə onların sayı tapılır. Sətirlərin sayı isə ilkin şərtli dövrün gövdəsində tapılır (`inc(setir);`).

Misal. *A(4,4)* matrisinin bütün müsbət elementlərinin və onların indekslərinin tapılması və nəticələrin faylda yadda saxlanılması.

```
program Musbet_element;
uses crt;
var a:array[1..4,1..4] of real;
    n,i,j:word;
    netice:text;
begin
 for i:=1 to 4 do
  begin
    write(i,'-ci sətir elementlərini daxil edin:');
    for j:=1 to 4 do readln(a[i,j]);
  end;
 clrscr;
 Assign(netice,'musbet_el');
 rewrite(netice); { fayl yazılmaq üçün açılır }
 writeln(netice,'sətir':6,'sütun':6,
 ' musbət elementlər':16);
 for i:=1 to 4 do
 for j:=1 to 4 do
 if a[i,j]>=0 then
 writeln(netice,i:6,j:6,' ':12,a[i,j]:8:4);
 close(netice);
 write('Proqramin sonu');
end.
```

Burada, *A(4,4)* matrisinin elementləri klaviaturadan daxil edilir, proqramın nəticələri isə `musbet_el` adlı fayl yaradılaraq orada saxlanır. Bu fayl pascal kompilyatorunun yerləşdiyi qovluqda yaradılacaqdır (çünki, proqramda faylın hansı qovluqda yaradılması üçün xüsusi yol göstərilməmişdir). `Assign` proseduru vasitəsilə `musbet_el` adlı fayl ilə əlaqə yaradılır, `rewrite(netice);` proseduru ilə fayl yazılmaq üçün açılır və ikiqat dövr daxilində müsbət elementlər tapılaraq `writeln(netice,i:6,j:6,' ':12,a[i,j]:8:4);` proseduru ilə fayla yazılır. Sonda fayl bağlanır (`close(netice);`).

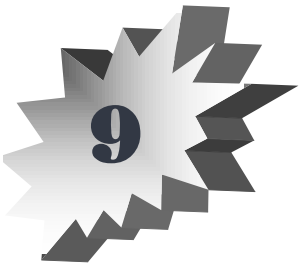
Misal. *A(5,5)* matrisinin hər bir sətir elementlərini cəmləyib, nəticələri faylda saxlayın.

```
program set_cem;
uses crt;
var a:array[1..5,1..5] of real;
    i,j:word;
    s:File of real;
    s1:real;
begin
 for i:=1 to 5 do
  begin
    write(i,'-ci sətir elementlərini daxil edin:');
```

```
      for j:=1 to 5 do readln(a[i,j]);
    end;
  clrscr;
  Assign(s,'son');
  rewrite(s);
  for i:=1 to 5 do
    begin
      s1:=0.;
      for j:=1 to 5 do
      s1:=s1+a[i,j];
      write(s,s1);
    end;
  close(s);
  write('Proqramın sonu');
end.
```

Bu proqramın iş prinsipi əvvəlki proqramın iş prinsipinə tamamilə analojidir.

Doqquzuncu fəsil



QRAFİKLƏRİN PROQRAMLAŞDIRILMASI

Bu fəsildə Turbo Pascal dilinin qrafik imkanları araşdırılacaq, ekranın mətn və qrafik rejimlərini, qrafik koordinat sistemini, qrafik rejimin qoşulması və ondan çıxış qaydalarını öyrənəcəyik. **`Graph`** moduluna daxil olan qrafik rejimin əsas sabitləri və alt proqramları, təsviri ekrana çıxarma prosedurları, qrafik primitivlərin qurulması və mətnin ekrana çıxarılması prosedurları müfəssəl izah olunacaqdır. Qrafik təsvirlərin qurulması üçün çox zəngin və maraqlı məsələlər proqramlaşdırılacaqdır.

9.1. Mətn və qrafik rejimlər

Qrafik proqramlaşdırmanın əsaslarını öyrənməzdən əvvəl, qrafik rejimin nə olduğunu araşdıraq. Məlumdur ki, informasiyanın təsvir edilməsi üçün ən çox monitordan istifadə edilir. Monitor (ekran) iki rejimdə işləyir: mətn və qrafik rejimlər. *Mətn rejimində* ekran *25∗80=2000* sayda mövqelərdən ibarət olur. Buna mətn rejimində *ekranın yolvermə qabiliyyəti* deyilir. Bu halda ekranda təsvir edilən ən kiçik obyekt *simvol* olur. Həm də bu halda monitorun rəng imkanları çox məhdud olur və bu rejimdə əsasən mətnlər təsvir edilir, lakin bəzi qrafik təsvirlər də ekrana çıxarıla bilər. Belə təsvirlər isə çox keyfiyyətsiz və primitiv olacaqdır.

Qrafik təsvirlərlə işlədikdə proqramçının sərəncamında olan rənglər yığımı böyük əhəmiyyətə malikdir. Rənglər yığımı *rənglər palitrasını* təşkil edir. Mümkün rənglərin miqdarı isə displey və videoadapterin imkanlarından və habelə videorejimdən asılıdır. Proqramlaşdırmada adətən üç rəngin – *qırmızı* (*Red*), *yaşıl* (*Green*) və *göy* (*Blue*) rənglərin qarışığından istifadə edilir ki, buna da *RGB təsvir sxemi* deyilir. *RGB* sxemin istifadə edilməsi elektron – şüa borusunun konstruktiv xüsusiyyətlərindən asılıdır. Elektron – şüa borusunda hər

bir qrafik – nöqtə yuxarıda göstərilən rənglərdən ibarət olur. Hər bir əsas rəng isə *256* intensivliyə malik olduğundan, rəng və çalarların ümumi sayı *256∗256∗256=16 777 216* olacaqdır. Lakin bütün qurğular, o cümlədən, şırnaqlı printerlər bu qədər rəngi təsvir etmək qabiliyyətinə malik deyildir. Ona görə də təsvirin çap edilməsi üçün, avtomatik olaraq, printerin imkan verdiyi rənglər seçilir. Analoji qayda şrift simvollarına da aiddir.

Monitoru qrafik rejimə keçirmək üçün xüsusi inisializasiya əməliyyatı yerinə yetirmək lazımdır. Bu əməliyyatdan sonra, **`Graph`** modulunun standart alt proqramlarının köməyi ilə, istənilən rəngli, istənilən qrafik çəkmək olar. Qrafik rejimdə ekran nöqtələrdən ibarət olur ki, bunlara da *piksellər* (*"picture element"*) deyilir. Simvolla müqayisədə piksel çox kiçikdir. Qrafik rejimdə ekran, məsələn, *EGA tipli* adapter *640∗350* (*350 sətir*, hər sətirdə *640 piksel*) sayda piksellərdən ibarət olur. Buna qrafik rejimdə *ekranın yolvermə qabiliyyəti* deyilir. Pikselin parametrlərini (rəng, parlaqlıq) yadda saxlamaq üçün videoyaddaş və ya videobuferdən istifadə edilir. Şəklin fasiləsiz təsviri üçün onu ən azı *25–30* hers tezliklə təkrar etmək lazımdır. Bundan kiçik tezliklərdə xoşagəlməz və yorucu təsvirlər alınır.

Təsviri yadda saxlamaq və təkrar etmək üçün kompüterin daxilində xüsusi plata vardır ki, ona *videokart* deyilir. Müasir videokartlar şəkli *100* və daha artıq hers tezliklə təkrarlayır.

Sətirlərin və sətirlərdə piksellərin sayı adapterdən və onun rejimlərindən asılıdır. Hər bir adapterin altıya qədər rejimləri ola bilər. Bu rejimlər bir–birindən ekranın yolvermə qabiliyyəti və mümkün rənglər yığımı ilə fərqlənir. Adapterlərin adları (identifikatorlar) və onların rejimləri **`Graph`** modulunun sabitlər (`Const`) bölməsində göstərilir. Bəzi adapterlərin maksimal yolvermə qabiliyyətinə baxaq (mötərizədə adapterin rejimlərinin sayı göstərilmişdir):

- *EGA (Enhanced Graphics Adapter)* – 640∗350 (*2*);
- *VGA (Video Graphics Array)* – 640∗480 (*4*);
- *İBM 8514* – 1024∗768 (*2*);
- *SVGA (super –VGA)* – 1024∗768;
- *SVGA (super–VGA)* – 1280∗1024.

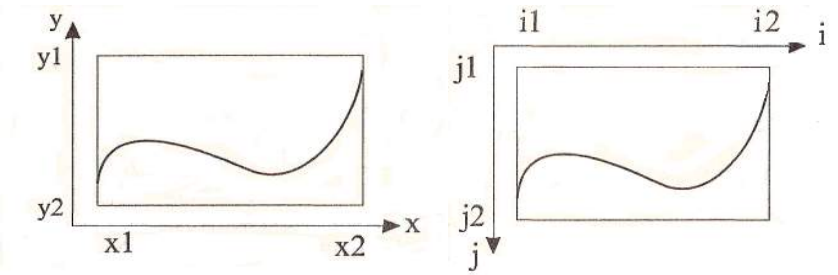
Monitor və adapter cansız qurğulardır. Onları idarə etmək üçün *drayver* adlanan xüsusi proqramlardan istifadə edilir. Məhz drayver sətirlərin, piksellərin sayını və s. müəyyən edir. Videoadapterlərin bütün drayverləri **Borland** firmasının yaratdığı `.BGI` (*Borland Graphics Interface*) tipli fayllarda yerləşir, məsələn, `cga.bgi`, `egavga.bgi` və s.

9.2. Qrafik koordinat sistemi

Qrafik proqramlaşdırmada əsas çətinlik ondan ibarətdir ki, adi halda qrafiklər kağız üzərindəki dekart koordinat sistemində layihələndirilir, kompüterdə isə həmin qrafiklər monitorun qrafik koordinat sistemində təsvir

edilməlidir. Monitorun koordinat sistemində isə koordinat başlanğıcı, yəni (*0,0*) nöqtəsi kimi, ekranın sol yuxarı küncü qəbul edilir (şəkil 9.1).

Hər hansı *y= f(x)* funksiyasını kağız üzərində qurmaq üçün *(X1,X2)∗(Y1,Y2)*



Şəkil 9.1. Adi və ekran koordinat sistemləri

ölçülü düzbucaqlı seçilir. Əsas problem bu funksiyanı ekranda çəkdikdə yaranır. Belə ki, bu halda *(X1,X2)∗(Y1,Y2)* düzbucaqlısını *(I1,I2)∗(J1,J2)* ekran düzbucaqlısına çevirmək lazım gəlir. Belə çevirmə üçün aşağıdakı düsturlardan istifadə edilə bilər:

$$\frac{x-X1}{X2-X1}=\frac{i-I1}{I2-I1},\quad \frac{y-Y1}{Y2-Y1}=\frac{j-J2}{J1-J2}.$$

Koordinatları çevirmək üçün bu düsturları aşağıdakı funksiya tipli alt proqramlar şəklində tərtib etmək olar.

```
Function II(x:real):integer;
{ X oxu üzrə çevirmə }
Begin
  II:=I1+Trunc((X-X1)*(I2-I1)/(X2-X1));
End;

Funstion JJ(y:real):integer;
{ Y oxu üzrə çevirmə }
Begin
  JJ:=J2+Trunc((Y-Y1)*(J1-J2)/(Y2-Y1));
End;
```

Ekran koordinatları tam ədədlər olmalıdır, ona görə də baxılan proqramlarda `Trunc` funksiyası tətbiq edilmişdir.

Proqramlaşdırma zamanı hər bir pikselin ünvanına da müraciət etmək və onları rəngləmək olar. Ən sadə qrafik rejimdə *256* rəng, xətti modelli videoyaddaş və ekranın *320∗200* yolvermə qabiliyyəti istifadə olunur. *320∗200* yolvermə qabiliyyətində ekranın buferinin ölçüsü *320∗200∗1=64000* bayt olmalıdır. Mümkün rənglərin sayı *256* olduğu üçün, pikselin rəngi haqqında

informasiyadan ibarət hər bir piksel üçün *1* bayt kifayət edir. Xətti modelli videoyaddaşın piksellərinin ünvanları cədvəl 9.1 – də göstərilmişdir.

Cədvəl 9.1. Xətti modelli videoyaddaşın piksellərinin ünvanları

	x=0	**x=1**	**…**	**x=318**	**x=319**
y=0	*0*	*2*	*...*	*318*	*319*
y=1	*320*	*321*	*...*	*638*	*639*
…	*...*	*...*	*...*	*...*	*...*
y=199	*63681*	*63682*	*...*	*63998*	*63999*

Bu cədvələ uyğun olaraq qrafik koordinatda hər bir pikselin ünvanını belə tapa bilərik:

```
Ünvan:=320*y+x;
```

9.3. Qrafik rejimin qoşulması və ondan çıxış

Qrafik rejimdə işləmək üçün bir neçə üsul mövcuddur.

1. Turbo Pascal dilinin inteqrallaşdırılmış mühitini qrafik rejimə kökləmək lazımdır. Bunun üçün mühitin *Options* menyusundan *Directiories* əmrini icra edib, açılan pəncərədə *Tab* klavişinin köməyi ilə, *Unit Directiories* rejimi qarşısında **Graph.tpu** faylına yolu göstərmək lazımdır (bu faylda **Graph** modulu yerləşir), məsələn, `D:\BP\BGİ`.

2. Turbo Pascal dilinin inteqrallaşdırılmış mühitini kökləmədən, sadəcə olaraq, **Graph.tpu** faylını **turbo.exe** faylının yerləşdiyi qovluğa köçürmək lazımdır.

Hər iki üsulda ekranı qrafik rejimə keçirmək üçün proqramın `Uses` bölməsinə **Graph** və **Graphs** modullarını əlavə etmək, yəni

```
Uses CRT, Graph, Graphs;
```

sətrini, icraedici hissədə isə **Open_Graph;** prosedurunu yazmaq lazımdır.

3. Qrafik rejimə keçdikdə, proqram videoadapterin növünü müəyyən etməlidir. Proqramda videoadapterin növünü aşkar göstərmək olar və ya uyğun parametrlərin qiymətlərini proqram özü seçə bilər. Videoadapterin növünü aşkar şəkildə göstərdikdə

```
InitGraph( gd, gm, 'C:\TP\BGI' );
```

prosedurunu çağırmaq lazımdır. Burada `gd` və `gm` adları sərbəst seçilmişdir və Siz ixtiyari adlar yaza bilərsiniz. `gd` dəyişəni videoadapterin növünü müəyyən edir və proqramın icraedici hissəsində ona qiymət mənimsədilməlidir. Belə qiymət kimi videoadapterin adı və ya onun kodundan ibarət sabitlər istifadə oluna bilər. Bu sabitlərdən bir neçəsi cədvəl 9.2 – də göstərilmişdir.

Cədvəl 9.2. Videoadapterin sabitləri

Sabitlər	Qiymətlər
CGA	1
MCGA	2
EGA	3
EGA64	4
EGAMono	5
HercMono	7
ATT400	8
VGA	9
PC3270	10

Videoadapterin növünü avtomatik olaraq müəyyənləşdirmək üçün proqramda

```
gd:=Detected;  və ya  gd:=0;
```

yazmaq lazımdır. `gd` dəyişəninin qiyməti müəyyənləşdirildikdən sonra, `gm` dəyişəninə qiymət müəyyənləşdirilməlidir. Bu dəyişən videoadapterin qrafik rejimini müəyyən edir. `gm` dəyişəninin ala biləcəyi mümkün qiymətlərdən bir neçəsi cədvəl 9.3 – də göstərilmişdir.

Cədvəl 9.3. Videoadapterin qrafik rejimləri

Sabitlər	Qiymətlər	Qrafik rejimin təsviri
EGALo	0	*640∗200, 16 rəng, 4 səhifə*
EGAHi	1	*640∗350, 16 rəng, 2 səhifə*
EGA64Lo	0	*640∗200, 16 rəng, 1 səhifə*
EGA64Hi	1	*640∗350, 4 rəng, 1 səhifə*
HercMonoHi	0	*720∗348 nöqtə, 2 səhifə*
VGALo	0	*640∗200, 16 rəng, 4 səhifə*
VGAMed	1	*640∗350, 16 rəng, 2 səhifə*
VGAHi	2	*640∗480, 16 rəng, 1 səhifə*
IBM8514Lo	0	*640∗480 nöqtə, 256 rəng*
IBM8514Hi	1	*1024∗768 nöqtə, 256 rəng*

Bu üsulda **`Open_Graph;`** prosedurunu yazmaq lazım deyil. Qrafik rejimlə işlədikdə səhv baş verərsə, **`İnitGraph`** proseduru sıfırdan fərqli nəticə – *səhv kodu* yaradır. Bu səhv haqqında məlumatı **`GraphResult`** funksiyası verir. Bunun üçün proqrama aşağıdakı proqram fraqmentini əlavə etmək lazımdır (sonrakı misallarda bu fraqment nümayiş etdiriləcəkdir):

```
uses Crt, Graph;
var Gd, Gm: Integer;
begin
  Gd := Detect;
  InitGraph(Gd,Gm,'c:\bp\bgi');
```

```
  if GraphResult <> grOk then
    begin
      Writeln('Qrafik rejimdə səhv aşkar edildi');
      Halt;
    end;
  ...
end.
```

Bu proqram fraqmentində

```
if GraphResult <> grOk then
```

sətri əvəzinə

```
if GraphResult <> 0 then
```

sətrini də yaza bilərik.

GraphResult funksiyası *15* məlumat verir. Onlardan üçünü göstərək:

- `GrOK= 0` –*qrafik rejim uğurla yerinə yetirilmişdir*;
- `GrNoİnitGraph= -1` –*qrafik rejim müəyyən edilməmişdir*;
- `GrFileNotFound= -3` –*qrafik drayver tapılmamışdır.*

Ekranı qrafik rejimdən çıxarmaq üçün proqramda **CloseGraph;** prosedurunu çağırmaq lazımdır.

RestoreCrtMode; və **SetGraphMode;** prosedurları ilə qrafik rejimi bağlamadan, mətn rejiminə və əksinə keçmək olar.

9.4. Qrafik rejimin əsas sabitləri və alt proqramları

9.4.1. Qrafik rejimin əsas sabitləri

Qrafik təsvirləri tətbiq etmək üçün **Graph** kitabxanasında *50*–dən çox prosedur və funksiya mövcuddur. Əvvəlcə, mövcud olan *16* sabiti sadalayaq (cədvəl 9.4). Bu sabitləri rəqəm və ya söz formasında istifadə etmək olar. Əyanilik üçün sözlərdən istifadə etmək daha məqsədəuyğundur.

Qrafikləri çəkdikdə elə etmək lazımdır ki, bir monitordan digərinə keçdikdə onların həndəsi ölçüləri pozulmasın. Bu məqsədlə proqramda

```
GetMaxX : İnteger;
```

və

```
GetMaxY : İnteger;
```

funksiyalarından istifadə etmək lazımdır. Bu funksiyalar monitorun maksimal koordinatlarını müəyyən edir. Belə ki, monitorun koordinat başlanğıcı sol yuxarı küncdə ((*0,0*) nöqtəsi) olduğu halda, sonu ekranın sağ aşağı küncündə, yəni (`GetMaxX,GetMaxY`) nöqtəsində olur.

Cədvəl 9.4. Əsas rəng sabitləri

Proqramda rəngin adına müraciət	Proqramda rəngin adına kodla müraciət	Rəngin adı
`Black`	0	*Qara*
`Blue`	1	*Göy*
`Green`	2	*Yaşıl*
`Cyan`	3	*Mavi*
`Red`	4	*Qırmızı*
`Magenta`	5	*Bənövşəyi*
`Brown`	6	*Qəhvəyi*
`Light Grey`	7	*Açıq boz*
`Dark Grey`	8	*Tünd boz*
`Light Blue`	9	*Açıq göy*
`Light Green`	10	*Açıq yaşıl*
`Light Cyan`	11	*Açıq mavi*
`Light Red`	12	*Çəhrayı*
`Light Magenta`	13	*Açıq qırmızı*
`Yellow`	14	*Sarı*
`White`	15	*Ağ*

Təsviri ekrana çıxarmaq üçün xüsusi prosedurlar nəzərdə tutulmuşdur.

9.4.2. Təsviri ekrana çıxarma prosedurları

`ClearDevice;` – qrafik ekranı təmizləyir, fonun rəngini müəyyən edir və cari mövqe kimi (*0,0*) nöqtəsini təyin edir.

`SetColor` (*reng* **`: word);`** – qələmin rəngini *reng* rəngli edir (məsələn, `SetColor(Red);` və ya `SetColor(4);` – qırmızı rəng).

`SetBkColor` (*reng*`:` **`word);`** – fonun rəngini *reng* rəngli edir (məsələn, `SetBkColor(Green);` və ya `SetBkColor(2);` – yaşıl rəng).

`Graph` modulunun əsas hissəsi nöqtə, düz xətt parçaları, qövs, çevrə və s. çəkmək üçün prosedurlardan ibarətdir. Bu qrafik elementlərə *qrafik primitivlər* deyilir. Qrafik primitivləri çəkmək üçün aşağıdakı prosedurlar tətbiq edilir :

`PutPixel` (*x, y* **`: İnteger;`** *reng* **`: word);`** – (*x, y*) koordinatlı nöqtədə pikseli *reng* rəngi ilə rəngləyir.

`Line` (*x1, y1, x2, y2* **`: İnteger);`** – cari rəngli rəqəmlə (*x1, y1*) başlanğıc və (*x2, y2*) – son nöqtələrdən keçən düz xətt parçası çəkir.

`Circle` (*x, y* **`: İnteger;`** *Radius*`:`**`word);`** – cari rəngli qələmlə mərkəzi (*x, y*) nöqtəsində yerləşən *Radius* – radiuslu çevrə çəkir.

Ellipse (*x, y* **: İnteger;** *Bash_bucaq, Son_bucaq, X_Radius, Y_Radius* **: word);**
proseduru cari rəngli qələmlə ellips çəkir. Burada, *Bash_bucaq* və *Son_bucaq* – ellips sektorunun başlanğıc və son bucaqlarıdır; (*x, y*) – ellipsin mərkəzidir.

Rectangle (*x1, y1, x2, y2* **: İnteger);** – cari rəngli qələmlə (*x1, y1*) – (*x2, y2*) diaqonallı düzbucaqlı çəkir.

Bar (*x1, y1, x2, y2* **: İnteger);** – sol yuxarı küncü (*x1, y1*), sağ aşağı küncü isə *(x2,y2)* nöqtəsində olan düzbucaqlı çəkir. Bu prosedurun `Rectangle` prosedurundan fərqi ondadır ki, çəkilən düzbucaqlının konturları olmur və düzbucaqlının daxilindəki sahə isə əvvəlcədən müəyyən edilmiş rənglə doldurulur.

FillEllipse (*x, y* **: İnteger;** *X_Radius, Y_Radius* **: Word);** – mərkəzi (*x, y*) nöqtəsində olan ellips çəkir və onun daxili sahəsi əvvəlcədən müəyyən edilmiş rənglə doldurulur.

Sector (*x, y* **: İnteger;** *Bash_Bucaq, Son_Bucaq, X_Radius, Y_Radius* **: Word);**
proseduru cari rəngli qələmlə ellips sektoru çəkir və onun daxili sahəsi əvvəlcədən müəyyən edilmiş rənglə doldurulur.

Qapalı həndəsi fiqurların rənglə doldurulması aşağıdakı prosedurlar vasitəsilə yerinə yetirilir :

SetFillStyle (*Doldurma, reng*: **Word);** – `Bar`, `FillEllipse`, `Sektor` və s. prosedurları ilə çəkilmiş fiqurları *reng* rəngi ilə doldurur. *Doldurma* parametri doldurma tərzini müəyyən edir. Doldurma tərzini müəyyən edən sabitlərin adları və kodları cədvəl 9.5 – də göstərilmişdir.

Cədvəl 9.5. Həndəsi fiqurların doldurulma tərzləri sabitləri

Sabitin adı	Sabitin kodu	Həndəsi təsviri
`EmptyFill`	0	*Fonun rəngi ilə tam doldurma*
`SolidFill`	1	*Verilmiş rənglə tam doldurma*
`LineFill`	2	*Üfqi xətlərlə doldurma*
`LtSlashFill`	3	*Diaqonal xətlərlə doldurma(///)*
`SlashFill`	4	*Qalın diaqonal xətlərlə doldurma(///)*
`BkSlashFill`	5	*Əksinə qalın diaqonal xətlərlə doldurma(\\\)*
`LtBkSlashFill`	6	*Əksinə diaqonal xətlərlə doldurma(\\\)*
`HatchFill`	7	*Tor şəkilli doldurma*
`XhatchFill`	8	*Maili tor şəkilli doldurma*
`InterleaveFill`	9	*Növbələşən xətlə doldurma*
`WideDotFill`	10	*Seyrək yerləşən nöqtələr*
`CloseDotFill`	11	*Sıx yerləşən nöqtələr*
`UserFill`	12	*İstifadəçinin müəyyən etdiyi üslub*

`FloodFillStyle` (*x, y* **`: İnteger;`** *reng_xett* **`: Word`**`);` – *reng_xett* rəngli xətti ilə əhatələnmiş (*x,y*) nöqtəsi ətrafında yerləşən bütün sahəni `SetFillStyle` şablonu ilə doldurulur.

Qeyd. Qrafik rejimdə `Write` və `Writeln` prosedurları korrekt işləməyə bilər. Onların əvəzinə xüsusi mətni ekrana çıxarma prosedurları tətbiq edilməlidir.

9.4.3. Mətni ekrana çıxarma prosedurları

Mətn rejimində ekranda mətni təsvir etmək üçün adətən *8∗8* pikselli matrisdən istifadə edilir. Belə matrisin tutduğu yer *işarə tutumu* adlanır. Yuxarıda qeyd olunduğu kimi, ən çox *80* simvoldan ibarət *25* sətirlik, yəni *80∗25=2000* yolvermə qabiliyyətinə malik ekrandan istifadə olunur. Bununla bərabər, *8∗14, 8∗16, 9∗14, 9∗16* işarə tutumlu matrisə və *40∗25, 80∗43, 80∗50* yolvermə qabiliyyətinə malik ekranlar da mövcuddur.

Qrafik rejimdə ekrana mətn çıxarmaq üçün aşağıdakı prosedurlardan istifadə edilir:

`OutTextXY` (*x,y* **`: İnteger;`** *Text* **`: string`**`);` –(*x,y*) nöqtəsindən başlayaraq *Text* mətnini ekrana çıxarır.

`SetTextStyle` (*font, hj, size*)**;** − ekranda mətnin yerləşməsini idarə edir. Burada *font* parametri vektor şriftin adını, *hj* parametri mətnin üfqi və ya şaquli istiqamətdə yerləşməsini, *size* parametri isə şriftin ölçüsünü (miqyas əmsalı) müəyyən edir. *font* parametrinin ala biləcəyi mümkün sabitlər cədvəl 9.6 – da göstərilmişdir.

Cədvəl 9.6. Turbo Pascal dilinin vektor şriftləri

Sabitin adı	Şriftin kodu	Şriftin yerləşdiyi fayl
`TriplexFont`	1	*trip.chr*
`SmallFont`	2	*litt.chr*
`SansSerifFont`	3	*sans.chr*
`GothicFont`	4	*goth.chr*

Əgər proqram cədvəldə göstərilən şriftləri tapa bilməzsə, onda səhv baş vermir, sadəcə olaraq susmaya görə təyin olunmuş şrift istifadə olunur. Bu prosedurun ikinci parametri *hj* iki qiymət ala bilər:

- `HorizDir` –mətn *üfqi* (adi) istiqamətdə yerləşir;
- `VertDir` –mətn *şaquli* istiqamətdə yerləşir.

Nəhayət, prosedurun üçüncü parametri – *size* şriftin miqyas əmsalıdır, məsələn, rastr şriftlər üçün *1* miqyas əmsalı *8∗8* pikselli matrisə, *2* əmsalı isə *16∗16* pikselli matrisə uyğun gəlir.

SetTextJustufy (*horj*, *verj*)**;** – ekranda mətni üfqi (birinci parametr) və şaquli (ikinci parametr) istiqamətlərdə düzləndirir. Düzləndirmə formaları cədvəl 9.7 – də göstərilmişdir.

Cədvəl 9.7. `SetTextJustufy` prosedurunda düzləndirmə formaları

Üfqi istiqamətdə			Şaquli istiqamətdə		
Sabitin adı	**Kodu**	**Düzlən-dirmə**	**Sabitin adı**	**Kodu**	**Düzlən-dirmə**
`LeftText`	0	*Sola tərəf*	`BottomText`	0	*Aşağıya tərəf*
`CenterText`	1	*Mərkəzə görə*	`CenterText`	1	*Mərkəzə görə*
`RightText`	2	*Sağa tərəf*	`TopText`	2	*Yuxarıya tərəf*

ClearDevice; – cari göstəricini ilkin vəziyyətə, yəni (*0,0*) koordinatlı nöqtəyə yerləşdirir və ekranı fonun rəngi ilə rəngləyərək onu təmizləyir.

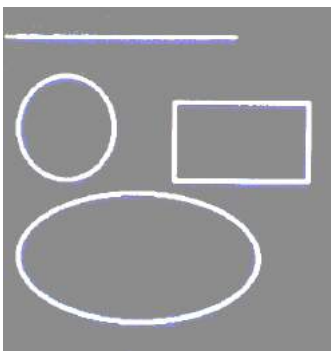
RestoreCrtMode; – sistemi qrafik rejimdən əvvəlki mətn rejiminə qaytarır.

9.5. Qrafiklərin qurulmasına aid misallar

Yuxarıda öyrəndiyimiz prosedur və funksiyaları tətbiq etməklə proqramlar tərtib edək.

Misal. Qrafik primitivlərin qurulması.

```
program fiqurlar;
uses Crt, Graph,graphs;
begin
  clrscr;
  Open_graph;
  line(10,50,200,50);
  Circle(60,120,40);
  Rectangle(150,100,260,160);
  Ellipse(120,220,0,360,100,50);
  Readln;
  CloseGraph;
end.
```



Şəkil 9.2. Qrafik primitivlər

Burada, `line` proseduru ilə (*10,50*) və (*200,50*) nöqtələrini birləşdirən düz xətt, `Circle` proseduru ilə mərkəzi (*60,120*) nöqtəsində olan, radiusu *40* piksel olan çevrə, `Rectangle` proseduru ilə diaqonallarının koordinatları (*150,100*) və (*260,160*) olan düzbucaqlı və `Ellipse` proseduru ilə mərkəzi (*120,220*) nöqtəsində, başlanğıc bucağı *0*, radiusu *100* və son bucağı *360* olan (yəni qapalı), radiusu *50* piksel olan ellips çəkilir (şəkil 9.2). Əgər `Ellipse`

prosedurunda başlanğıc və son bucaqları, uyğun olaraq, məsələn, *130* və *400* götürsəniz, qapalı olmayan oval alacaqsınız; əgər başlanğıc və son radiusları *100* və *150* götürsəniz, dartılmış (şaquli) ellips alacaqsınız; əgər başlanğıc və son radiusları eyni, məsələn, *100* götürsəniz, onda çevrə çəkiləcəkdir. Ümumiyyətlə, təqdim olunan bütün qrafik proqramlarda, parametrlərə müxtəlif qiymətlər verərək onların necə dəyişməsini müşahidə etmək lazımdır.

Misal. Qrafik primitivlər və mətn.

```
program fiqur_metn;
uses Crt,Graph,graphs;
begin
  Open_graph;
  line(10,50,200,50);
  Circle(60,120,40);
  Rectangle(150,100,260,160);
  Ellipse(120,220,0,360,100,50);
  OutTextXY(83,220,'TURBO PASCAL');
  Readln;
  CloseGraph;
end.
```

Bu proqramda, yuxarıdakı misalda çəkilmiş həndəsi fiqurlar təkrar olunur, fərq ondadır ki, ellipsin daxilində `TURBO PASCAL` mətni yazılacaqdır.

Misal. Üçbucağın çəkilməsi.

```
Program ucbucaq1;
uses Crt, Graph, Graphs;
begin
  Open_graph;
  line(300,100,450,450);
  line(300,100,150,450);
  line(150,450,450,450);
  Readln;
  CloseGraph;
end.
```

Bu proqramda, `line` proseduru ilə *3* düz xətt çəkilir. Lakin, koordinatlar elə seçilmişdir ki, bu düz xətlər üçbucaq əmələ gətirir. Bu proqramı belə də yaza bilərik:

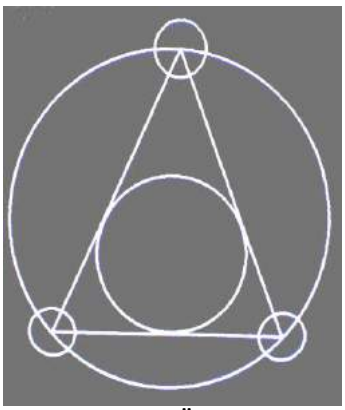
```
Program ucbucaq2;
uses Crt, Graph;
var Gd, Gm: Integer;
begin
  Gd := Detect;
  InitGraph(Gd, Gm, 'c:\bp\bgi');
  if GraphResult <> grOk then
    begin
      Writeln('Səhv aşkar edildi');
      Halt(1);
    end;
```

```
  line(300,100,450,450);
  line(300,100,150,450);
  line(150,450,450,450);
  Readln;
  CloseGraph;
end.
```

Nəticə nöqteyi–nəzərindən bu proqramın yuxarıdakı proqramdan heç bir fərqi yoxdur. Lakin, burada qrafik rejimin qoşulması üçün yuxarıda qeyd etdiyimiz üçüncü üsuldan istifadə edilmişdir, yəni `uses` operatorunda **`Graphs`** modulu qoşulmamış, onun əvəzinə proqramın icra hissəsində `InitGraph` proseduru istifadə edilmişdir.

Misal. Üçbucaq və çevrələrin çəkilməsi.

```
program ucbucaq_cevre1;
uses Crt, Graph, Graphs;
begin
  Open_Graph;
  Circle(310,70,30);
  line(310,70,450,400);
  line(310,70,150,400);
  Circle(150,400,30);
  Circle(304,303,95);
  Circle(300,270,200);
  line(150,400,450,400);
  Circle(450,400,30);
  Readln;
  CloseGraph;
end.
```



Şəkil 9.3. Üçbucaq və çevrələrin qurulması

Bu proqramla düz xətlər və çevrələr çəkilir. Prosedurların parametrlərinin qiymətləri elə seçilmişdir ki, düz xətlər üçbucaq əmələ gətirir və bu üçbucağın daxilinə, xaricinə və mərkəzləri üçbucağın təpə nöqtələrində yerləşən çevrələr çəkilir (şəkil 9.3.).

Bu proqramı belə də yaza bilərik:

```
program ucbucaq_cevre2;
uses Crt,Graph;
var Gd,Gm: Integer;
begin
  Gd := Detect;
  InitGraph(Gd, Gm, 'c:\bp\bgi');
  if GraphResult <> 0 then
    begin
      Writeln('Qrafik rejimdə səhv!');
      Halt(1);
    end;
  Circle(310,70,30);
  line(310,70,450,400);
  line(310,70,150,400);
```

```
  Circle(150,400,30);
  Circle(304,303,95);
  Circle(300,270,200);
  line(150,400,450,400);
  Circle(450,400,30);
  Readln;
  CloseGraph;
end.
```

Biz indiyədək həndəsi primitivləri çəkdikdə onların parametrlərinə konkret qiymətlər verirdik. Bu parametrlər dövr altında müxtəlif qiymətlər alan dəyişənlər də ola bilər. Məhz bu halda, həndəsi fiqurların ölçüləri dəyişdiyindən, daha maraqlı təsvirlər alınır (həndəsi fiqurlar dinamiki dəyişir). Bu andan etibarən tərtib edəcəyimiz bütün proqramlarda belə xarakterli məsələlər əhatə olunacaqdır. Həmin məsələlərdə şərh işarələri (**{** və **}**) daxilində yazılmış operatorlar və prosedurlar görəcəksiniz. Proqramları yerinə yetirdikdə həmin şərh simvollarını pozub (ona analoji sətirləri isə, əksinə, şərh simvolları daxilinə alıb) proqramın nəticələrinin necə dəyişməsini müşahidə etməyi təkidlə tövsiyə edirik, bu halda proqramı daha dərindən qavraya biləcəksiniz.

Sonrakı misalların əksəriyyətində `Delay` prosedurundan istifadə edilmişdir. Onun ümumi forması belədir:

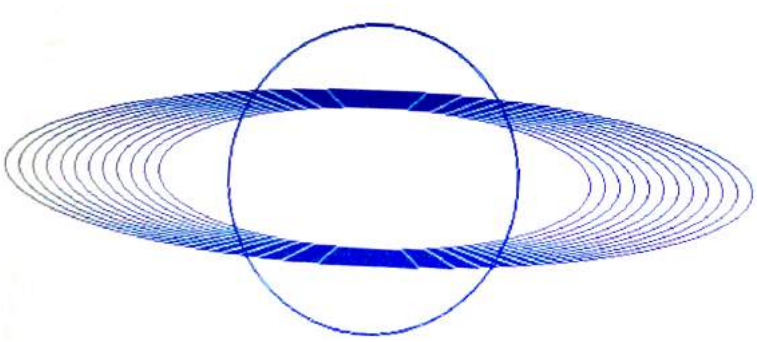
`Delay` (*time*)**;**

Bu prosedur proqramın icrasını *time* parametri ilə göstərilmiş millisaniyələr qədər ləngidir. *time* parametrinə qiymət verdikdə kompüterinizin sürətini nəzərə alın: əgər kompüteriniz köhnədirsə, bu parametrə çox böyük qiymətlər verməyin.

Misal. Çevrə və onun ətrafına konsentrik ellipslərin çəkilməsi.

```
program peyk;
uses Crt, Graph,graphs;
var i,t:word;
begin
  clrscr;
  Open_graph;
  circle(300,220,100);
  circle(300,220,101);
  t:=0;
  for i:=1 to 12 do
    begin
      t:=i+5;
      {delay(50000);}
      ellipse(300,220,0,360,90+10*t,40+t);
      {Ellipse(300,220,130,410,90+10*t,40+t);}
    end;
  Readln;
  CloseGraph;
end.
```

Proqramın nəticəsi sanki Saturn planetini və onun peyklərini xatırladır (şəkil 9.4). `for` operatorunda *12* əvəzinə digər ədədlər yazmaqla – ellipslərin sayını, `t`–nin ifadəsində addımı dəyişməklə – ellipslər arasındakı məsafəni dəyişdirə bilərsiniz. `{Ellipse(300,220,130,410,90+10*t,40+t);}` prosedurunu qoşsanız ellipslər qapalı olmayacaq (sanki, planetin arxa hissəsi görünməyəcəkdir). Və nəhayət, şəklin çəkilməsi prosesini müşahidə etmək istəyirsinizsə, `delay` prosedurunu qoşun.

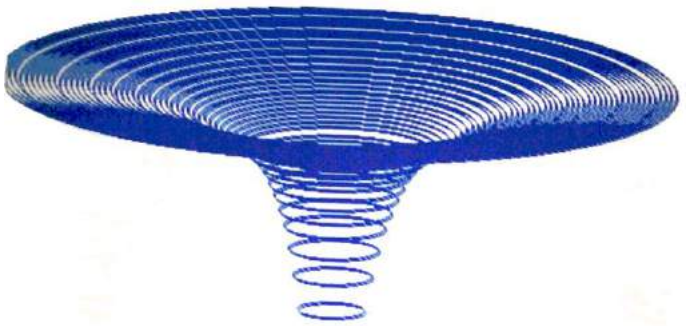


Şəkil 9.4. "Saturn" planeti

Misal. "Qara dəlik" təsviri.

```
program burulqan;
uses Crt,Graph,
     graphs;
var i,t:word;
     x,y:integer;
begin
  clrscr;
  Open_graph;
  setBkColor(15);
  for i:=60 downto 5 do

    begin
      t:=i+1;
      y:=trunc(400/t);
      x:=5*t;
      SetColor(1);
      {Delay(50000);}
      Ellipse(320,200+3*y,0,360,x,trunc(x/5));
      ellipse(321,201+3*y,0,360,x,trunc(x/5));
      ellipse(322,202+3*y,0,360,x,trunc(x/5));
    end;

  Readln;
  CloseGraph;
end.
```

Bu proqramın nəticəsi gur çaylarda əmələ gələn burulğanı xatırladır, əslində isə bu təsvir təbiət hadisəsi olan "qara dəlik" təsviridir (şəkil 9.5). Bu proqramda biz `SetBkColor(15);` və `SetColor(1);` prosedurlarını tətbiq etdik. Birinci prosedur ekranı (fonu) `15` rəngi ilə, yəni ağ rənglə rəngləyir, ikinci prosedur isə `1` rəngi ilə, yəni göy rənglə təsviri çəkir (qələmin rəngi). `15` və `1` ədədlərinin əvəzinə uyğun olaraq `white` və `blue` yaza bilərik.



Şəkil 9.5. "Qara dəlik"

Misal. Kola üçün qədəh təsviri.

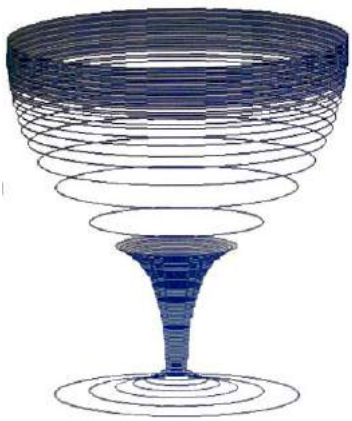
```
program qedeh;
uses Crt,
    Graph,graphs;
var i,t:word;
    x,y,st:word;
begin
 clrscr;
 Open_graph;
 {setBkColor(11);}
 {SetColor(4); }

   { qədəh hissəsi }
 for i:=200 downto 2 do
  begin
    x:=Trunc(100*cos(pi/i));
    y:=Trunc(100*sin(pi/i));
    {Delay(50000); }
    Ellipse(220,65+trunc(2.35*y),
    0,360,2*x,trunc(x/3));
  end;

 { qədəhin ayaq hissəsi }
 st:=y;
 for i:=10 to 75 do
  begin
    t:=5+i;
    x:= trunc(35/(t/15));
```

```
      {Delay(50000); }
      Ellipse(220,65+2*(st+t),0,360,2*x,trunc(x/3));
    end;
  st:=st+i;

  { qədəhin oturacaq hissəsi }
  for i:=2 to 10 do
    begin
      x:=trunc(2.5*exp(i/3));
      {Delay(50000); }
      Ellipse(220,65+2*(st+2*i),0,360,2*x,trunc(x/3));
    end;
  Readln;
  CloseGraph;
end.
```

Proqramın nəticəsi şəkil 9.6 – da göstərilmişdir. Qədəhin şüşə divarlarının çəkilməsində *sinx* və *cosx* funksiyalarından istifadə edilmişdir. Qədəhin oturacaq hissəsində ellipslərin radiuslarının kiçik qiymətlərdən böyük sürətlə artması üçün eksponensial funksiya tətbiq edilmişdir. Dövrlərin parametrlərinin qiymətlərini, ellipslərin parametrlərini, rəngləri dəyişdirməklə və `Delay` prosedurunu qoşmaqla, Siz, çox maraqlı təsvirlər izləyə biləcəksiniz.

Misal. Müxtəlif tərzlərdə mətnin ekrana çıxarılması.

```
program metn;
uses Crt,Graph,graphs;
var ad:string;
begin
  clrscr;
  gotoxy(5,5);
  writeln('Mətni daxil
   edin:');read(ad);
  Open_graph;
  settextstyle(2,VertDir,15);
  setcolor(2);
  OutTextXY(10,10,ad);
  settextstyle(4,HorizDir,55);
  setcolor(3);
  OutTextXY(100,100,ad);
  Readln;
  CloseGraph;
end.
```



Şəkil 9.6. Kola üçün qədəh

Proqram icra olunduqda mətnin daxil edilməsini tələb edir. Siz, mətn daxil etdikdən sonra, həmin mətn ekranda iki formada təsvir olunacaqdır. Birinci formada ekranın (*10,10*) nöqtəsində `SmallFont` şrifti ilə ölçüsü *15* punkt olan yaşıl rəngli, şaquli istiqamətdə mətn təsvir olunacaqdır. Bunu `settextstyle(2,VertDir,15);` proseduru həyata keçirir (burada *2*

ədədi `SmallFont` şriftinin kodudur). İkinci formada ekranın (*100,100*) nöqtəsində `GothicFont` şrifti ilə ölçüsü *55* punkt olan mavi rəngli, üfqi istiqamətdə mətn təsvir olunacaqdır. Bunu `settextstyle(4,HorizDir, 55);` proseduru həyata keçirir (burada *4* ədədi `GothicFont` şriftinin kodudur). Mətnin ekrana çıxarılmasını `OutTextXY` proseduru yerinə yetirir. Mətnin rəngini isə `setcolor` proseduru müəyyənləşdirir.

Misal. Ekran qoruma proqramı.

```
program ekran_qoruma1;
uses crt,graph,graphs;
var x,y:integer;
    a:string;
begin
 clrscr;
 gotoxy(5,5);
 write('Mətni
       daxil edin:');
 read(a);
 Open_graph;
 Randomize;
  repeat
   x:=random(800);
   y:=random(600);
   settextstyle(random(6),
   random(2),random(10));
   if (x+textheight(a)<500) and
   (y+textwidth(a)<400) then
    begin
      setcolor(random(15)+1);
      outtextxy(x,y,a);
      delay(60000);
      cleardevice;
    end;
  until keypressed;
 readln;
 closegraph;
end.
```

Bu proqram ekranı qoruma funksiyasını həyata keçirir. Daxil edilmiş mətn ekranın ixtiyari təsadüfi hissəsində təsadüfi ölçülü, təsadüfi rəng və təsadüfi istiqamətlərdə təsvir olunur. Proqramda **`TextHeight`** və **`TextWidth`** prosedurlarından istifadə edilmişdir ki, bu prosedurlar uyğun olaraq, piksellərlə, sətrin *hündürlüyü* və *enini* müəyyən edir. Bu proqramda **`Crt`** modulundan `Keypressed` proseduru da istifadə edilmişdir. İstənilən klaviş basıldıqda bu prosedur `True` qiyməti qaytarır. Ona görə də ixtiyari klavişi basdıqda dövrdən çıxış baş verəcək və proqram öz işini dayandıracaqdır. `Random` funksiyası ilə təsadüfi ədədlər yaradılır. Bu ədədlər ekran koordinatlarını müəyyənləşdirir. `if` operatoru ilə mətnin ekranın hüdudlarından kənara çıxmamasına nəzarət edilir.

`Settextstyle` prosedurundakı `Random` funksiyası isə şriftin adına, mətnin istiqamətinə və şriftin ölçüsünə uyğun təsadüfi ədədlər yaradır. Son şərtli dövr operatoru təkrar olunduqca, `cleardevice;` proseduru hər dəfə ekranı təmizləyir. Əgər bu proseduru pozsanız, müəyyən müddətdən sonra ekran mətnlərlə tam dolacaqdır. İxtiyari klavişi basdıqda proqramdan çıxış baş verir.

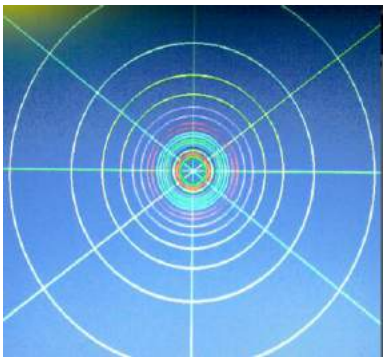
Bu proqramı belə də yaza bilərik:

```
program ekran_qoruma2;
uses crt,graph;
const
     drive:integer=EGAHI;
     mode:integer=EGA;
     path='c:\bp\bgi';
var i,x,y:integer;
     a:string;
begin
  clrscr;
  gotoxy(5,5);
  writeln('Mətni daxil edin:');
  readln(a);
  İnitGraph(drive,mode,path);
  Randomize;
    repeat
    x:=random(800);
    y:=random(600);
    settextstyle(random(6),
    random(2),random(10));
    if (x+textheight(a)<500) and
    (y+textwidth(a)<400) then
      begin
        setcolor(random(15)+1);
        outtextxy(x,y,a);
        delay(50000);
        cleardevice;
      end;
    until keypressed;
  readln;
  closegraph;
end.
```

Misal. ”Hörümçək toru” təsviri.

```
program Horumchek_Toru;
uses crt,graph,graphs;
var
   i:word;
begin
  open_graph;
  SetBkColor(blue);
  SetColor(LightCyan);
```

```
  Line(0,0,GetMaxX,GetMaxY);
  Delay(1000);
  SetColor(Yellow);
  Line(0,GetMaxY,GetMaxX,0);
  Delay(1000);
  SetColor(Lightgreen);
  Line(0,GetMaxY div 2,
  GetMaxX,GetMaxY div 2);
  Delay(1000);
  SetColor(Lightgray);
  Line(GetMaxX div 2,0,
  GetMaxX div 2, GetMaxY);
  Delay(1000);
  SetColor(Lightred);

  for i:=2 to 20 do
    begin
      SetColor(16-i div 2);
      Circle(GetMaxX div 2,
      GetMaxY div 2, GetMaxY div i);
      Delay(5000-15*i);
    end;

  Readln;
  closegraph;
end.
```



Şəkil 9.7. ”Hörümçək toru” təsviri

Bu proqram koordinat sistemini qurur və koordinat başlanğıcından keçən düz xətlər və mərkəzi koordinat başlanğıcında olan konsentrik çevrələr çəkir ki, bu da hörümçək torunu xatırladır. Proqramda `GetMaxX` və `GetMaxY` funksiyalarından istifadə edilmişdir. Bu funksiyalar, proqramı digər monitorlu kompüterdə işlətdikdə, qrafikin çəkilməsində həndəsi ölçülərlə əlaqədar baş verə biləcək problemin əmələ gəlməsinin qarşısını alır. Proqramın nəticəsi şəkil 9.7–də göstərilmişdir. `for` operatorunun parametrlərini dəyişməklə, Siz, müxtəlif təsvirlər müşahidə edəcəksiniz.

Misal. Serpinski “xalısının” qurulması.

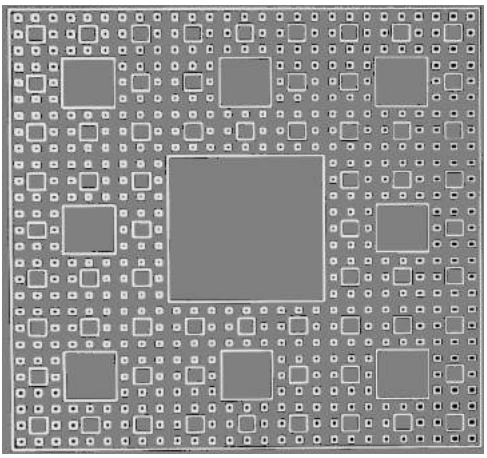
Serpinski “xalısının” qurulması üçün əvvəlcə tərəfi vahidə bərabər olan kvadrat götürülür, sonra kvadratın hər tərəfi üç bərabər hissəyə, kvadratların özləri isə tərəfi *1/3* olan *9* bərabər hissəyə bölünür. Alınan həndəsi təsvirdən mərkəzi kvadrat çıxarılır. Daha sonra, yaranmış *8* kvadratın hər biri üzərində belə bölgü əməliyyatı ardıcıl təkrar olunur. Belə təsvirlərə *fraktallar* deyilir. Fraktallar kompüter qrafikasında geniş tətbiq olunur. Şəkilçəkmə redaktorları vasitəsilə hər hansı bir təsviri çəkdikdə təsvir yaddaşda böyük yer tutur. Həmin təsviri fraktal kimi yaratdıqda isə əməliyyat iterasiyalı alqoritm üzrə yerinə yetirildiyi üçün, çox az yaddaş tələb olunur. Fraktalların qurulma prinsipi ondan ibarətdir ki, obyekt həndəsi ölçülərini kiçildir və yeni koordinatlarla çəkilir.

```
Program Serpin_kv;
Uses CRT,Graph;
Var
   gd,gm:Integer;
   x1,y1,x2,y2,x3,y3:real;

procedure serp(x1,y1,x2,y2:real;n:integer);
var
   x1n,y1n,x2n,y2n:real;

begin
 if  n > 0  then
 begin
   x1n:=2*x1/3+x2/3;
   x2n:=x1/3+2*x2/3;
   y1n:=2*y1/3+y2/3;
   y2n:=y1/3+2*y2/3;
   rectangle(round(x1n),
   round(y1n),round(x2n),
   round(y2n));
   delay(3000);
   serp(x1,y1,x1n,y1n,n-1);
   delay(5000);
   serp(x1n,y1,x2n,y1n,n-1);
   delay(3000);
   serp(x2n,y1,x2,y1n,n-1);
   delay(3000);
   serp(x1,y1n,x1n,y2n,n-1);
   delay(3000);
   serp(x2n,y1n,x2,y2n,n-1);
   delay(3000);
   serp(x1,y2n,x1n,y2,n-1);
   delay(3000);
   serp(x1n,y2n,x2n,y2,n-1);
   delay(3000);
   serp(x2n,y2n,x2,y2,n-1);
 end;
end;

Begin
 gd:=detect;
 InitGraph(gd,gm,'c:\bp\bgi');
 rectangle(50,50,400,400);
 Serp(50, 50,400,400,4);
 ReadLn;
 CloseGraph;
End.
```

Burada, `Serp` adlı *rekursiv* alt proqram tərtib edilmiş və əsas proqramdan ona müraciət olunduqda, faktik arqumentlər kimi, kvadratların koordinatları və

dövrlər sayı (*4*) ötürülmüşdür. Kvadratlar `rectangle` proseduru ilə çəkilir. Dövrlər sayını artırdıqda kvadratların ölçüləri kiçiləcək, sayları isə artacaqdır.

Proqramın nəticəsi şəkil 9.8 – də göstərilmişdir.

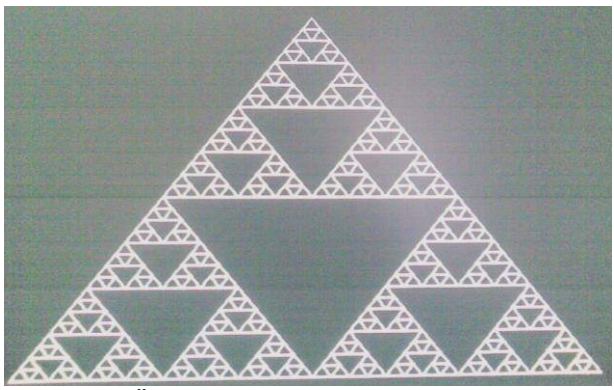


Şəkil 9.8. Serpinski "xalısı"

Misal. Üçbucaqlardan ibarət Serpinski "xalısının" qurulması.

```
Program Serp_ucbucaq;
Uses CRT,Graph;
Var
   gd,gm:Integer;
Const
   iter=5;

Procedure tr(x1,y1,x2,y2,x3,y3:Real);
Begin
  Line(round(x1),round(y1),round(x2),round(y2));
  Line(round(x2),round(y2),round(x3),round(y3));
  Line(round(x3),round(y3),round(x1),round(y1));
End;

Procedure draw(x1,y1,x2,y2,x3,y3:Real;n:Integer);
Var
   x1n,y1n,x2n,y2n,x3n,y3n:Real;
Begin
  If  n > 0  then
    Begin
      x1n:=(x1+x2)/2;
      y1n:=(y1+y2)/2;
      x2n:=(x2+x3)/2;
      y2n:=(y2+y3)/2;
      x3n:=(x3+x1)/2;
```

```
      y3n:=(y3+y1)/2;
      delay(30000);
      tr(x1n,y1n,x2n,y2n,x3n,y3n);
      delay(3000);
      draw(x1,y1,x1n,y1n,x3n,y3n,n-1);
      delay(3000);
      draw(x2,y2,x1n,y1n,x2n,y2n,n-1);
      delay(3000);
      draw(x3,y3,x2n,y2n,x3n,y3n,n-1);
    End;
End;

Begin
  gd =Detect;
  InitGraph(gd,gm,'');
  tr(320,80,600,400,40,400);
  draw(320,80,600,400,40,400,iter);
  Readln;
  CloseGraph;
End.
```



Şəkil 9.9. Üçbucaqlardan ibarət Serpinski "xalısı"

Bu proqram əvvəlki proqrama tamamilə analojidir, yalnız kvadratlar əvəzinə üçbucaqların qurulması ilə ondan fərqlənir. Proqramın nəticəsi şəkil 9.9–da göstərilmişdir.

Misal. Üçbucaqlardan ibarət Serpinski "xalısının" nöqtələrlə qurulması.

```
Program Serp_ucbucaq2R;
Uses CRT,Graph;
Var
   gd,gm:Integer;
   l,x,y:Real;
Begin
  gd:=Detect;
```

```
  InitGraph(gd,gm,'c:\bp\bgi');
  x:=0; y:=0;
  Randomize;
  While not Keypressed Do
    Begin
      l:=2/3*pi*random(3);
      x:=x/2+cos(l);
      y:=y/2+sin(l);
      PutPixel(320+Round(x*110),255+Round(y*110),15);
    End;
  Readln;
  CloseGraph;
End.
```

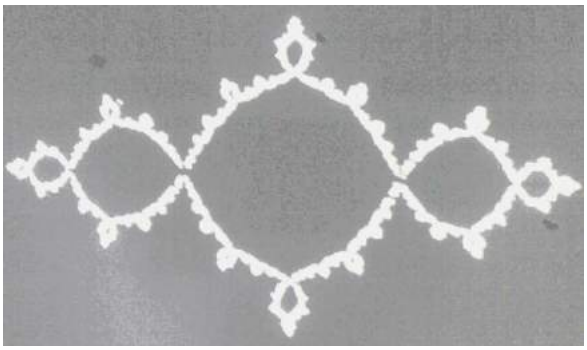
Bu proqramda ilkin şərtli dövr təşkil olunur, dövrün yerinə yetirilmə şərti "*heç bir klaviş basılmadıqda*" (`While not Keypressed Do`) şərtidir. Dövr daxilində `x` və `y` koordinatları sinus və kosinus funksiyaları vasitəsilə hesablanır və onlar yuvarlaqlaşdırılaraq (`Round` funksiyası) `PutPixel` prosedurunun arqumentləri kimi istifadə edilir. Bu prosedur ağ rənglə (*15* kodu) nöqtələr çəkir.

Misal. Naxışlı fraktallar.

```
program serp_ornament;
uses crt,graph,graphs;
var d,rt:integer;
    i:longint;
    wx,wy,cx,cy,x,y,m,n,r,theta:real;
begin
  Open_graph;
  x:=GetMaxX; { ekranın eni }
  y:=GetMaxY; { ekranın hündürlüyü }
  for i:=1 to 100000 do { iterasiyaların sayı }
    begin
      cx:=-1;cy:=0; { görünüş sabitləri }
      wx:=x-cx;
      wy:=y-cy;
      if wx>0 then theta:=arctan(wy/wx);
      if wx<0 then theta:=3.14159+arctan(wy/wx);
      if wx=0 then theta:=1.57079; {pi/2}
      theta:=theta/2;
      r:=sqrt(wx*wx+wy*wy);
      if Random<0.5
        then
          r:=sqrt(r)
        else
          r:=-sqrt(r);
      x:=r*cos(theta);
      y:=r*sin(theta);
```

```
      m:=-5+(x)*100+GetMaxX/2;
      n:=(y)*100+GetMaxY/2;
      PutPixel(trunc(m),trunc(n),White);
    end;
  readln;
  Closegraph;
end.
```

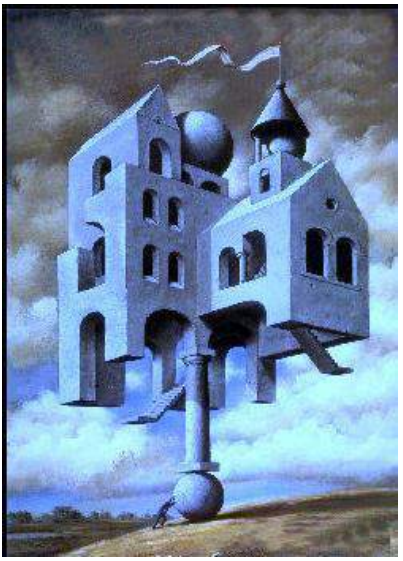
Proqramın nəticəsi şəkil 9.10 – da göstərilmişdir. Bu proqramın da prinsipi



Şəkil 9.10. Naxışlı fraktallar

Serpinski "xalısının" qurulması prinsipi ilə eynidir. Burada, kvadrat və ya üçbucaq əvəzinə qeyri–müəyyən formalı fiqur götürülmüşdür ki, o da ölçülərini kiçildib koordinatlarını dəyişdikdə maraqlı naxış əmələ gətirir. Burada da `x` və `y` koordinatları sinus və kosinus funksiyaları vasitəsilə hesablanır. Naxış ağ rəngli (`White`) nöqtələr (`PutPixel` proseduru) vasitəsilə qurulur. Nöqtələrin koordinatlarının tam ədədlər olması üçün `trunc` funksiyasından istifadə edilmişdir. Əgər naxışın qurulması prosesini izləmək istəsəniz, onda `PutPixel` prosedurundan əvvəlki sətrə `delay` prosedurunu əlavə edin, lakin onun parametrinə çox böyük qiymətlər verməyin. Çünki, dövr *100000* dəfə təkrar olunur.

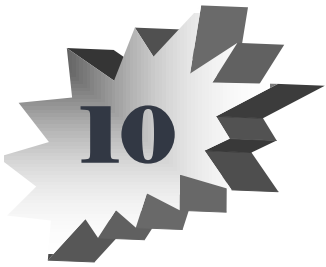
İKİNCİ HİSSƏ



DELPHI

- **Delphi ilə tanışlıq**
- **Object Pascal dili**
- **Komponentlərlə iş**
- **Əlavələrdə formaların yeri**
- **Müstəsna hallar**
- **Qrafiklərlə iş**

Onuncu fəsil



DELPHİ İLƏ TANIŞLIQ

Delphi–də proqramların yaradılması **IDE** (*Integrated Development Environment*) – **İnteqrallaşdırılmış iş mühitində** yerinə yetirilir. Onun köməyi ilə əlavə layihələndirilir, proqram kodları yazılır və sazlanır.

Delphi–nin **IDE** mühiti çoxpəncərəli sistemdən ibarətdir. İnteqrallaşdırılmış iş mühitinin görünüşü onun sazlanmasından asılı olaraq müxtəlif ola bilər. Delphi yükləndikdə ekranda ilkin olaraq dörd pəncərə görünür (şəkil 10.1):

- *Əsas pəncərə* – **Delphi – Project1**;
- *Obyektlər inspektoru pəncərəsi* – **Object Inspector**;
- *Forma konstruktoru pəncərəsi* – **Form1**;
- *Kod redaktoru pəncərəsi* – **Unit1.**

Pəncərələrin çox olmasına baxmayaraq, Delphi bir sənədli mühitdir: o, eyni zamanda yalnız bir əlavə ilə – əlavə layihəsi ilə işləməyə imkan verir. Pəncərənin sərlövhə sətrində layihənin adı **Project1** və bütün Windows pəncərələrinə xas olan pəncərəni idarəetmə düymələri yerləşir. Əsas pəncərəni bükdükdə Delphi interfeysi və onunla birlikdə bütün açıq pəncərələr bükülür; onu bağladıqda isə Delphi ilə iş qurtarır.

Əsas pəncərə aşağıdakı hissələrdən ibarətdir:

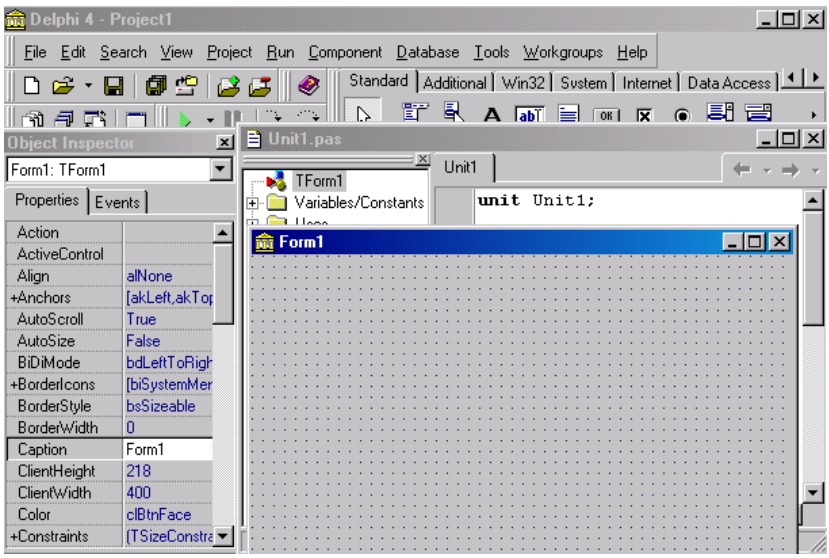
Əsas menyu;
Alətlər paneli;
Komponentlər palitrası.

Əsas menyu Delphi–nin versiyalarından asılı olaraq bir neçə menyudan ibarət olur ki, bu menyularda Delphi–nin müxtəlif funksiyalarını icra etməyə imkan verən çoxlu əmrlər toplanmışdır.

Alətlər paneli əsas menyunun altında, əsas pəncərənin sol hissəsində yerləşir. Bu paneldə yerləşən düymələr, Windows–un bütün pəncərələrində olduğu kimi, daha çox işlədilən əmrləri cəld icra etmək üçündür. Həmin düymələr aşağıdakı 5 növ alətlər panelinə aiddir:

Standart –*Standart*;
View –*Baxış*;
Debug –*Sazlama*;
Custom –*İstifadəçi*;
Desktop –*İş masası.*

Bu panelləri və onlara aid düymələri sazlamaq mümkündür. Bunun üçün mausun göstəricisi alətlər paneli oblastında yerləşdikdə onun sağ düyməsini basmaqla açılan kontekst menyudan istifadə etmək lazımdır.



Şəkil 10.1. Delphi pəncərələri

Komponentlər palitrası əsas menyunun altında, əsas pəncərənin sağ hissəsində yerləşir və yaradılacaq formalarda yerləşdirilən çoxlu komponentlərdən ibarətdir. Komponentlər bir növ qurucu bloklardır və onlardan əlavə forması konstruksiya edilir. Komponentlər qruplaşdırılaraq ayrı–ayrı səhifələrdə yerləşdirilmişdir. Mausun düyməsini səhifənin yarlıkı üzərində basmaqla lazım olan səhifəni açmaq olar.

İlkin olaraq Komponentlər palitrası aşağıdakı səhifələrdən ibarət olur:

`Standard` –*Standart*;
`Additional` –*Əlavə*;
`Win32` –*32 mərtəbəli Windows interfeysi*;

System	*–Sistem funksiyalarına daxilolma*;
Data Access	*–BDE–nin köməyi ilə verilənlər bazasına daxilolma*;
Data Controls	*–Verilənləri idarəetmə elementlərinin yaradılması*;
ADO	*–ActiveX verilənlərinin obyektlərindən istifadə etməklə verilənlər bazası ilə əlaqə*;
Interbase	*–Eyniadlı verilənlər bazasına birbaşa daxilolmanın təmini*;
Midas	*–Paylanmış verilənlər bazası üçün əlavələrin işlənməsi*;
Internet Express	*–Eyni zamanda Web–server əlavəsi və paylanmış verilənlər bazasının müştəri–əlavəsi olan eyniadlı əlavənin yaradılması*;
Internet	*–İnternet şəbəkəsi üçün Web–server əlavəsinin yaradılması*;
FastNet	*–İnternet şəbəkəsinə daxilolma protokolunun təmini;*
Decision Cube	*–Çoxölçülü analiz*;
QReport	*–Hesabatların tərtib edilməsi*;
Dialogs	*–Standart dialoq pəncərələrinin yaradılması;*
Win 3.1	*–Windows 3.x interfeysi*;
Samples	*–Misal nümunələri*;
ActiveX	*–ActiveX komponentləri*;
Servers	*–COM ümumi serveri üçün VCL örtüyü.*

Delphi–nin müxtəlif versiya və konfiqurasiyalarında bu komponentlərin bəziləri olmaya bilər. Komponentlər palitrasını da sazlamaq olar. Bunun üçün *Palette Properties* (*Komponentlər palitrasının xassələri*) dialoq pəncərəsini açmaq lazımdır. Bu pəncərə əsas menyunun *Component/Configure Palette…* (*Komponent/Komponentlər palitrasının sazlanması*) və ya Komponentlər palitrasının kontekst menyusunun *Properties* (*Xassələr*) əmri ilə çağrılır. Onun köməyi ilə komponentləri və habelə onların yerləşdikləri səhifələri pozmaq, əlavə etmək və onların yerlərini dəyişmək mümkündür.

Forma konstruktoru pəncərəsi ilkin olaraq ekranın mərkəzində yerləşir və `Form1` adlı sərlövhədən ibarət olur. Bütün layihələndirmə işləri məhz onun üzərində yerinə yetirilir. Ona görə də hər bir proqramda ən azı bir forma olur. Əgər proqramda bir neçə forma istifadə olunarsa, onda onların sərlövhəsi `Form1`, `Form2` və s. kimi adlardan ibarət olur. Lakin, bu standart adları biz özümüz proqramın mahiyyətinə uyğun daha mənalı adlarla əvəz edə bilərik. Hər hansı bir əlavə yaratdıqda proqramçı Komponentlər palitrasından müvafiq komponentləri (obyektləri) forma üzərində yerləşdirir. Bu obyektlərin forma üzərində düzgün və səliqəli yerləşdirilməsi üçün forma nöqtəli şəbəkədən ibarət olur. Proqram hazır olduqda isə həmin şəbəkə yox olur. Layihələndirmə zamanı Forma konstruktoru "kadr arxasında" qalır, proqramçı isə formanın özü ilə

işləyir və ona görə də Forma konstruktoruna sadəcə olaraq Forma pəncərəsi və ya Forma deyilir.

Kod redaktoru pəncərəsi Forma konstruktoru pəncərəsinin arxasında yerləşir və bu pəncərə tərəfindən demək olar ki, tam örtülür. Pəncərənin sərlövhəsi `Unit1.pas` adlanır; proqramçı bu adı dəyişdirə bilər. Bu pəncərədə "boş" formaya uyğun, Object Pascal dilində yazılmış *modulun kodları* (*yunit*) yerləşir, başqa sözlə, kod redaktoru heç vaxt boş olmur. Çünki, Delphi bizim yerinə yetirəcəyimiz işlərin xeyli hissəsini özü avtomatik olaraq yerinə yetirir. Buna baxmayaraq, biz proqrama yeni operatorlar əlavə etdikdə, onu məhz bu pəncərədə yerinə yetirəcəyik. Bəzi operatorları Delphi bizim xahişimizlə özü əlavə edəcək, digərlərini isə özümüz yazacağıq. Kod redaktoru adi mətn redaktorudur və onun köməyi ilə modulun mətninə və ya digər mətn tipli fayllara düzəlişlər etmək mümkündür.

Forma konstruktoru və Kod redaktoru pəncərələrinin yerini dəyişmək üçün ***F12*** klavişini və ya lazım olan pəncərənin oblastında mausun düyməsini basmaq kifayətdir. Mausun göstəricisini sərlövhə sətrində yerləşdirərək onu hərəkət etdirməklə də pəncərələrin yerlərini dəyişmək mümkündür.

Kod redaktoru pəncərəsinin sol tərəfində daha bir pəncərə – **Kod bələdçisi** pəncərəsi yerləşir. Burada forma modulunun bütün obyektləri, məsələn, dəyişən və prosedurlar ağac şəklində təsvir edilir. Onun köməyi ilə əlavənin obyektlərinə daha rahat baxmaq və lazım olan obyektlərə daha cəld müraciət etmək olar. Xüsusən böyük modullarla işlədikdə Kod bələdçisindən istifadə etmək daha əlverişli olur.

Növbəti vacib pəncərə **Object Inspector** adlanır. Biz bu pəncərəyə **Obyektlər inspektoru** deyəcəyik. O, əsas pəncərənin altında, ekranın sol tərəfində yerləşir. Bu pəncərədə `Form1` formasında yerləşdirilən obyektlərin xassə və hadisələri təsvir etdirilir. Biz bu pəncərədən çox tez–tez istifadə edəcəyik. Forma üzərində yerləşdirilən obyektləri mausla seçdikdə Obyektlər inspektorunda onların xassələri görünəcək və biz bu xassələri öz məqsədimizə uyğun olaraq dəyişdirəcəyik. Pəncərəni bağlama düyməsini basdıqda bu pəncərə bağlanır və ekrandan yox olur. Onu ekranda yenidən göstərmək üçün *View/Object Inspector* əmrini icra etmək və ya ***F11*** klavişini basmaq lazımdır.

Obyektlər inspektoru pəncərəsi iki səhifədən ibarətdir: *Properties* (*Xassələr*) və *Events* (*Hadisələr*).

Properties səhifəsində forma üzərində seçilmiş obyektin *xassələri* haqqında informasiya təsvir olunur və qeyd etdiyimiz kimi, layihələndirmə zamanı bir sıra xassələri çox asanlıqla dəyişdirə bilərik.

Events səhifəsi göstərilən *hadisə* baş verdikdə komponentin yerinə yetirəcəyi proseduru müəyyən edir. Əgər bu və ya digər hadisə üçün prosedur müəyyənləşdirilmişdirsə, onda layihənin yerinə yetirilməsi prosesində bu hadisə baş verdikdə həmin prosedur avtomatik olaraq icra olunacaqdır. Bu prosedurlar hadisələri emal etdiyi üçün onlara *prosedur–emaledicilər* və ya sadəcə olaraq

emaledicilər deyilir. Belə hadisələrə misal olaraq mausun düyməsinin bir və ya iki dəfə basılması zamanı baş verəcək əməliyyatları (formanın bağlanması, sərlövhənin dəyişdirilməsi və s.) göstərmək olar.

10.1. Layihənin kompilyasiyası və yerinə yetirilməsi

Delphi layihəni kompilyasiya etdikdə susmaya görə onun fayllarını *C:\Program Files\Delphi\Bin* qovluğunda yerləşdirir. Ona görə də proqramı kompilyasiya etməzdən əvvəl, yeni qovluq yaradıb kompilyasiya olunmuş faylları orada saxlamaq lazımdır. Kompilyasiya nəticəsində *8* fayl yarandığı üçün hər bir yeni layihəni ayrı–ayrı qovluqlarda saxlamaq məqsədəuyğundur. İndi isə Delphi–də ilk proqramımızı yaradaq. Bunun üçün heç bir proqram kodu yazmadan, sadəcə olaraq, *Run/Run* əmrini və ya alətlər panelindəki *Run* düyməsini (yaşıl rəngli düymə) və yaxud da ***F9*** klavişini basın. Kompilyasiyanın nəticəsi olaraq ekranda nöqtələrdən ibarət şəbəkəsi olmayan "boş" forma görəcəksiniz. Bu forma Delphi yükləndikdə ekranda görünən forma deyil, Windows pəncərələrinin malik olduğu idarəedici elementlərdən ibarət mükəmməl bir pəncərədir. Bu pəncərə heç bir əməliyyat yerinə yetirməsə də, öz növbəsində Windows sisteminin bütün standart əmrlərini icra edir. İndi isə *File* menyusundan *SaveAll* əmrini icra etməklə layihəni yaratdığınız qovluqda yadda saxlayın. Bu məqsədlə Sizə, əvvəlcə yuniti – `Unit1.pas`, sonra isə layihəni – `Project1.dpr` adı altında saxlamaq təklif olunacaqdır. Bundan sonra isə Siz qovluğa baxdıqda görəcəksiniz ki, orada *2* yox *6* fayl saxlanmışdır. İndi isə *Project* menyusundan *Compile* əmrini icra edin və ya ***Ctrl+F9*** klavişlərini basın. Delphi layihəni icra olunan fayla kompilyasiya edəcəkdir və görəcəksiniz ki, Sizin qovluğunuza daha *2* fayl (`Project1.exe` və `Unit1.dcu`) əlavə edilmişdir. Layihənin tərkibinə daxil olan əsas fayllar aşağıdakılardır (mötərizədə faylların tipi göstərilmişdir):

- **Layihə kodu** (*.dpr*);
- **Formanın təsviri** (*.dfm*);
- **Forma modulu** (*.pas*);
- **Modullar** (*.pas*);
- **Layihənin parametrləri** (*.opt*);
- **Resursların təsviri** (*.res*).

Bu fayllardan başqa digər fayllar da, məsələn, onların ehtiyat surətləri yarana bilər: *~DP – DPR* fayllar üçün, *~PA – PAS* fayllar üçün. Bu faylların xarakteristikaları ilə bir az sonra tanış olacağıq. İndi isə yenidən kompilyasiyaya qayıdaq.

Beləliklə, layihənin yerinə yetirilməsi *Run/Run* (*Yerinə yetirmək/Yerinə yetirmək*) əmri ilə və ya ***F9*** klavişini basmaqla başlayır. Yaradılmış əlavə öz işinə başlayır. Əgər proqramda səhvlər varsa, bu barədə məlumat verilir. Əlavənin surətlərini işə salmaq olmaz. Əgər sonsuz dövr etmə halları baş

verərsə, onda proqramın işini *Run/Program Reset* (*Yerinə yetirmək/Proqramın dayandırılması*) əmri ilə və ya ***Ctrl+F2*** klavişlərini basmaqla dayandırmaq olar.

Kompilyasiya prosesi *Project/Compile <Project1>* (*Layihə/Kompilyasiya <Layihə1>*) əmri ilə və ya ***Ctrl+F9*** klavişlərini basmaqla başlayır. Bu əmrdə layihənin adı yazılır (ilk dəfə `Project1` adı ilə). Layihəni başqa ad ilə yadda saxladıqda uyğun olaraq bu əmrdə də layihənin adı dəyişməlidir.

Layihələndirmənin istənilən mərhələsində proqramı kompilyasiya etmək olar. Bu, forma üzərində yerləşidirilmiş komponentlərin görünüşünü və onların funksiyalarının düzgün yerinə yetirilməsini yoxlamaq üçün çox əlverişlidir. Kompilyasiya zamanı bütün modul faylları kompilyasiya olunur və nəticədə hər bir fayl üçün modulun ilkin mətnindən ibarət *.dcu* tipli fayl yaranır. Layihənin bütün modulları kompilyasiya edildikdən sonra layihə faylı kompilyasiya edilir. Nəticədə layihə ilə eyniadlı icra olunan əlavə faylı yaranır (*.exe* tipli).

Tam funksiyalı əlavə bütün fayllardan yaranır. Ona görə də kompilyasiyadan başqa, layihənin *yığılması* yerinə yetirilir. Bunun üçün *Project/Build <Project1>* (*Layihə/Yığmaq <Layihə1>*) əmri icra olunmalıdır.

Kompilyasiya nəticəsində icra üçün tam hazır olan, müstəqil *.exe* tipli fayl yaranır ki, bu fayl Windows sistemi altında işləyən bütün əlavələr kimi yüklənir və bu zaman kompüterin proqram təminatında Delphi proqramlaşdırma mühitinin olması vacib deyildir.

Beləliklə, biz Delphi–də ilk layihəni yaratdıq. Bu layihənin ən maraqlı cəhəti ondan ibarət oldu ki, biz "heç nə etmədən" onu yaratdıq. Buna isə səbəb ondan ibarətdir ki, Delphi hər bir layihə üçün proqramçıya boş forma təklif edir. Əslində isə bu forma boş deyildir. O, Windows pəncərələrinin əsas elementlərindən, `Form1` sərlövhəsindən, pəncərəni bükmə, ekran boyu açma, bağlama, onun ölçülərini dəyişmə əmrlərindən və sistem menyusu düymələrindən ibarətdir.

Windows sisteminin istənilən əlavəsi müvafiq pəncərədə yerinə yetirildiyi üçün, hətta o boş olsa belə, yenə də pəncərəyə malik olmalıdır. Delphi də Windows altında işləyən əlavələr yaratdığı üçün, hər bir layihəyə forma ayırır ki, bu forma üçün artıq təsvir və moduldan ibarət iki fayl yaradılmışdır. Sadə əlavə hər bir əlavənin yerinə yetirdiyi ən zəruri əməliyyatları təmin edən karkasdan – boş formadan ibarət olur. Proqramçıya klaviatura emaledicisi və ya maus drayveri yazmaq, pəncərələrlə işləmək üçün prosedurlar yaratmaq lazım gəlmir. Bütün bu işləri onun əvəzindən Delphi–nin yaradıcıları etmişlər. Əlavənin karkası tam yekunlaşmış, bitkin əlavədir, sadəcə olaraq heç nə yerinə yetirmir. "Heç nə yerinə yetirmir" fikrini iki mənada başa düşmək olar. Pəncərə və onunla birlikdə əlavə doğrudan da heç nə yerinə yetirmir, çünki, heç bir funksiyaya malik deyildir. Digər tərəfdən bu boş forma çox işlər görür: məsələn, istifadəçidən klaviatura və mausla əlaqədar əməllər gözləyir, özünün ölçülərinin, yerlərinin dəyişdirilməsi və s. kimi əmrlərə reaksiya verir. Windows–da əlavələr yaradan proqramçı çox yaxşı bilir ki, bu əlavələr hansı çətinliklərlə yaranır, yəni bir çox mürəkkəb elementləri (indekslər, kontekstlər, əks əlaqələr sistemi və s.) proqramçı özü yaradır. Bütün bu mürəkkəb işləri

Delphi öz üzərinə götürür. Onun gördüyü işlər haqqında boş formadan ibarət layihənin neçə bayt yer tutması (*~275 Kb*) ilə fikir yürütmək olar.

10.2. Layihənin xarakteristikaları

10.2.1. Layihə faylı

Layihə faylı ən əsas fayldır və əslində elə proqramın özüdür. Bir formadan ibarət əlavələr üçün layihə faylı aşağıdakı kodlardan ibarət olur:

```
program Project1;

uses
   Forms,
   Unit1 in 'Unit1.pas' {Form1};

{$R*.Res}

Begin

   Application.Initialize;
   Application.CreateForm(TForm1, Form1);
   Application.Run;

end.
```

Layihənin (proqramın) adı layihə faylının adı ilə eyni olur və diskdə saxladıqda ona ad vermək lazımdır (susmaya görə layihənin adı `Project1` olur). Layihənin resurs və parametrlər faylları da onunla eyniadlı olur, layihə faylının adını dəyişdikdə digər faylların da adları avtomatik olaraq dəyişir.

Layihə faylının yuxarıda göstərilən məzmunlu proqram kodlarını Delphi özü yazmışdır. Əksər hallarda proqramçının bu fayla müdaxilə etməsinə ehtiyac olmur. Lakin, bəzi hallarda proqramçıya bu kodlara yeni sətirlər əlavə etmək lazım gəlir. Bu fayla baxmaq və ya ona düzəlişlər etmək üçün Kod redaktoru pəncərəsində *Project/View Source* (*Layihə/Mənbəyə baxış*) əmrini icra etmək lazımdır.

Bütün layihənin yığılması layihə faylı kompilyasiya edildikdə baş verir. Bu zaman yaranan əlavənin (*.exe* faylın) adı layihə faylının adı ilə eyni olur.

Layihə faylının `Uses` bölməsində `Forms` modulunun adı yazılmışdır. Bu modul tərkibində forma olan bütün əlavələr üçün vacibdir. Bundan başqa, bu bölmədə bütün layihə formaları modullarının adları sadalanır – ilkin olaraq bu, `Form1` formasının `Unit1` moduludur.

`$R` direktivi layihəyə resurslar faylını qoşmaq üçündür. Susmaya görə bu faylın adı layihə faylının adı ilə eyni olur. Ona görə də resurs faylının adı əvəzinə "*" simvolu göstərilmişdir. Proqramçı `$R` direktivini əlavə etməklə və faylın adını göstərməklə başqa resursları layihəyə qoşa bilər.

Layihə proqramı isə cəmi üç operatordan ibarətdir. Bu operatorlar əslində aşağıdakı metodları çağırır:

`Application.Initialize` – bütün Delphi əlavələrində ən birinci çağrılan metoddur: o, *OLE* və digər alt sistemləri yoxlayır, əlavəni inisializasiya edir;

`Application.CreateForm(TForm1, Form1)` – bütün zəruri elementləri ilə birlikdə `Form1` formasını yaradır;

`Application.Run` – əlavəni işə buraxır.

Proqramçı layihədə hər hansı bir əməliyyatı yerinə yetirdikdə Delphi layihə faylının kodunu avtomatik olaraq dəyişir. Məsələn, layihəyə yeni forma əlavə edildikdə layihə faylının koduna iki yeni sətir əlavə olunur, formanı layihədən çıxardıqda isə bu sətirlər avtomatik olaraq pozulur.

Əksər əlavələr üçün layihənin faylı eyni və ya oxşar koda malik olduğu üçün, biz gələcəkdə məsələlər həll etdikdə, bu faylın məzmununu göstərməyəcəyik.

10.2.2. Forma modulu faylı

Bu faylda formanın siniflərinin təsviri yerləşir. Susmaya görə bu faylın adı `Unit1.pas` olur. Faylın birinci sətri `unit` işçi sözü ilə başlayır. Bütün modulların da birinci sətri bu işçi sözlə başlayır. Deməli, bu faylda proqram modulu yerləşir və ona *yunit* də deyirlər. Əslində modul layihədə yeganə proqram vahididir ki, o, məhz proqramçının özü tərəfindən yaradılır, başqa sözlə, layihəçinin bütün yaradıcı fəaliyyəti məhz özünü bu modulda əks etdirir. Lakin, bu o demək deyildir ki, proqramçı modulun bütün kodlarını özü yazır. Burada da Delphi öz köməyini əsirgəmir: modulun strukturu Delphi tərəfindən artıq hazır şəkildə proqramçıya təqdim olunur, hadisələrə uyğun bir neçə sətri də Delphi özü yazır və modulda hadisə emaledicisinin yerinə yetirəcəyi əməliyyatlara aid kodların əlavə ediləcəyi yeri də göstərir. Proqramçı, modulun Delphi tərəfindən mətn kursoru ilə göstərilən hissəsinə müvafiq proqram kodları əlavə edir. Ümumiyyətlə, layihələndirmə (proqramlaşdırma) işləri forma üzərində və yunitdə yerinə yetirilir.

İndi isə boş forma üçün *forma modulu* faylının məzmununa baxaq:

```
unit Unit1;

interface

uses
   Windows, Messages, SysUtils, Classes, Graphics,
   Controls, Forms, Dialogs, StdCtrls;
```

```
type
   TForm1=class(TForm)

   Private
   {private declarations}

   public
   {public declarations}

   end;

var
   Form1:TForm1;

implementation

{$R*.DFM}

end.
```

Biz bu bölmədə modulun strukturunda göstərilən bölmə və operatorları izah etməyəcəyik; onların müfəssəl izahı Object Pascal dilinin "*Modullar*" bölməsində veriləcəkdir.

Yunitin mətni Kod redaktoru pəncərəsində təsvir olunur. Pəncərəni bağlama düyməsini basdıqda modul ekrandan yox olur. Onu yenidən ekranda təsvir etmək üçün *File/Open* (*Fayl/Açmaq*) əmrini icra etmək lazımdır. Bu modulu ***Ctrl+F12*** klavişlərini basmaqla və ya *View/Units…* (*Baxış/Modullar…*) əmrini icra etməklə də ekranda təsvir etmək olar. Sonuncu əmr icra olunduqda ekranda *View Unit* (*Modula baxış*) dialoq pəncərəsi peyda olur ki, bu pəncərədə `Unit1` seçərək *Ok* düyməsini basmaq lazımdır. Yeri gəlmişkən qeyd edək ki, elə bu pəncərədən `Project1` seçməklə layihə faylının mətnini də ekranda təsvir etmək olar.

Modul kompilyasiya edildikdə, avtomatik olaraq, *.dcu* tipli fayl da yaranır. Bu, kompilyasiya edilmiş modul kodlarından ibarət fayldır. Bu faylı pozmaq olar, lakin hər dəfə modul kompilyasiya edildikdə o, yenidən yaranacaqdır.

10.2.3. Forma faylı

Hər bir forma üçün layihənin tərkibində, avtomatik olaraq, formanın təsviri faylı yaranır (*.dfm* tipli). Bu faylın adı da yunitin adı ilə eyni olur. Forma faylına əl ilə və ya hər hansı bir üsulla düzəlişlər etmək məsləhət görülmür. Bununla, Siz, yalnız faylı korlaya bilərsiniz.

Formanın təsviri faylı – Delphi–nin resursudur. Bu faylda formanın özü haqqında informasiya təsvir olunur: o, hansı ölçüdədir, ekranda hansı vəziyyətdədir, onun üzərində hansı komponentlər vardır və bu komponentlərin

xarakteristikaları necədir. Proqramçı Forma konstruktorunda obyektləri yerləşdirdikdə və Obyektlər inspektorunda həmin komponentlərin xassələrini müəyyənləşdirdikdə bu məlumatlar avtomatik olaraq forma faylında yadda saxlanır. Layihəni yadda saxladıqdan sonra, zərurət yaranarsa, bu faylın məzmununu ekrana çıxarmaq olar. Bunun üçün əvvəlcə Forma konstruktoru pəncərəsində bu faylın təsvirinə uyğun formanı bağlamaq, sonra isə *File/Open* (*Fayl/Açmaq*) əmrini icra etmək lazımdır. Forma konstruktoru pəncərəsini yenidən ekranda təsvir etmək üçün, *File/Close* (*Fayl/Bağlamaq*) əmri ilə forma faylını bağladıqdan sonra, *View/Forms* (*Baxış/Formalar*) əmrini və ya ***Shift+F12*** klavişlərini basmaq lazımdır. İndi isə nümunə üçün, üzərində `Button1` düyməsi yerləşdirilmiş `Form1` forması üçün, forma faylının mətninə baxaq (bu nümunədə həmin düymə üçün `OnClick` – düyməbasma hadisə emaledicisi də yazılmışdır):

```
Object Form1:TForm1
   Left=192
   Top=107
   Width=544
   Height=375
   Caption='Form1'
   Color=clBtnFace
   Font.Charset=DEFAULT_CHARSET
   Font.Color=clWindowText
   Font.Height=-11
   Font.Name='MS Sans Serif'
   Font.Style=[ ]
   OldCreatOrder=False
   PixelsPerInch=13
Text.Height=13
Object Button1:TButton1
   Left=88
   Top=120
   Width=75
   Height=25
   Caption='Button1'
   TabOrder=0
   OnClick=Button1Click
   end
end
```

Qeyd edək ki, formaya digər komponentlər əlavə edildikdə bu fayla onun xarakteristikaları haqqında məlumat əlavə ediləcəkdir. Bu nümunədən görünür ki, forma faylında komponentlərin adları göstərilir. Onların tipləri (sinifləri) isə forma modulunda (yunitdə) təsvir olunur. Əgər bu faylın `Caption='Button1'` sətrində, yəni düymənin sərlövhəsini müəyyənetmə sətrində `'Button1'` əvəzinə başqa mətn yazsaq (məsələn, `'Açmaq'`), onda düymənin üzərindəki yazı bu mətnlə əvəz olunacaqdır. Lakin, yuxarıda qeyd

etdiyimiz kimi, bu faylda dəyişikliklərin edilməsi məsləhət görülmür, belə dəyişiklər adətən Obyektlər inspektorunda yerinə yetirilir.

10.2.4. Resurslar faylı

Layihəni birinci dəfə yadda saxladıqda, avtomatik olaraq, tipi *.res* olan resurslar faylı yaranır. Bu faylın adı da layihə faylının adı ilə eyni olur (`Project1.res`). Resurslar faylında piktoqramlar, kursorlar və s. kimi resurslar saxlanır. İlkin olaraq bu faylda, susmaya görə, məşəl təsvirindən ibarət layihə piktoqramı yerləşir. Sonralar isə bu piktoqramı dəyişmək olar. Bunun üçün *Tools/Image Editor* (*Servis/Təsvirlər redaktoru*) əmri ilə şəkilçəkmə redaktorunu çağıraraq bu redaktorda `Project1.res` faylını açmaq lazımdır.

Layihə kompilyasiya edildikdə *.dof* tipli daha bir fayl da yaranır ki, bu faylda layihə haqqında əlavə informasiyalar: versiya, müəllif hüququ və s. saxlanır. Bu faylın pozulması layihəyə heç bir xələl gətirmir.

10.2.5. Layihə parametrləri faylı

Bu fayl, *Project/Options…* (*Layihə/Parametrlər…*) əmri ilə açılan dialoq pəncərəsində sazlanan parametrlərin əsasında Delphi tərəfindən avtomatik olaraq yaradılır. Dialoq pəncərəsinin *Forms* və *Application* səhifələrinin parametrləri həm layihə, həm də resurslar faylına, *Compiler* və *Linker* səhifələrinin parametrləri isə layihə parametrləri faylına yazılır. Bu faylın adı, susmaya görə, `Project1.opt` olur. Nümunə üçün layihə parametrləri faylının aşağıdakı fraqmentini göstərmək olar:

```
[Compiler]
A=1
B=0
C=1
D=1
E=0
…
```

10.2.6. Modul faylları

Yuxarıdakı fayllardan fərqli olaraq, modul faylları Delphi tərəfindən avtomatik olaraq yaradılmır və ümumiyyətlə, bu fayl olmaya da bilər. Bu modulların forma ilə heç bir əlaqəsi olmur, hər hansı bir xüsusi məsələnin həlli üçün Object Pascal dilində tərtib edilərək ayrı–ayrı fayllarda saxlanır. Onu layihəyə qoşmaq üçün forma modulunun `Uses` bölməsində həmin modulun adını göstərmək lazımdır.

Ümumiyyətlə, layihənin bir neçə modulları tərəfindən istifadə olunan prosedur, funksiya və verilənləri ayrı bir modulda yerləşdirmək məqsədəuyğundur.

10.3. Əlavə interfeysi

Komponentlər palitrasından seçilən və forma üzərində yerləşdirilən komponentlər əlavə interfeysini təşkil edir, komponentlər özləri isə bir növ qurucu bloklar olur. Proqramçı əlavə interfeysini konstruksiya etdikdə komponentləri forma üzərində yerləşdirir və əlavə yerinə yetirildikdən sonra, o, hansı görünüşdə olacaqdırsa, konstruksiyaetmə zamanı demək olar ki, onu elə o cür görür.

Delphi–də vizual (görünən) və qeyri–vizual (sistem) komponentlər mövcuddur. Bu layihənin yerinə yetirilmə mərhələsində belədir, əlavə layihələndirildikdə isə bütün komponentlər görünür.

Vizual komponentlərə düymələr, siyahılar, dəyişdiricilər, formalar və s. aiddir. Bu komponentlərə idarəedici komponentlər və ya idarəetmə elementləri deyilir. Məhz vizual komponentlər əlavə interfeysini yaradır.

Qeyri–vizual komponentlərə vacib, lakin köməkçi əməliyyatlar yerinə yetirən komponentlər, məsələn, `Timer` saniyəölçəni və ya `Table` verilənlər yığımı aiddir.

Əlavə interfeysi yaradıldıqda hər komponent üçün aşağıdakı əməliyyatlar yerinə yetirilir:

1. *Komponentlər palitrasından komponentlərin seçilməsi və onların forma üzərində yerləşdirilməsi*;
2. *Komponentin xassəsinin dəyişdirilməsi.*

Proqramçı bu əməliyyatları Forma konstruktorunda Komponentlər palitrası və Obyektlər inspektoru vasitəsilə yerinə yetirir. Əlavə interfeysinin yaradılması prosesi ənənəvi proqramlaşdırmadan daha çox konstruksiyaetmə işlərinə oxşayır. Ona görə də əlavənin yaradılması prosesi proqramlaşdırma yox, *konstruksiyalaşdırma* adlanır.

10.3.1. Komponentlər palitrası və forma

Delphi–nin əsas pəncərəsindəki alətlər panelindən biri digərlərindən nəzərə çarpacaq dərəcədə fərqlənir. Bu Komponentlər palitrasıdır (şəkil 10.2). Bu paneldə müxtəlif idarəedici elementlər – proqramın yığıldığı "kərpiclər" toplanmışdır. Həmin elementlər Komponentlər palitrasının müxtəlif səhifələrində (Delphi ilə ilk tanışlıqda bu səhifələrin adlarını sadaladıq) yerləşir.

Şəkil 10.2. Komponentlər palitrası

Mausun düyməsini səhifənin yarlıkı üzərində basdıqda bu səhifədə toplanmış komponentlər palitrada təsvir olunur. Mausun göstəricisini komponentin üzərində bir az ləngitdikdə onun adı peyda olur. Delphi–də yüzdən çox komponentlər mövcuddur. İndiyədək Delphi–yə aid elə bir ədəbiyyat yoxdur ki, orada bütün bu komponentlər tam əhatə edilmiş olsun. Biz də yazacağımız proqramlarda onların ən vaciblərini öyrənəcəyik.

Komponentlər palitrasından komponenti seçmək üçün onun piktoqramı üzərində mausun düyməsini basmaq lazımdır. Bu zaman onun piktoqramı çökdürülmüş (sıxılmış) vəziyyətdə olur. Bundan sonra, formanın boş sahəsində mausun düyməsini basdıqda, onun üzərində komponentin piktoqramı peyda olur, Palitrada isə komponentin piktoqramı adi görkəm alır. Komponentin piktoqramı üzərində mausun düyməsini iki dəfə basdıqda komponent nəinki seçilir, hətta o, avtomatik olaraq formanın orta hissəsində yerləşdirilir. Hər hansı komponenti seçdikdən sonra, seçməni ləğv etmək üçün, Palitranın sol tərəfindəki ox təsvirli nişan üzərində mausun düyməsini basmaq lazımdır.

Forma üzərində bir neçə eyni komponent yerləşdirmək üçün Komponentlər palitrasında onu seçməzdən əvvəl, *Shift* klavişini basılı saxlamaq lazımdır. Bu halda, forma üzərində mausun düyməsini basaraq komponenti orada yerləşdirdikdə, Palitrada onun piktoqramı çökdürülmüş vəziyyətdə qalır və mausun düyməsini növbəti dəfə basdıqda həmin komponent təkrarən forma üzərində yerləşdirilir. Komponentin seçilməsini ləğv etmək üçün ya digər komponenti seçmək ya da ox təsvirli nişan üzərində mausun düyməsini basmaq lazımdır.

Delphi–də obyektlərin, o cümlədən, komponentlərin tiplərinin təsvirində **`T`** hərfi göstərilir. Əksər hallarda komponentlərin işarələrində onların adları deyil, tipləri göstərilir. Komponentləri işarə etmək üçün biz onların adlarını istifadə edəcəyik, başqa sözlə `TButton` yox, `Button`, `TEdit` yox, `Edit` yazacağıq.

Komponenti forma üzərində yerləşdirdikdən sonra, Delphi avtomatik olaraq, modul faylı və təsvirlər faylına dəyişikliklər edir. Hər yeni komponent üçün modul faylına (yunitə) belə formatlı sətir əlavə edilir:

```
Komponentin adı: komponentin tipi;
```

Məsələn, `Button` düyməsi və `Edit` birsətirli mətn redaktoru üçün bu sətir

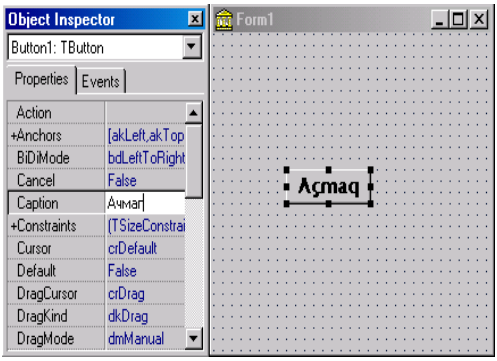
```
Button1: TButton;
Edit1: TEdit;
```

kimi olacaqdır. Əgər forma üzərində bir neçə eyni komponent yerləşdirilərsə, onlar ardıcıl olaraq nömrələnir:

```
Button1: TButton;
Button2: TButton;
Button3: TButton;
```

`Button` düyməsi üçün təsvirlər faylına avtomatik olaraq belə kod yazıla bilər:

```
Object Button1: TButton
   Left=88
   Top=120
   Width=75
   Height=25
   Caption='Button1'
   TabOrder=0
End
```



Şəkil 10.3. Komponentin seçilməsi və onun xassələrinə müraciət

Bu kodda düymənin koordinatları, ölçüləri, sərlövhəsi və fokus almaq xassəsi təsvir olunmuşdur. Forma üzərində komponentin yerini dəyişdikdə – `Left` və `Top` xassələrinin qiyməti, düymənin özünü böyütdükdə və ya kiçiltdikdə isə onun `Width` və `Height` xassələri və s. dəyişəcəkdir.

İndi isə formanın xüsusiyyətlərini öyrənək.

Biz artıq bilirik ki, forma gələcək layihənin karkasıdır və biz proqramlaşdırmadan əvvəl forma ilə işləyirik. Proqram hazır olduqda və onu işə buraxdıqda həmin bu forma mükəmməl pəncərəyə çevriləcəkdir. Hər bir proqramın ən azı bir pəncərəsi, deməli forması olmalıdır.

Forma üzərində yerləşdirilmiş komponent tərəflər və bucaqlar üzrə kvadrat şəkilli markerlərlə qeyd olunur (şəkil 10.3). Forma üzərində istənilən komponenti seçdikdə də belə markerlər əmələ gəlir. Komponenti seçmək üçün onun oblastında mausun düyməsini basmaq kifayətdir. Komponenti seçdikdən sonra, onun yerini və ölçülərini dəyişdirmək olar. Komponentin ölçüsünü dəyişdirmək üçün mausun göstəricisini marker üzərində yerləşdirib, ikitərəfli oxun əmələ gəlməsinə nail olduqdan sonra, mausun sol düyməsini basaraq onu hərəkət etdirmək lazımdır. Komponentin üfqi və ya şaquli ölçülərini dəyişdirdikdə tərəflər üzərindəki markerlərdən, komponentin ölçülərini mütənasib dəyişdikdə isə künclərdəki markerlərdən istifadə etmək lazımdır. Forma üzərində komponentin yerini dəyişdirdikdə isə mausun göstəricisini komponentin daxilində yerləşdirib mausun sol düyməsini basılı saxlayaraq onu hərəkət etdirmək lazımdır. Bundan başqa, forma üzərində bir neçə komponenti düzləndirmək və ya bu və ya digər komponenti ön və ya arxa plana keçirmək olar. Bütün bu əməliyyatlar şəkilçəkmə redaktorundakı əməliyyatları xatırladır. Eyni zamanda bir neçə komponenti seçmək üçün, *Shift* klavişini basılı saxlayaraq, hər bir komponent üzərində mausun sol düyməsini basmaq lazımdır.

Susmaya görə, forma üzərində komponentlər nöqtəli şəbəkə xətlərinə görə düzləndirilir. Bu inteqrallaşdırılmış iş mühitinin parametrlərində *Snap to Grid* (*Şəbəkəyə görə düzləndirmə*) parametri qarşısında bayraq (✓ işarəsi) qoymaqla

müəyyənləşdirilir. Susmaya görə şəbəkənin addımı (nöqtələr arasındakı məsafə) *8* pikselə bərabərdir və layihələndirmə zamanı formanın səthində şəbəkə həmişə təsvir olunur. Şəbəkəyə görə düzləndirmə, şəbəkənin təsviri (*Display Grid–Şəbəkənin təsviri bayrağı*) və üfqi və şaquli şəbəkə addımları *Environment Options* (*Mühitin parametrləri*) dialoq pəncərəsinin *Preferences* (*Üstünlüklər*) səhifəsində müəyyənləşdirilir. Bu pəncərəni çağırmaq üçün *Tools/Environment* (*Servis/Mühitin parametrləri*) əmrini icra etmək lazımdır.

10.3.2. Obyektlər inspektoru

Obyektlər inspektoru forma və onun üzərində yerləşdirilmiş komponentlərin xassə və hadisələrini müəyyən etmək üçündür. Forma üzərində komponent seçildikdə və o, markerlərlə əhatə olunduqda Obyektlər inspektorunun birinci sətrində həmin komponentin adı və tipi (məsələn, `Button1:TButton`, şəkil 10.3), növbəti sətirlərdə isə bu komponentə xas olan xassələr təsvir olunur. Layihəçi həmin xassələrin qarşısında onlara qiymət verir və ya bu qiymətləri təklif olunan variantlardan seçir. Formanın özünün xassələri də analoji qaydada müəyyənləşdirilir. Lakin forma seçilmiş vəziyyətdə olduqda markerlərlə əhatələnmir. Ona görə də formanı seçmək üçün onun komponentlər olmayan boş sahəsində mausun sol düyməsini basmaq kifayətdir. Bu və ya digər komponenti Obyektlər inspektorunun birinci sətrində yerləşən açılan siyahıdan da seçmək və onun xassələrinə müraciət etmək olar. Belə seçmə xüsusən o vaxt zəruri olur ki, bir komponent o biriləri tərəfindən tam örtülmüş vəziyyətdə olur və görünmür.

Obyektlər inspektorunun sol tərəfində (*Properties* səhifəsində) komponentin bütün xassələrinin adları göstərilir. Bu xassələrə əlavənin işlənmə mərhələsində müraciət olunur. Hər bir xassənin sağ tərəfində həmin xassənin qiyməti yerləşir. Bu xassələrdən başqa, komponentin elə xassələri də ola bilər ki, onlara yalnız əlavə yerinə yetirildiyi zaman müraciət oluna bilər.

Xassə əlavə yerinə yetirildikdə komponentin əks olunması və fəaliyyətini müəyyənləşdirən atributlardan ibarətdir. Obyektlər inspektorunda komponentin xassəsini dəyişdikdə, bu dəyişiklik komponentin özündə əks olunur, yəni elə layihələndirmə prosesində dəyişikliklərin nəticəsi artıq görünür. Məsələn, düymənin `Caption` (Sərlövhə) xassəsinə hər hansı bir ad verdikdə (şəkil 10.3) həmin ad düymənin üzərində əks olunur. Xassəyə qiymət verdikdən və ya onu seçdikdən sonra, ya *Enter* klavişini basmaq ya da sadəcə digər xassəyə keçmək lazımdır. Dəyişikliyi ləğv etmək üçün *Esc* klavişini basmaq kifayətdir. Xassələrə müəyyən edilmiş qiymətlərin bəziləri təsvir forması faylında, bəziləri isə modul faylında avtomatik olaraq nəzərə alınır.

Komponentlərin əksər xassələrinə, məsələn, `Color` (Rəng), `Caption` (Sərlövhə) və s. susmaya görə qiymətlər əvvəlcədən müəyyən edilmişdir (biz onları da dəyişdirə bilərik).

Komponentə onun `Name` (Ad) xassəsi ilə müraciət olunur. Forma üzərində komponentin yerini və ya ölçülərini dəyişdirdikdə bu parametrlərlə əlaqədar

xassələrin (`Left`, `Top`, `Width`, `Height`) də qiymətləri avtomatik olaraq dəyişir.

Əgər forma üzərində bir neçə komponent seçilərsə, onda Obyektlər inspektorunda xassələrə qiymətlər müəyyənləşdirmək üçün xassə redaktorlarından istifadə olunur. Bu redaktorlar hər hansı bir xassə ilə işlədikdə avtomatik olaraq qoşulur. Belə redaktorlar *4* növdür:

Sadə (mətn) redaktor – xassənin qiyməti daxil edilir və ya adi simvol sətri kimi redaktə edilir. `Caption`, `Left`, `Height`, `Hint` kimi xassələrə qiymətlər bu redaktorla müəyyənləşdirilir;

Sadalanan redaktor – xassənin qiymətləri açılan siyahıdan seçilir. Kursoru xassənin qiymətlər oblastında yerləşdirdikdə peyda olan ox üzərində mausun düyməsini basdıqda açılan siyahıdan qiymətlər seçilir. Bu redaktor `FormStyle`, `Visible` və `ModalResult` kimi xassələr üçün istifadə edilir;

Çoxluq tipli redaktor – xassənin qiymətləri təklif olunan çoxluqdan seçilən qiymətlərin kombinasiyasından ibarətdir. Obyektlər inspektorunda çoxluq tip xassənin adından sol tərəfdə "+" işarəsi olur. Bu xassənin adı üzərində mausun düyməsini iki dəfə basdıqda əlavə siyahı açılır ki, bu siyahıdan xassənin qiymətləri tərtib edilir. Bu siyahı xassənin bütün mümkün qiymətlərindən ibarətdir, hər bir qiymətdən sağda *True* (Doğru) və ya *False* (Yalan) göstərmək olar. *True* qiymətinin seçilməsi onu göstərir ki, bu qiymət qiymətlər kombinasiyasına qoşulur, *False* isə – qoşulmur. Bu redaktor `BorderIcons` və `Constrains` kimi xassələr üçün istifadə edilir.

Obyekt tipli redaktor – xassə özü obyekt olduğu üçün, öz növbəsində o, digər xassələrə (alt xassələrə) malik olur və onların hər birini ayrılıqda redaktə etmək olar. `Font` (Şrift), `Items` (Siyahı) və `Lines` (Sətir) kimi xassələr üçün istifadə edilir. Xassə–obyektin qiymətlər oblastında mötərizədə obyektin tipi, məsələn, (`TFont`) və (`TStrings`) göstərilir. Xassə–obyektin adından solda "+" işarəsi ola bilər. Bu halda xassənin qiyməti çoxluq tipli redaktorla müəyyənləşdirilir. Qiymətlər oblastında üzərində üç nöqtə (`...`) olan düymə ola bilər. Bu o deməkdir ki, bu xassə üçün xüsusi redaktor mövcuddur və onu həmin düymə üzərində mausu basmaqla çağırmaq olar. Məsələn, `Font` xassəsi üçün həmin düyməni basdıqda şriftin parametrlərini müəyyən etmək üçün standart Windows dialoq pəncərəsi açılır.

Obyektlər inspektorunda olan xassələri inspektorun özündə deyil, yunitdə də dəyişmək olar. Bunun üçün mənimsətmə operatorundan istifadə olunur. Məsələn, formanın `Form1` sərlövhəsini "*Bloknot*" adlandırmaq üçün yunitdə

```
Form1.Caption:='Bloknot';
```

kodu yazmaq lazımdır. Lakin, bu çox vaxt aparır, digər tərəfdən belə təyinetmə yalnız layihə yerinə yetirildikdən sonra qüvvəyə minir, layihələndirmə zamanı isə görünmür. Buna baxmayaraq, proqramda belə təyinetmələr çox tez–tez tətbiq edilir.

10.4. Əlavənin funksiyalarının müəyyənləşdirilməsi

Biz komponentlərin forma üzərində yerləşdirilməsini və onlara xassələrin müəyyənləşdirilməsini öyrəndik. Lakin, bu komponentlərin və ümumilikdə əlavənin hansı əməliyyatı yerinə yetirməsi haqqında bizdə təsəvvür yaranmadı. Bizim isə əsas məqsədimiz sadə və ya mürəkkəb məsələləri həll etməyə qabil olan, Windows sistemi altında işləyə bilən, mükəmməl pəncərəli əlavə yaratmaqdan ibarətdir. Bunun üçün proqramçı komponentin yerinə yetirəcəyi funksiyanı – istifadəçinin bu və ya digər əməlinə komponentin reaksiyasını müəyyən etməlidir. Məsələn, forma üzərində yerləşdirilmiş düymə basıldıqda və ya dəyişdirici seçildikdə nə baş verəcəyi müəyyənləşdirilməlidir. Belə reaksiyaların müəyyənləşdirilməsi əlavənin *funksiyalarını* müəyyənləşdirir.

Fərz edək ki, forma üzərində `Button` düyməsi yerləşdirilmişdir və istəyirik ki, bu düymə basıldıqda forma bağlansın. Obyektlər inspektorunun köməyi ilə düymənin sərlövhəsini `Bağlamaq` (`Caption`–`Bağlamaq`) adlandıraq. Bu düyməni basdıqda düymə doğrudan da basılacaq, lakin forma bağlanmayacaq. Çünki, ona *"bağlanmaq" funksiyası* təyin olunmamışdır.

Düymənin bu və ya digər hadisəyə reaksiya verməsi üçün hadisə emaledicisini yaratmaq və ya onun prosedurunu göstərmək lazımdır. Hadisə emaletmə prosedurunu yaratmaq üçün, forma üzərində düyməni seçərək Obyektlər inspektorunun *Events* (*Hadisələr*) səhifəsini açmaq lazımdır. Düymə basıldıqda `OnClick` hadisəsi baş verdiyi üçün məhz bu hadisə emaledicisini yaratmaq lazımdır. Bunun üçün Obyektlər inspektorunun `OnClick` hadisəsinin qiymətlər oblastında mausun düyməsini iki dəfə basmaq lazımdır. Bunun nəticəsində Delphi, avtomatik olaraq, forma modulunda prosedur–emaledicini hazırlayır. Bu zaman Kod redaktoru pəncərəsi (yunit) ön plana keçir, modulun `type` bölməsinə avtomatik olaraq prosedur–emaledicinin adından ibarət

```
Procedure Button1Click(Sender:TObject);
```

sətri yazılır və kursor, prosedurda `Button1` düyməsi basıldıqda yerinə yetiriləcək əməliyyatın kodlarının əlavə ediləcəyi mövqedə yerləşir. Proqramçı bu kodları özü yazır. Bu hadisə nəticəsində forma bağlanmalı olduğu üçün, kursorun yerləşdiyi mövqeyə `Form1.Close;` və ya sadəcə `Close;` yazmaq lazımdır. Beləliklə, forma modulunun kodu belə olacaqdır:

```
unit Unit1;

interface

uses
   Windows, Messages, SysUtils, Classes, Graphics,
   Controls, Forms,Dialogs, StdCtrls;
```

```
Type

   TForm1=Class(TForm)
     Button1:TButton;
     procedure Button1Click(Sender: TObject);


   private
         {private declarations}

   Public
         {public declarations}

   end;


var
   Form1:TForm1;


Implementation

{$R *.DFM}


procedure TForm1.Button1Click(Sender:TObject);

begin

  Form1.Close;   // Biz yalnız bu sətri yazdıq

end;

end.
```

Xatırladaq ki, modulun bu təsvirində yalnız `Form1.Close;` sətri proqramçı tərəfindən yazılmışdır, qalan bütün sətirlər Delphi tərəfindən avtomatik olaraq yaradılmışdır.

Analoji qayda ilə digər komponentlər üçün başqa hadisə emalediciləri yaradılır. Hadisələrə Object Pascal dilinin izahında və uyğun komponentləri öyrəndikdə daha ətraflı baxacağıq.

Prosedur–emaledicini pozmaq üçün proqramçının özünün əlavə etdiyi sətirləri pozmaq kifayətdir. Bundan sonra, layihəni yadda saxladıqda və ya kompilyasiya etdikdə, həmin prosedur avtomatik olaraq bütün fayllardan pozulacaqdır.

Yeni emaledici yaratmaq əvəzinə, əgər varsa, mövcud emaledicini istifadə etmək olar. Bunun üçün Obyektlər inspektorunda, hadisənin qiymətlər

oblastındakı oxun üzərində, mausun düyməsini basmaqla açılan prosedurlar siyahısından onu seçmək lazımdır. Obyektin hadisələri xassələr olduğu üçün müəyyən tipə malik olur. Hər hadisəyə onunla eyni tipli olan emaledici təyin etmək olar. Siyahıdan lazım olan proseduru seçdikdən sonra, o, hadisə emaledicisinə təyin olunur.

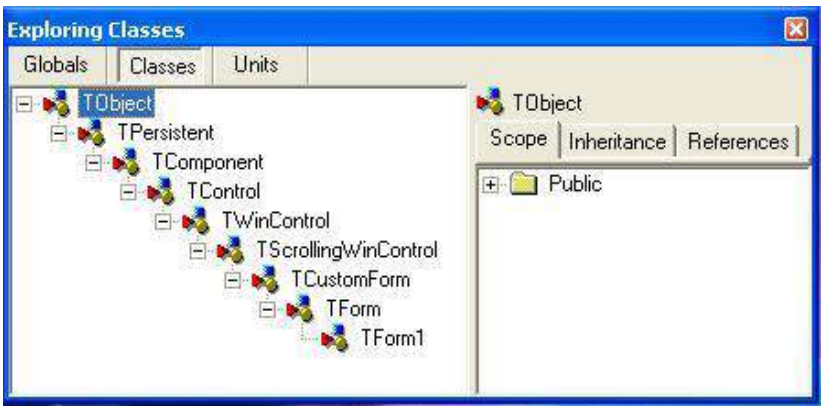
Eyni bir proseduru müxtəlif komponentlər ilə əlaqələndirmək olar. Belə prosedur ümumi emaledici adlanır və onunla əlaqədar istənilən hadisə baş verdikdə çağrılır.

10.5. İnteqrallaşdırılmış iş mühitinin vasitələri

Əlavələri daha asan və səmərəli yerinə yetirmək üçün inteqrallaşdırılmış iş mühitinin (**IDE**) tərkibinə çoxlu vasitələr daxildir. Bu vasitələrin köməyi ilə istifadəçi inteqrallaşdırılmış mühitin parametrlərini, məsələn, Kod redaktorunun şriftini, rəngini, sazlayıcının parametrlərini və s. idarə edə bilər. Bu parametrlərin bəziləri ilə qısaca tanış olaq.

IDE–nin parametrləri *Tools/Environment Options…* (*Servis/Mühitin parametrləri…*) əmri ilə çağrılan *Environment Options* (*Mühitin parametrləri*) dialoq pəncərəsində müəyyənləşdirilir. Bu pəncərədə çoxlu parametrlər qruplaşdırılaraq ayrı–ayrı səhifələrdə yerləşdirilmişdir.

Proqramda istifadə olunan modullara, əlavənin qlobal dəyişənlərinə bir–bir baxmaq, `interface` və ya `implementation` bölmələrində elan olunmuş dəyişənlərə baxmaq, modul kodunda qlobal dəyişənlərin istifadə edildiyi



Şəkil 10.4. Layihə icmalçısının pəncərəsi

hissəyə birbaşa keçmək və s. kimi əməliyyatlar *Layihə icmalçısının* (*Project Browser*) köməyi ilə yerinə yetirilir. Layihə icmalçısı *View/Browser* (*Baxış/Layihə icmalçısı*) əmri ilə çağrılır. Layihə icmalçısının *Exploring<…>* (*Tədqiqat <…>*) pəncərəsində (şəkil 10.4) üç tip obyektə baxmaq olar: *Globals* (*Qlobal simvollar*), *Classes* (*Siniflər*) və *Units* (*Modullar*). Obyektlərin

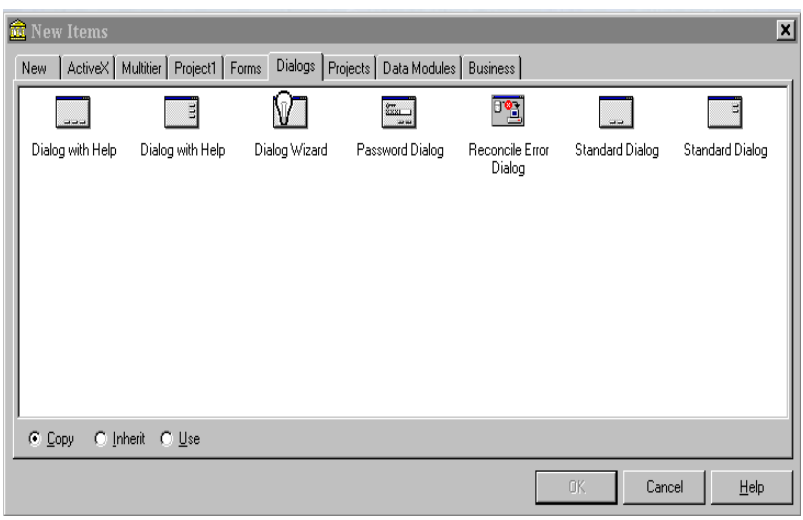
pəncərədə təsvirini idarə etmək üçün *Explorer Options* (*Bələdçinin parametrləri*) pəncərəsindən istifadə edilir. Bu pəncərənin parametrlərinə *Environment Options* (*Mühitin parametrləri*) dialoq pəncərəsinin *Explorer* (*Bələdçi*) səhifəsindən və yaxud Layihə icmalçısının *Properties* (*Xassələr*) kontekst menyusundan daxil olmaq olar.

Delphi eyni bir obyekti şablon kimi dəfələrlə istifadə etməyə imkan verir. Belə obyektlər xüsusi yerdə (*Repository*) saxlanır. *Obyektlərin saxlandığı yer File/New* (*Fayl/Yeni fayl*) əmri ilə çağrılır (şəkil 10.5). Burada ən müxtəlif obyektlər, məsələn, əlavə, forma, hesabat şablonları və Forma ustaları saxlanır. Müxtəlif obyektlər qruplaşdırılaraq pəncərədə görünən səhifələrdə yerləşdirilmişdir. Layihəyə yeni obyekti əlavə etmək üçün üç üsul təklif edilir:

`Copy` –obyektin surəti layihəyə əlavə edilir;
`Inherit`–obyektdən irsi mənimsəmə yolu ilə yaranan varis–obyekt layihəyə əlavə olunur;
`Use` –bütün faylları ilə birlikdə obyekt özü layihəyə əlavə olunur.

Obyektlərin saxlanıldığı yerin parametrləri *Tools/Repository…* (*Servis/Saxlanılan yer…*) əmri ilə çağrılan *Object Repository* (*Obyektlərin saxlanıldığı yer*) pəncərəsi ilə idarə olunur. Bu zaman Obyektlərin saxlanıldığı yerə yeni səhifələr əlavə etmək (*Add Page…*), onları pozmaq (*Delete Page…*) və ya onların adlarını dəyişmək (*Rename Page…),* həmçinin obyektlərə düzəlişlər etmək (*Edit Object*) və onları pozmaq (*Delete Object*) olar.

İnteqrallaşdırılmış mühitdə yeni əlavə yaratdıqda Delphi interfeysini bağlamadan, köhnə layihəni *File/Close All* (*Fayl/Hamısını bağlamaq*) əmri ilə bağlamaq məqsədəuyğundur.



Şəkil 10.5. Yeni obyektin seçilməsi pəncərəsi

1.6. İnteqrallaşdırılmış iş mühitində ən çox istifadə edilən əmrlər

Delphi inteqrallaşdırılmış iş mühitində ən çox istifadə edilən əmrlər Cədvəl 10.1.–də göstərilmişdir.

Cədvəl 10.1. Delphi inteqrallaşdırılmış iş mühitində ən çox istifadə edilən əmrlər

Əmrin adı	**Klavişlər**	**Əmrin yerinə yetirdiyi funksiya**
File/New		*Yeni obyekt (əlavə, forma, yunit və s.) yaratmaq*
Run/Run	**F9**	*Layihənin yerinə yetirilməsi*
View/Units…	**Ctrl+F12**	*Modulun ekranda təsviri*
View/Forms	**Shift+F12**	*Forma konstruktorunun ekranda təsviri*
Component/Configure Palette		*Komponentlər palitrasının sazlanması*
View/Code Explorer		*Kod bələdçisinin ekranda təsviri*
View/Object Inspector	**F11**	*Obyektlər inspektorunun ekranda təsviri*
Project/View Source		*Layihə faylının ekranda təsviri*
Project/Options…	**Ctrl+Shift+F11**	*Layihənin parametrlərinin təyini*
Project/Compile Project1	**Ctrl+F9**	*Layihənin kompilyasiyası*
Project/Compile All Projects		*Bütün layihənin kompilyasiyası*
Project/Build Project1		*Layihənin yığılması*
Project/Build All Projects		*Bütün layihənin yığılması*
Run/Program Reset	**Ctrl+F2**	*Sonsuz dövr etmə zamanı proqramın dayandırılması*
Tools/Environment Options		*Delphi interfeysinin parametrlərinin sazlanması*
View/Project Manager	**Ctrl+Alt+F11**	*Layihə menecerinin ekranda təsviri*
View/Debug Windows		*Layihə sazlayıcısının ekranda təsviri*
View/Browser	**Shift+Ctrl+B**	*Layihə icmalçısının ekranda təsviri*
Tools/Repository		*Obyektlərin saxlanıldığı yerin sazlanması*
	F12	*Forma konstruktoru ilə Kod redaktorunun yerlərinin dəyişdirilməsi*
Save Project As...		*Layihəni necə yadda saxlamaq*
Save All	**Shift+Ctrl+S**	*Bütün layihəni saxlamaq*
Close All		*Bütün layihəni bağlamaq*

On birinci fəsil

OBJECT PASCAL DİLİ

Bu fəsildə biz Object Pascal dilinin əsas elementləri ilə tanış olacağıq. Object Pascal dili Delphi proqramlaşdırma dilidir və standart Pascal dilinin obyektyönlü genişlənməsidir. Kitabın birinci hissəsində Turbo Pascal dilinin əsas hissəsi ətraflı şərh olunduğu üçün, bu fəsildə Object Pascal dilinin Turbo Pascal dilindən fərqli cəhətlərinə nəzər yetiriləcək, bəzi sahələr daha da gücləndiriləcək və əsas diqqət obyektyönlü proqramlaşdırmaya yönəldiləcəkdir. Obyektyönlü proqramlaşdırmanın xüsusiyyətləri, sinif, sahə, xassə, metodlar öyrəniləcək və Object Pascal dilində proqramlaşdırmanın əsas vasitə və üsullarına baxacağıq.

11.1. Dilin əlifbası

Object Pascal dilinin əlifbasına aşağıdakı simvollar daxildir:

53 hərf – latın əlifbasının baş (**`A – Z`**) və kiçik (**`a – z`**) hərfləri və nəzərə çarpdırma (**`_`**) işarəsi;

10 ərəb rəqəmləri: **`0 – 9`**;

23 xüsusi simvol:

```
+  –  *  /  .  ,  :  ;  =  >  <  '
(  )  {  }  [  ]  #  $  ^  @  probel;
```

Xüsusi simvolların birləşməsindən aşağıdakı simvollar əmələ gəlir:

:= –*mənimsətmə*;
<> –*bərabər deyildir*;
.. –*qiymətlərin dəyişmə diapazonu*;
<= –*kiçik və ya bərabərdir*;
>= –*böyük və ya bərabərdir*;
(* və ***)** –**{** *və* **}** *fiqurlu mötərizələrin əvəzləyiciləri*;
(. və **.)** –**[** *və* **]** *kvadrat mötərizələrin əvəzləyiciləri.*

Object Pascal dilinin *lüğəti* və *proqramın strukturu* Turbo Pascal dilinin lüğəti və strukturu ilə tamamilə eynidir.

11.2. Şərhlər

Şərhlər – proqramın daha yaxşı başa düşülməsi üçün proqramın istənilən yerində yazıla bilən izahedici mətndən ibarətdir. Belə mətn bir sətirdə və ya bir neçə sətirlərdə yazıla bilər. Əgər sətrin əvvəlində ikiqat *sleş* (//) işarəsi yazılarsa, onda bütün bir sətir şərh kimi qəbul olunacaqdır. Şərhlər bir neçə sətirdə yazılarsa, onları **{** və **}** mötərizələri və ya onların ekvivalenti olan **(*** və ***)** simvol birləşmələri daxilində yazmaq lazımdır.

Misal.

```
// Nyuton üsulu


{ Biz Delphi dilini
öyrənirik. Delphi obyektyönlü
proqramlaşdırma
dilidir }


// sum:=sum+x;


{ begin
for k:=1 to 10 do
a[k]:=x*k;
end; }


(* Laboratoriya işi №1. Dövrü hesablama
 proseslərinin proqramlaşdırılması *)
```

11.3. Verilənlərin tipləri

Object Pascal dilində müəyyən edilmiş tiplərin təsnifatı cədvəl 11.1 – də göstərilmişdir.

Cədvəl 11.1. Object Pascal dilində müəyyən edilmiş tiplər

Qrup	Alt qrup
Sadə	***Tamədədli*** ***Liter (simvollu)*** ***Məntiqi (bul)*** ***Həqiqi***
Struktur	***Sətir*** ***Massiv*** ***Çoxluq*** ***Yazı*** ***Fayl*** ***Sinif***
Göstərici	***Tipləşdirilmiş*** ***Tipləşdirilməmiş***
Variant	
Prosedur	

11.3.1. Verilənlərin sadə tipləri

Bəzi sadə tiplər fiziki (əsas) və ümumi tiplərə bölünür. *Fiziki tiplərin* təməli dil yaradıldıqda qoyulur və kompüterin xüsusiyyətlərindən asılı olmur. *Ümumi tiplər* fiziki tiplərdən hər hansı birinə uyğun gəlir və kompilyator bu tipləri istifadə etdikdə daha səmərəli kod yaratdığı üçün, onlara daha çox üstünlük verilir.

Bəzi sıralı tipləri proqramçı özü də yarada bilər, ona görə də belə tiplərə istifadəçi tipləri deyilir. İstifadəçi tiplərinə *sadalanan* və *interval* tiplər aiddir.

11.3.1.1. Tamədədli tiplər

Fiziki *tamədədli* tiplər və onların əsas göstəriciləri cədvəl 11.2 – də verilmişdir.

Tamədədli tiplər üçün iki – `Integer` və `Cardinal` tipli ümumi tiplər müəyyənləşdirilmişdir. `Integer` tipli verilənlərin göstəriciləri `LongInt` tipinin, `Cardinal` tipinin göstəriciləri isə `LongWord` tipinin göstəriciləri ilə eynidir.

Tam ədədlərin qarşısında "+" və "–" işarələri yazıla bilər. Onlar onluq və onaltılıq say sistemlərində təsvir oluna bilər. Tam ədədləri onaltılıq say sistemində təsvir etmək üçün ədədin qarşısında `$` işarəsi qoyulmalıdır. Tam ədədlər `$00000000` – `$FFFFFFFF` diapazonunda yerləşir.

Cədvəl 11.2. Tamədədli tiplər

Tipin adı	Dəyişmə diapazonu	Yaddaşda təsviri
`Byte`	0 – 255	*işarəsiz 1 bayt*
`Word`	0 – 65 535	*işarəsiz 2 bayt*
`ShortInt`	(–128) – 127	*işarə ilə 1 bayt*
`Int64`	(-2^{63}) – $(2^{63}-1)$	*işarə ilə 8 bayt*
`SmallInt`	(–32 768) – 32 767	*işarə ilə 2 bayt*
`LongWord`	0 – 4 294 967 295	*işarəsiz 4 bayt*
`LongInt`	(–2 147 483 648) – 2 147 483 647	*işarə ilə 4 bayt*

11.3.1.2. Liter tiplər

Liter tiplərin qiymətləri ayrı–ayrı simvollar yığımından ibarət olur. Simvollar üçün də fiziki və ümumi tiplər müəyyənləşdirilmişdir. Fiziki tiplər **`AnsiChar`** və **`WideChar`** tipləridir.

`AnsiChar` tipi *1* bayt yer tutur və simvolları kodlaşdırmaq üçün Amerika milli standartlar institutunun *ANSI* (*American National Standarts Institute*) kodunu istifadə edir. `WideChar` tipi *2* bayt yer tutmaqla *Unicode* beynəlxalq simvollar yığımından istifadə edir. *Unicode* simvollar yığımına *60* mindən çox element daxildir və milli əlifbaların simvollarını kodlaşdırmağa imkan verir. *Unicode* simvollarının ilkin *256* simvolları *ANSI* kodu ilə eynidir.

Bu fiziki tiplərdən başqa, **`Char`** ümumi tipi də müəyyənləşdirilmişdir ki, bu tip `AnsiChar` tipinə ekvivalentdir.

Object Pascal dilində *məntiqi*, *sadalanan* və *interval* tiplər Turbo Pascal dilində olduğu kimidir.

11.3.1.3. Həqiqi tiplər

Həqiqi tiplər də fiziki və ümumi olur. Fiziki tiplər cədvəl 11.3 – də göstərilmişdir:

Həqiqi ədədlərin ümumi tipi `Real` tipidir və o, `Double` tipinə uyğundur.

Həqiqi ədədlər, bütün dillərdə olduğu kimi, burada da qeyd olunmuş və sürüşkən vergüllü formalarda təsvir olunur və burada əlavə izahata ehtiyac görmürük. `Comp` və `Currency` tipləri həqiqi ədədləri qeyd olunmuş vergüllü formada təsvir edir. `Comp` tipi əslində tam ədədləri təsvir edir, lakin həqiqi tipə

aiddir. Bu tipli dəyişənə həqiqi tipli qiymətlər mənimsətdikdə, o avtomatik olaraq yaxın tam ədədə qədər yuvarlaqlaşdırılır.

Cədvəl 11.3. Həqiqi ədədlərin fiziki tipləri

Tipin adı	Dəyişmə diapazonu	Yaddaşda təsviri, *bayt*
Real48	$2{,}9\cdot10^{-39}$ – $1{,}7\cdot10^{38}$	*6*
Single	$1{,}7\cdot10^{-45}$ – $3{,}4\cdot10^{38}$	*4*
Double	$5\cdot10^{-324}$ – $1{,}7\cdot10^{308}$	*8*
Extended	$3{,}6\cdot10^{-4951}$ – $1{,}1\cdot10^{4932}$	*10*
Comp	$(-2\cdot10^{63}+1)$ – $(2\cdot10^{63}-1)$	*8*
Currency	(–922337203685477,5808) – 922337203685477,5807	*8*

11.3.2. Verilənlərin struktur tipləri

Cədvəl 11.1 – də göstərildiyi kimi, struktur tiplərə aşağıdakılar aiddir:

- *sətirlər*;
- *massivlər*;
- *çoxluqlar*;
- *yazılar* ;
- *fayllar*;
- *siniflər*.

Sinif tiplər verilənlərin xüsusi tipləridir. Obyektyönlü proqramlaşdırmada sinif tiplər xüsusi əhəmiyyət kəsb etdiyi üçün, onların izahı fəslin sonunda, ayrı bir bölmədə şərh olunacaqdır. Yerdə qalan tiplərin isə Turbo Pascal dilindən fərqli cəhətlərinə baxaq.

11.3.2.1. Sətirlər

Sətirlər üç fiziki və bir ümumi tiplə təsvir olunur. Fiziki tiplər `ShortString` (*255* simvol), `AnsiString` (*~$2\cdot10^{31}$* simvol) və `WideString` (*~$2\cdot10^{30}$* simvol) tiplərindən ibarətdir. Əslində sətir dəyişənləri, müəyyən mənada, elementləri simvollar olan massivləri ifadə edir.

`AnsiString` və `WideString` tipləri dinamik massivləri təsvir edir. Birinci tip – *ANSI*, ikinci tip isə *Unicode* kodları ilə kodlaşdırılır.

Sətir tipli verilənlərin ümumi tipi `String` tipidir ki, `ShortString` və ya `AnsiString` tiplərinə uyğun gəlir. Sətirlər massivlərə uyğun gəldiyindən, sətrin istənilən simvoluna massiv elementi kimi müraciət etmək olar. Bunun

üçün dəyişənin adının yanında, kvadrat mötərizə daxilində, simvolun nömrəsini göstərmək lazımdır. Massivin sıfrıncı elementi idarəedici element olmaqla sətir tipli dəyişənin faktiki uzunluğunu göstərir.

Sətir tipli verilənlərin yuxarıda göstərilən tiplərindən başqa, `PChar` tipi də mövcuddur ki, bu tip sonu sıfırla qurtaran sətirləri təsvir edir, yəni sətrin sonunda `#0` kodu yerləşir. Bu tip dəyişən sətrin başlanğıcına *göstəricidir*, başqa sözlə, maşının yaddaşında sətrin birinci simvolunu göstərir. Artıq bilirik ki, `String` tipli sətirlərin uzunluğu birinci simvolla müəyyənləşdirilir. `PChar` tipli sətirlərdə isə sətrin uzunluğu kompüterin əsas yaddaşının ölçüsü ilə məhdudlaşır. Bəs proqram bu sətrin uzunluğunu necə müəyyən edir? Bunu başa düşmək üçün sətrin simvollarının yaddaşda necə yadda saxlanmasını xatırlayaq. Bildiyimiz kimi, hər bir simvol bir ədəddir. Bununla yanaşı, yalnız bir ədəd mövcuddur ki, o, simvolların kodlaşdırılmasında iştirak etmir. Bu sıfır ədədidir. Bax, elə buna görə də, proqram sıfır ədədi rast gələnə qədər `PChar` tipli sətrin simvollarının kodlarını oxuyur və sıfır ədədini sətrin sonu əlaməti kimi qəbul edir.

Proqramda `PChar` tipini birbaşa istifadə etmək olmaz. Məsələn,

```
var
   s: PChar;
begin
  S:='Delphi';
end;
```

proqram fraqmenti səhv olacaqdır. Burada, `S` dəyişəni `PChar` tipli elan olunaraq ona qiymət mənimsədilmişdir, halbuki, `S` sətir tipli dəyişən deyil, *göstəricidir*. Biz isə sadəcə olaraq onu elan etmişik, yaddaşda isə ona yer ayırmamışıq. `String` tipli sətrin uzunluğunu göstərmədikdə, Delphi avtomatik olaraq, ona maksimal ölçü üçün, yəni *255* simvolun yerləşməsi üçün yaddaşda yer ayırır. `PChar` tipinin isə maksimal ölçüsü anlayışı olmadığı üçün, yaddaşda onun yerləşməsi qayğısına proqramçı özü qalmalıdır.

`PChar` tipi haqqında təsəvvür yaratmaq üçün bir proqram fraqmentinə baxaq.

Misal.

```
var
    s: array[1..1000] of char;
   s1: PChar;
begin
  s1:=@s;
end;
```

Burada, `s` dəyişəni *1000* simvoldan ibarət massiv, `s1` dəyişəni isə `PChar` tipli elan olunmuşdur və sonra `s1` dəyişəninə `s` qiyməti mənimsədilmişdir. Bununla da `s1` dəyişəni *1000* simvoldan ibarət massivin yerləşdiyi yaddaş oblastını göstərir.

11.3.2.2. Massivlər

Object Pascal dilində statik və dinamik massivlər mövcuddur.

Statik massivlərin ölçüləri elanetmə bölməsində əvvəlcədən müəyyənləşdirilir. Proqramın yerinə yetirildiyi müddətdə onların ölçüləri dəyişmir. Belə massivlərin ümumi təsvir forması aşağıdakı kimidir:

`Array` [*indekslər*] **`of`** *elementlərin tipi***;**

Burada, *indekslər* interval tip ilə göstərilir.

Misal.

```
Const max= 1000;
Type a= array[1..5,1..8] of integer;
Var  x,y: a;
     b: array[1..50] of real;
     c: array[1..40] of char;
     d: array['a'..'z'] of integer;
     z: array[1..max] of boolean;
```

`x` və `y` dəyişənləri *40* elementdən – *5* sətir və *8* sütundan ibarət ikiölçülü massiv olur; massivin elementləri `integer` – tam tiplidir. `b` dəyişəni *50* həqiqi tipli elementdən, `c` dəyişəni *40* simvol tipli elementdən, `d` dəyişəni *26* tam ədədlərdən, `z` dəyişəni isə *1000* məntiqi tipli elementlərdən ibarət massivlər kimi təyin olunur.

Dinamik massivlər isə elə massivlərdir ki, elanetmə zamanı yalnız onların elementlərinin tipi göstərilir, ölçüləri isə proqram yerinə yetirildikdə müəyyən edilir və bu ölçülər proqram boyu dəyişə bilər. Bu massivlərin ümumi təsvir forması belədir:

`Array of` *elementlərin tipi***;**

Dinamik massivlərin elementlərinin nömrəsi həmişə sıfırdan başlayır.

Proqramın icrası zamanı dinamik massivə ölçülərin verilməsi xüsusi prosedurla yerinə yetirilir. Bu prosedurun ümumi forması belədir:

`SetLength(var` *s; NewLength* **: `integer);`**

Burada, *s* – dinamik massivin adı, *NewLength* isə massivin indeksinə veriləcək qiymətdir.

Misal.

```
Var  m: array of real;
     i: integer;
begin
  …
  SetLength(m,100);
  for i:=0 to 99 m[n]:=n;
  SetLength(m,200);
  …
end.
```

`m` dinamik massivinin elementləri həqiqi tiplidir. Əvvəlcə onun elementlərinin sayının *100* olduğu müəyyənləşdirilir. Massivin hər bir elementinə onun sıra nömrəsinin qiyməti verilir. Massivin elementinin nömrəsi sıfırdan başladığı üçün onun sonuncu elementinin nömrəsi *99* olur. Dövrdən sonra massivin ölçüsünə *200* qiyməti verilir.

Çoxölçülü dinamik massivləri (məsələn, ikiölçülü massivi) təsvir etmək üçün aşağıdakı konstruksiyadan istifadə olunur:

`Array of Array of` *elementlərin tipi***;**

Bu halda, çoxölçülü (xüsusi halda ikiölçülü) dinamik massivlərə yeni ölçülər təyin etmək üçün tətbiq olunan prosedur belə yazılır:

`SetLength (var` *s*; *NewLength1*, *NewLength2* **`: integer);`**

Burada, *NewLength1* və *NewLength2* – massivin uyğun olaraq *1*–ci və *2*–ci indekslərinə veriləcək yeni qiymətlərdir.

Object Pascal dilində *çoxluqlar* və *yazılar* Turbo Pascal dilində olduğu kimidir. *Fayllar* da Turbo Pascal dilində olduğu kimidir, lakin yalnız mətn tipli faylların təsvirində **`Text`** sözü əvəzinə **`TextFile`** və ya **`System.Text`** yazmaq lazımdır.

11.3.3. Göstəricilər

Adi dəyişən proqramda təsvir olunduqdan dərhal sonra, əsas yaddaşda onun üçün yer ayrılır və proqramın icrası zamanı daimi olaraq orada saxlanır. Bu cür dəyişənlər *statik* dəyişənlər adlanır. Statik dəyişənlər kompüterin yaddaşından səmərəli istifadə etməyə imkan vermir. *Göstəricilər* proqramın icrası zamanı dəyişənləri yaratmağa imkan verir, başqa sözlə, belə dəyişənlər dinamik olur. Proqramın icrası zamanı, zərurət yaranarsa, həmin dəyişənin tutduğu yaddaşı boşaldıb, başqa dəyişən üçün istifadə etmək olar.

Göstəricilərin vacibliyi haqqında biliklərimizi bir qədər də artıraq. Gəlin, prosedurların necə çağrılmasını xatırlayaq. Tutaq ki, çağrılacaq prosedurun iki parametri var: ədəd və sətir. Bu prosedur necə çağrılır? Bilirik ki, bu parametrlər stekə qəbul olunur, yəni əvvəlcə birinci ədəd, sonra isə ikinci ədəd stekə qəbul olunur. Prosedur icra olunmazdan öncə bu parametrlər əks istiqamətdə stekdən çıxarılır. Stekə qəbul olunmuş birinci ədəd *2* bayt yer tutacaq, ikinci parametr – sətir isə, məsələn, əgər *10* simvoldan ibarət olarsa, *10* bayt, üstəgəl sətrin sonunu və ya onun ölçüsünü göstərmək üçün *1* bayt yer tutacaq. Bütövlükdə prosedur üçün stekdə ən azı *12* bayt yaddaş tələb olunacaq. İndi isə təsəvvür edək ki, prosedura ötürüləcək dəyişən *500* elementdən ibarət massivdir, sətir isə *1000* simvoldan ibarətdir. Bu zaman stekdə kilobaytlarla yaddaş tələb olunacaq. Kompüterlərin hazırkı yaddaş həcmi ilə müqayisədə bu çox kiçik yaddaşdır və bu böyük faciə deyildir. Lakin, proqramçılar unudurlar

ki, bu qədər həcmə malik informasiya əvvəlcə stekə köçürülür, sonra isə həmin həcmdə yaddaşdan çıxarılır. Belə köçürmə isə kifayət qədər vaxt tələb edir və proqram mənasız bir işə boş–boşuna nə qədər vaxt sərf edəcək. Bəs, əgər bizə prosedura, ölçüsü *3-4* meqabayt olan təsvir ötürmək lazım gələrsə, onda necə? Bu təsvirləri də stekə köçürək? Bir neçə yüksək keyfiyyətli təsvirlər olarsa, onda stek tam dolacaqdır. Bu vəziyyətdən çox asan çıxış yolu var: stekə verilənləri, təsvirləri göndərmək yox, onların yerləşdiyi yaddaş oblastına *göstərici* vermək lazımdır. İstənilən göstərici isə cəmi *4* bayt yer tutur.

Beləliklə, göstəricilərdən istifadə etməklə, proqramçı kompüterin yaddaşından daha səmərəli istifadə etmək imkanı qazanır, lakin bu proqramın mürəkkəbləşməsi hesabına başa gəlir.

Object Pascal dilində tipləşdirilmiş və tipləşdirilməmiş göstəricilər mövcuddur.

Tipləşdirilmiş göstərici təsvir və ya elanetmə zamanı müəyyənləşdirilmiş tipli verilənlərə istinad edə bilər. Bu zaman ünvanlaşdırılan verilənlərin tipləri qarşısında **^** işarəsi qoyulur. Tipləşdirilmiş göstəricilərin ümumi təsvir forması belədir:

Type *göstəricinin tipi* **= ^** *ünvanlaşdırılan verilənin tipi***;**

Tipləşdirilməmiş göstərici **Pointer** tipli olur və istənilən tip verilənlərə istinad oluna bilər.

Misal.

```
Type
   top= ^integer;
   tar= ^massiv;
   massiv= array [1..10] of LongInt
Var
   a1,a2: ^integer;
   c1,c2: top;
       r: tar;
     s,q: pointer;
```

`a1`, `a2`, `c1` və `c2` dəyişənləri `integer` tipli dəyişənləri, `r` dəyişəni elementləri `Longint` tipli olan massivi, `s` və `q` dəyişənləri isə istənilən tipli dəyişənləri ünvanlaşdıra bilər.

Göstərici vasitəsilə ünvanlaşdırılan verilənlərin qiymətlərini istifadə etmək üçün göstərici adsızlaşdırılmalıdır. Bu məqsədlə dəyişənin adının sağ tərəfində **^** işarəsi yazılır.

İstənilən göstəriciyə sıfır qiyməti mənimsətmək olar, lakin bu *0* ədədi deyil, `Nil` qiymətidir. Əgər göstəriciyə `Nil` (sıfır) qiyməti mənimsədilərsə, onda göstərici heç bir dəyişənə istinad etməyəcək, sanki məhv ediləcəkdir. Bəs göstəriciyə nə üçün sıfır qiyməti mənimsədilməlidir? Bu yaddaşa heç bir təsir göstərmir, lakin, təhlükəsizlik nöqteyi–nəzərindən əhəmiyyətlidir. Məsələ ondadır ki, yaddaş boşaldıldıqdan sonra, göstərici hər hansı bir ədədi (hər hansı

ünvanı) özündə saxlayır, lakin bu ünvandakı verilənlər doğru deyildir. Əgər bu ünvana müraciət olunarsa, onda səhv baş verəcəkdir.

Göstərici vasitəsilə müraciət olunacaq obyektin ünvanını müəyyənləşdirmək üçün həmin obyektin adının sağ tərəfində **@** əməliyyatını yazmaq lazımdır.

Misal.

```
Var  p: ^integer;
   n,k: integer;
   ...
   p:= @n;
   n:= 100;
   k:= p^+10;
```

Burada, `n` dəyişəninə `p` göstəricisi ilə istinad edilir və nəticədə `k` dəyişəninin qiyməti *110* olur.

Misal.

```
Var
    s: pointer;
   s1: string;
begin
   s:= @s1;
  s1:= 'Mən Delphi öyrənirəm';
  Edit1.Text:= string(s^);
end;
```

Bu misalın birinci sətrində `s` göstəricisinə `s1` sətrinin göstəricisi mənimsədilmiş, sonra isə həmin sətrin məzmunu dəyişdirilmişdir. Sonuncu sətirdə `s` ünvanında yerləşən mətn `Edit` mətn redaktoruna çıxarılmışdır (bu redaktorla növbəti fəsillərdə tanış olacağıq). Bunun üçün aşkar şəkildə göstərilməlidir ki, `s` ünvanında məhz `string(s^)` sətri yerləşir.

11.3.4. Variant tiplər

Verilənlərin variant tipləri o zaman istifadə olunur ki, onların tipləri ya əvvəlcədən məlum olmur və ya proqramın icrası zamanı dəyişir.

Variant tipi təsvir etmək üçün **`Variant`** operatorundan istifadə edilir. Dəyişən bu tip elan edildikdən sonra, ona tamədədli (`Int64` tipindən başqa), həqiqi, simvol, sətir və məntiqi tiplər mənimsətmək olar.

Misal.

```
Var a1,a2: Variant;
     m: integer;
     n: string;
     k: real;
```

```
begin
  ...
   m:= 15;
  a1:= m;
   n:= 'BAKI';
  a2:= n;
   k:= 1.07;
  a1:= k;
  a2:= True;
  ...
end
```

Göründüyü kimi, `a1` dəyişəninə əvvəlcə tamədədli (`15`), sonra isə həqiqi (`1.07`) və `a2` dəyişəninə isə əvvəlcə sətir tipli (`'BAKI'`), sonra isə məntiqi (`True`) qiymətlər mənimsədilmişdir.

11.3.5. Prosedur tiplər

Verilənlərin prosedur tipləri prosedur və funksiya tipli alt proqramları adi qiymətlər kimi istifadə etməyə imkan verir. Bu qiymətlər hər hansı dəyişənə mənimsədilə və ya parametrlər kimi ötürülə bilər.

Prosedur tipli verilənlərin təsviri prosedur və funksiyanın sərlövhəsinə oxşayır, lakin alt proqramın adı olmur.

Misal.

```
type TNotifyEvent=
      procedure(Sender:TObject) of object;
```

Prosedur tip verilənlər hadisə emaledicisini təyin etmək üçün istifadə olunur. Bu prosedurun `TObject` tipli yalnız bir `Sender` parametri vardır. Hadisə emaledicisini Obyektlər inspektorunun köməyi ilə də təyin etmək olar.

Misal.

`Button1` düyməsinin `OnClick` düyməbasma hadisəsinə emaledici kimi `Button1Click` proseduru belə təyin olunur:

```
Button1.OnClick:=Button1Click;
```

Hadisə emaledicilərinin müfəssəl izahına növbəti fəsillərdə baxacağıq.

11.4. İfadələr

Object Pascal dilində hesabi ifadələr, məntiqi ifadələr və sətir ifadələri Turbo Pascal dilində olduğu kimidir. Delphi–də çoxlu standart prosedur və funksiyalardan istifadə olunur ki, onların ən əsasları kitabın sonundakı *Əlavədə* göstərilmişdir. Yadda saxlayın ki, bu *Əlavədə* verilən bir sıra riyazi prosedur və funksiyaları istifadə etdikdə, modulun **`Uses`** bölməsinə **`Math`** modulunu qoşmaq lazımdır.

11.5. Operatorlar

Turbo Pascal dilində proqrama ilkin verilənlər daxiletmə prosedurları ilə daxil edilir, nəticələr isə xaricetmə prosedurları ilə ekranda təsvir olunur və ya fayla yazılırdı. Delphi sistemində isə bu əməliyyatlar xüsusi *vizual komponentlərlə* yerinə yetirilir. Bu komponentləri növbəti fəsillərdə ətraflı öyrənəcəyik.

Birinci hissədə öyrəndiyimiz mənimsətmə operatoru, keçid operatoru, boş operator, strukturlaşdırılmış operatorlar, o cümlədən, tərkibli operator, şərti operator, seçim operatoru, parametrli, ilkin şərtli və son şərtli dövr operatorları, daxilolma operatoru Turbo Pascal dilində olduğu kimidir.

Object Pascal dilində də alt proqramların prosedur və funksiya tipləri, rekursiv alt proqram mövcuddur. Alt proqramlarda parametr və arqumentlər haqqında deyilmiş mülahizələr Object Pascal dili üçün də doğrudur. Modullar isə obyektyönlü proqramlaşdırmada mərkəzi yer tutur. Yuxarıda qeyd etdiyimiz kimi, Delphi–də proqramlaşdırma prosesi əslində modulda yerinə yetirilir. Ona görə də fəslin sonunda modulların strukturuna yenidən baxacağıq.

11.6. Obyektyönlü proqramlaşdırmanın xüsusiyyətləri

Bu bölmənin məqsədi oxucuları obyektyönlü proqramlaşdırmanın əsas ideya və prinsipləri ilə tanış etməkdən ibarətdir. Əslində bizim əsas və son məqsədimiz məhz obyektyönlü proqramlaşdırma texnologiyasının xüsusiyyətlərini mükəmməl öyrənməkdən ibarətdir. Çünki, bu texnologiya Turbo Pascal dilinin 5.5 (1989–cu il) versiyasından başlayaraq sonrakı versiyalarda bu və ya digər səviyyədə inkişaf etdirildiyi halda, Object Pascal dili və onun əsasında yaradılmış Delphi proqramlaşdırma sistemi tamamilə obyektyönlü proqramlaşdırma konsepsiyalarını həyata keçirir. Bu o deməkdir ki, yaradılan hər bir əlavə bir–biri ilə qarşılıqlı təsirlərdə olan *obyektlərdən* ibarət olur. Başqa sözlə, kompüter proqramı müxtəlif obyektlər üzərində yerinə yetiriləcək əməliyyatların təsvirindən ibarət olur. Obyekt kimi, məsələn, qrafik obyektlər, verilənlər bazasında yazılar və ya ədədi qiymətlər yığımı götürülə bilər.

Obyektyönlü proqramlaşdırmanın xüsusiyyətlərini anlamaq üçün proqramlaşdırmanın keçdiyi inkişaf mərhələlərinə qısaca nəzər yetirək. İlk proqramlar bütöv mətnlərdən ibarət idi. Burada proqram əmrlər ardıcıllığından ibarət idi. Bu proqramlarla çox işlər görmək mümkün deyildi. Belə proqramlara misal olaraq xətti hesablama proseslərinin proqramlarını misal göstərmək olar. Proqramda müəyyən məntiqi əməliyyatları yerinə yetirmək üçün proqramçının

imkanında yalnız şərti keçid operatorları mövcud idi. Bu operatorların tətbiqi ilə əmrlərin yerinə yetirilmə ardıcıllığını dəyişdirmək mümkün idi. Bununla bərabər, proqram yenə də "müstəvi" proqram olaraq qalırdı, belə proqramlarda yalnız ilkin verilənlərin daxil edilməsi kimi dialoqlar təşkil olunurdu. Lakin, biz *Ms Office* proqramları ilə işlədikdə onların xətti olmasını deyə bilmərik, çünki bu proqramlarda çoxlu canlı dialoqlar təşkil olunmuşdur. Siz nə istədiyinizi deyirsiniz, proqram isə əvəzində onu icra edir. Xətti ("müstəvi") proqramlaşdırma isə belə dialoqları yaratmağa imkan vermir. Proqramlaşdırmanın növbəti mərhələsində prosedur yanaşma meydana gəldi. Prosedur və funksiyaların tətbiqi bir vaxtlar proqramlaşdırmanın səmərəliliyini artırmaq üçün çox vacib və çox böyük bir addım idi. Bu yanaşmada proqramın müəyyən hissələri ayrı bloklar şəklində tərtib olunur və əsas proqramda ona dəfələrlə müraciət olunurdu. Hər müraciət zamanı isə prosedura müxtəlif qiymətlər ötürülə bilirdi. Bilirik ki, əksər hallarda prosedur və funksiyaların formal parametrləri mövcud olur ki, onlara müraciət etdikdə bu parametrlər faktik arqumentlərlə əvəz edilməlidir. Bu halda isə prosedur və funksiyaların səhv verilənlərlə çağrılması təhlükəsi əmələ gəlirdi ki, bu da proqramın icrasında imtinalara və ya onun qəza ilə yekunlaşmasına səbəb ola bilirdi. Ona görə də belə ənənəvi yanaşmanın tətbiqi ümumiləşməsi kimi verilənlərin və alt proqramların (prosedur və funksiya) birləşdirilməsi məqsədəuyğun hesab edilə bilərdi. Bu isə proqramlaşdırmanın növbəti inkişaf mərhələsi olan obyektyönlü proqramlaşdırmanın meydana gəlməsinə səbəb oldu. Burada proqramlar artıq "müstəvi" proqramlar deyildir və proqramçı yalnız prosedur və funksiyalarla kifayətlənmir, bütöv *siniflərlə* əməliyyat aparır.

Ənənəvi proqramlaşdırmada verilənlərin və ya onları emal edən metodların dəyişməsi proqrama ciddi dəyişikliklərin edilməsi zərurətini yaradır. Proqramçılar isə çox yaxşı bilirlər ki, proqramda dəyişiklik etmək ən xoşagəlməz işlərdən biridir, çünki, bu zaman səhvlərin yaranma ehtimalı artır ki, bu da vaxt sərfinin çoxalmasına gətirir. Obyektyönlü proqramlaşdırmadan istifadə olunduqda isə bu hal aradan qaldırılır, belə ki, minimal itkilərlə proqram modifikasiya olunur, genişləndirilir və ya ona əlavələr edilir. Beləliklə, obyektyönlü proqramlaşdırmanın əsas prinsipi ondan ibarətdir ki, haçansa, kim tərəfindənsə, yaradılmış proqram atılmamalı, itməməlidir və hazır blok şəklində digər proqramçıların proqramlarında istifadə edilməlidir. Windows əlavələri ilə işləyən istifadəçilər yəqin xatırlayırlar ki, *Word*, *Excel*, *Access* və s. kimi əlavələrdə tamamilə eyni qayda ilə icra olunan nə qədər əməliyyatlardan (eyni menyular, şriftlər, pəncərələr və s.) istifadə olunur. Bu əməliyyatlar blok kimi müxtəlif əlavələrdə istifadə edilmişdir. Yeni proqramlar işlənərkən, zərurət yarandıqda, mövcud proqramlardan hazır bloklar götürülür və onlar yeni tələbatlara uyğunlaşdırılır. Lakin, bütün bu deyilənlərdən belə nəticəyə gəlmək olmaz ki, obyektyönlü proqramlaşdırma proqramçıları bütün bəlalardan xilas edən "dərmandır", bununla bərabər qabaqcıl texnologiya kimi onun rolu şübhəsizdir. Obyektyönlü proqramlaşdırmanın ideya və metodlarını öyrənmək o qədər də asan məsələ deyildir, lakin bu texnologiyanın tətbiqi mürəkkəb proqramların işlənməsini əhəmiyyətli dərəcədə sadələşdirməyə imkan verir.

11.6.1. Obyektlər və onların xassələri

11.6.1.1. Obyekt nədir

Hər şeydən əvvəl, obyekt hər hansı bir konkret şeydir. Biz deyə bilərik ki, o haradan başlayır və harada qurtarır. İkincisi, o hər hansı daxili quruluşa malikdir ki, biz onu bilməyə də bilərik. Üçüncüsü, obyekt müəyyən hərəkətə malik olur ki, biz bu hərəkəti müşahidə, bəzi hallarda isə ona təsir edə bilərik. Obyektin belə təsviri bizdə tam aydın təsəvvür yaratmasa da, ətrafımızda çoxlu şeylər göstərə bilərik ki, ona uyğun gəlir. Məsələn, zəngli saat bizi səhər tezdən oyadır, onun zəngini dayandırmaq üçün düyməsini basırıq və növbəti səhər yenidən erkən oyanmaq üçün onun zəng dəstəyini fırladırıq. Deməli, "zəngli saat" obyektini qurmaq üçün onun dəstəyini fırladır, zəngini dayandırmaq üçün isə düyməsini basırıq. Biz zəngli saatdan istifadə etdikdə vacib deyil ki, onun daxilində neçə çarxın, yayın və s. olmasını bilək. Bizə yalnız onu qurmağı, vaxtı bilməyi, zəngi idarə etməyi bilmək, demək olar ki, kifayətdir. Bu yanaşma məhz obyekt yanaşmanı bildirir. Ətrafımızda nə qədər obyektlər var ki, biz onların daxili quruluşlarını və iş prinsiplərini bilməyərək onlardan istifadə edirik (televizor, soyuducu, kondisioner, tozsoran, avtomobil və s.). Lakin bu o demək deyil ki, bu obyektlər bizi, ümumiyyətlə maraqlandırmır. Əksinə, hətta daha çox maraqlandırır. Məsələn, əgər "divar" obyektə "mismar" obyekti vurmaq lazımdırsa, biz hökmən bu obyektlərin xassələrini qiymətləndirməliyik. Əgər divar taxtadandırsa, deməli, mismar dəmirdən olmalıdır, tərsinə isə mümkün deyildir. Buradan belə nəticəyə gəlmək olar ki, bütün obyektlərin xassələri var və bu xassələr ixtiyari deyil, müxtəlifdir. Başımız üzərində uçan həşəratların xassələrinə varmırıqsa, onda onlar obyekt deyil, sadəcə həşəratlardır. Əgər biz onları xassələrinə görə fərqləndiririksə, onda başa düşürük ki, başımız üzərində milçək, ağcaqanad, kəpənək və ya arı uçur və bundan asılı olaraq özümüzü müxtəlif cür aparırıq. Obyektin xassələri obyektin özü ilə sıx bağlıdır. Təbəssümsüz sima olmadığı kimi, xassəsiz obyekt də yoxdur.

Bəs bütün bunların proqramlaşdırmaya nə aidiyyəti vardır? Məsələ ondadır ki, biz proqramlaşdırmada müxtəlif obyektlər istifadə edəcəyik. Proqram obyektləri real həyatdakı obyektlərə çox oxşardır – onlar daxili quruluşları və hərəkətləri ilə bir–birindən fərqlənir. Kompüterin ekranında gördüyümüz hər şey obyektdir. Hər bir pəncərə, hər bir nişan, hər bir idarəetmə obyekti, hər bir menyu bir obyektdir. Proqramlaşdırmanın inkişaf tarixinə nəzər yetirsək görərik ki, 25–30 il bundan əvvəl proqramçı böyük zəhmət hesabına bu obyektləri özü yaradırdı və əksər hallarda məsələnin mürəkkəbliyindən asılı olaraq bu işi bir yox, bir neçə proqramçı kollektivi yerinə yetirirdi. Prosedurlu proqramlaşdırma dövründə proqramlar belə yaradılırdı. Bu proqramların sətirləri bir neçə minlərlə kodlardan ibarət olurdu.

Obyektyönlü proqramlaşdırma texnologiyasının yaradılması proqramlaşdırmada bir çox problemləri asanlıqla həll etməyə imkan yaratdı. Əgər biz obyektlərlə işləyəcəyiksə, bu o demək deyildir ki, bütün obyektləri biz özümüz yaratmalıyıq. Cəmiyyətdə çox qədimdən əmək bölgüsü mövcuddur. Məsələn,

inşaatçının bilməsi vacib deyildir ki, onlar üçün kərpici kim və necə hazırlayır. Onların vəzifəsi inşaat materiallarını almaq və ev tikməkdir. Başqa misal. Hamıya məlumdur ki, alma ağacı toxumdan əmələ gəlir. Lakin, kim öz bağında alma toxumu əkir? Alma ağacı yetişdirmək üçün üç illik şitil əkilir. Bağban üçün şitil obyektdir ki, bu obyektin müxtəlif xassələri vardır və bu xassə almanın növündən ibarətdir. Toxum isə onun üçün obyekt deyil, çünki bağbanın nöqteyi–nəzərincə, onun heç bir xassəsi yoxdur. Belə ki, toxumdan yalnız cır alma bitəcəkdir. Yaxşı növ alma almaq üçün onu calaq etmək lazımdır.

Kərpiclər hamısı eynidir, lakin onlardan müxtəlif tikili qurmaq olar. Hazır mənzil plitələrindən (paneldən) isə teatr, stadion yox, yalnız yaşayış mənzili inşa etmək olar. Ona görə də proqramlaşdırmada da biz proqramların qurulmasını elə obyektlərdən başlamalıyıq ki, onlar bizim imkanlarımızı məhdudlaşdırmasın və hər dəfə bizi hər şeyi yenidən başlamağa məcbur etməsin. Delphi–də gələcək obyektləri yaratmaq üçün çoxlu tədarük görülmüşdür ki, onlara *komponentlər* deyilir. Müəyyən bir işi yerinə yetirmək üçün biz bu uyğun komponentlərdən hər hansı birini seçəcəyik. Yuxarıda qeyd etdiyimiz kimi, Windows əlavələrinin bir çoxu təyinatından asılı olmayaraq tamamilə eyni formaya malik olur, çünki onlar eyni komponentlərdən yaradılmışdır. Lakin, bunu kompüter oyun proqramları haqqında demək mümkün deyildir. Onlar bir–birindən həm forma, həm də idarəetmə üsullarına görə kəskin fərqlənir. Demək olar ki, kompüter oyunları "toxumdan yetişdirilmişdir".

11.6.1.2. Obyektlərin xassə və metodları

Yuxarıda qeyd etdik ki, daxili quruluşlarını bilməsək də, biz, çoxlu məişət cihazlarından istifadə edirik. Ən pisi isə odur ki, vintaçanla televizoru qurdalayıb onu tamamilə sıradan çıxara bilərik. Lakin, biz televizorun bəzi xassələrinə təsir göstərə bilərik. Televizorun əsas xassəsi televerilişləri ekranda təsvir etməkdən ibarətdir. Hansı proqramın göstərilməsi haqqında məlumat isə onun daxilində saxlanır. Bunu televizorun necə etdiyini biz bilmirik və bilmək də istəmirik. Bizə lazımdır ki, veriliş xoşumuza gəlmədikdə proqramı çox asanlıqla dəyişə bilək. Bunun üçün isə müxtəlif vasitələr ola bilər. Məsələn, biz proqramı televizorun üzərində yerləşən düymələrlə və ya məsafədən idarəetmə pultu ilə dəyişdirə bilərik. Nəhayət, bizi heç bir proqram maraqlandırmazsa, onda televizora maqnitofon qoşub videofilmlərə baxa bilərik. Lakin bunun üçün televizorun videomaqnitofona qoşula bilmə xassəsi – xüsusi yuvası olmalıdır.

Kompüter proqramlarının obyektlərinin də xassələri mövcuddur. Bunlar elə parametrlərdir ki, biz proqramı yaratdıqda onları seçə və ya proqramı işlətdikdə isə onları dəyişdirə bilərik. Obyektin digər quruluşları bizim üçün bağlıdır və sadəcə olaraq, lazım olmadan, bizim nə isə etməyə imkanımız yoxdur. Obyektin xassələrini yoxlamaq və dəyişdirmək üçün xüsusi prosedurlar – *metodlar* istifadə olunur ki, onlar özləri də obyektin tərkibində olur. Obyektin xassələri ilə nə isə etmək üçün bunu sadəcə olaraq obyektin özündən soruşmaq lazımdır: obyekt onları bizə xəbər verəcək və ya dəyişdirəcəkdir.

Televizorun o qədər də çox xassələri yoxdur. Proqram obyektlərinin isə xassələri daha çoxdur. Məsələn, ekranda görünən düymənin ölçüsü, koordinatı, rəngi, adı və digər xassələri vardır. Biz yalnız özümüzə lazım olan xassələrlə işləyə bilərik. Digər xassələri isə proqramı yazdıqda bir dəfə müəyyənləşdirib, sonra onu tamamilə unuda bilərik.

Bir proqramda istifadə olunan obyekti digər proqramlarda istifadə etmək olar. Bu zaman:

- yeni obyektin əlavə edilməsi proqramda mövcud olan digər obyektlərin işini pozmur;
- proqramda digər obyektlərin varlığı yeni əlavə edilən obyektin özünü necə aparmasına təsir etmir.

Fərz edək ki, proqramçı tank döyüşlərini təsvir edən proqram – oyun yaradır. Bu oyunda çoxlu tank iştirak edir və baxmayaraq ki, hər bir tankın hərəkəti eyni bir prosedurla yazılır, hər bir tank müstəqil obyektdir və proqram onların hər birini fərdi idarə edir. Bu obyektlərin hər birinin aşağıdakı xassələri mövcuddur:

Xassə	**Fərq**
Təsvir	Hamısı eynidir
Bort nömrəsi	Hamısı müxtəlifdir
Komandirin soyadı	Hamısı müxtəlifdir
Texniki vəziyyəti	Oyunun başlanğıcında hamısı eyni, sonra isə fərqlənir
Döyüş dəsti	Oyunun başlanğıcında hamısı eyni, sonra isə müxtəlifdir
Ekranda vəziyyəti	Hamısı müxtəlifdir
Atəş səsi	Hamısı eynidir
Hərəkət zamanı səsləri	Hamısı eynidir

İndi isə təsəvvür edək ki, proqramçı oyuna tankdan başqa top və zirehli maşın daxil etmək istəyir. Bu o deməkdirmi ki, proqramçı yeni sinif obyektlər üçün bütün prosedurları yenidən yazmalıdır? Əsla yox: bəzi xassələri dəyişmək tamamilə kifayətdir. Məsələn belə:

Xassə	**Tank**	**Top**	**Zirehli maşın**
Qiyməti	230	40	50
Sürəti	35	0	55
Zirehi	110	0	40
Atəş sürəti	1	2	24
Təsviri	tank.bmp	art.bmp	btr.bmp
Atəş səsi	tank.wav	art.wav	btr.wav

11.6.1.3. Hadisələr və onların emalı

Sizin diqqətinizi zəngli saatın öz–özünə – insanın yatdığı zaman – zəng vurmasına yönəltmək istəyirik. Zəngli saat üçün bu amil *hadisə* kimi göstərilə bilər. Müəyyən hadisələrə reaksiya vermək xassənin müxtəlifliyidir. Hadisə baş verdikdə onun *emalı* icra olunur – zəngli saat üçün bu zəngin qoşulmasıdır.

Kompüter proqramlarının obyektləri də hadisələrə reaksiya verir. Hadisə baş verdikdə avtomatik olaraq xüsusi metod – hadisə *emaledicisi* işə düşür. Adətən, müxtəlif hadisələrə müxtəlif reaksiyalar verilir, lakin tamamilə mümkündür ki, bir neçə hadisəyə eyni bir emaledici uyğun gəlsin.

Hadisələr vasitəsilə proqramla istifadəçi arasında qarşılıqlı əlaqə yaranır. Belə ki, biz mausu hərəkət etdirdikdə və ya klavişi basdıqda bu hadisə kimi qeyd olunur və metod – emaledici vasitəsilə icra olunur. İstifadəçi ilə əlaqədar hadisələr istifadəçi hadisələri adlanır. İstifadəçi hadisələrindən başqa, proqram hadisələri də mövcuddur. Məsələn, tank mina üzərinə çıxdıqda "mina" obyekti emaledicisi – metodu işə düşür. Bu alt proqramda mina öz vəziyyətini dəyişir (yox olur), tank obyekti üzərində isə "partlama" hadisəsi baş verir. Nəticədə "tank" obyekti emaledicisi metodu işə düşür. Bu alt proqram zədələnmə dərəcəsini hesablayır və proqram "tank" obyektini ya məhv, ya da hərəkətsiz edir – onda "tank" obyektinin "sürət" xassəsi *0* qiyməti alır. Güllədən atəş, tankın partladılması və s. kimi hadisələr üçün emaletmə metodlarını proqramçı özü yazır. Bunlar ciddi proqramlarda əmrlər düymələri kimi tipik obyektlərə də aiddir. Bu düymələr üçün əsas hadisə düymələrin basılmasıdır. Düymələrin basılmasının nəticəsi isə ən müxtəlif hadisələr ola bilər. Bu reaksiyanı müəyyən etmək üçün proqramçı düyməbasma hadisə emaledicisi prosedurunu özü tərtib etməlidir.

Yenidən tank döyüşü proqramına qayıdaq. Bu proqramda tank bütün əməliyyatları "düşünmədən" icra edir. Tank bilmir ki, onun qarşısında müqavimət var, mina var – o, tamamilə kor və kardır. Lakin, bütün bunları onun əvəzinə proqramçı bilir və o, izləyir ki, tank mina üzərinə çıxmasın, düşmən gülləsinə tuş gəlməsin. Bəs necə etmək olar ki, bütün bunları tank özü etsin, daha "ağıllı" olsun? Təəssüf ki, baxılan hal üçün bu mümkün deyildir. Çünki, burada tankın xassələrini sazlamaq üçün heç nə yoxdur. Bir halda ki, xassə yoxdur, demək hadisə də yoxdur, çünki, hər bir obyekt üçün mümkün hadisə elə xassələrin müxtəlifliyidir. Hadisə yoxdursa, demək ona reaksiya da yoxdur.

Müasir proqramlar isə belə işləmir. Məsələn, Windows əməliyyat sistemini götürək – axı bu da proqramdır. O, kompüter qoşulan kimi işə düşür və kompüter söndürülənədək öz işini görür. Haradasa, Windows sisteminin dərinliklərində, ekran obyektlərində və idarəetmə elementlərində baş verən bütün dəyişiklikləri izləyən bir proses daim işləyir. Nəticədə, sistem mausun hərəkətinə, onun və ya klaviaturanın klavişlərinin basılmasına reaksiya verməyə həmişə tam hazır olur.

11.6.2. Sinif və obyekt

Yuxarıda qeyd etdik ki, obyektyönlü proqramlaşdırmada proqramçı çoxlu siniflər üzərində əməliyyat aparır. *Sinif* – xassə, metod və hadisələr yığımıdır, başqa sözlə, sinif – metod, xassə və hadisələrdən ibarətdir və bunlar onun mükəmməl işləməsini təmin edir. Yəqin ki, Sizlərdən hər biriniz Windows sistemində heç olmazsa, bir dəfə hər hansı düyməni basmışsınız. Həmin düymə məhz

- *xassəyə* (rəngi, ölçüsü, üzərində yazı, yazının şrifti və s.),
- *hadisəyə* (düymənin basılması),
- *metodlara* (mətnin ekrana çıxarılması, pəncərənin bağlanması və s.)

malikdir.

Xassə, metod və hadisələrin işini tam bir vahid kimi araşdıraq. Yenə də düymə sinfinə baxaq. Bu sinif aşağıdakı minimal yığıma malik olmalıdır:

- *xassələr*:
  - düymənin sol mövqeyi (X);
  - düymənin yuxarı mövqeyi (Y);
  - düymənin eni;
  - düymənin hündürlüyü;
  - düymənin sərlövhəsi;

- *metodlar*:
  - düyməni yaratmaq;
  - düyməni məhv etmək;
  - düyməni çəkmək;

- *hadisələr*:
  - düymə;
  - düymənin sərlövhəsi dəyişmişdir.

Bu obyekt tam bir vahid kimi işləyir. Məsələn, Siz, düymənin sərlövhəsini dəyişdirdiniz. Obyekt dərhal "*düymənin sərlövhəsi dəyişdi*" hadisəsini yaradır. Bu hadisəyə uyğun olaraq "*düyməni çəkmək*" metodu çağrılır. Bu metod düyməni obyektin xassələrində göstərilmiş mövqedə yerləşdirir və onun üzərində "*düymənin sərlövhəsi*" xassəsində göstərilmiş yazını təsvir etdirir.

Hər bir sinfin iki metodu vardır: "obyekti yaratmaq" və "obyekti məhv etmək". Obyekt yaradıldıqda onun xassələrini yadda saxlamaq üçün yaddaşda yer ayrılır və avtomatik olaraq qiymətlərlə doldurulur. Obyekti məhv etdikdə isə həmin yaddaş boşalır. Obyekti yaradan metod *konstruktor* (`constructor`), obyekti məhv edən metod isə *destruktor* (`destructor`) adlanır. Obyektin özünün yaradılması prosesi isə *inisializasiya* adlanır.

Object Pascal dilində obyekt mürəkkəb tipdir. Bu o deməkdir ki, hər hansı dəyişəni "obyekt" tipli elan etmək olar (necə ki, hər hansı dəyişən "ədəd" və ya "sətir" tipli elan edilirdi). İndi obyekt və sinif haqqında bildiklərimizi birləşdirək. Sinif tipini elan edək. Qəbul edək ki, `Obyekt1 – Düymə` tiplidir Obyektin yaradılması haqqında "proqramı" öz sözlərimizlə yazaq:

```
Proqramın başlanğıcı;
Dəyişənlər
     Obyekt1 : Düymə;
Kodun başlanğıcı
     Obyekt1:=Düymə.Obyekti_yaratmaq;
     Obyekt1.Sərlövhə:='Delphi';
     Obyekt1.Obyekti_məhv_etmək;
Kodun sonu.
```

Obyektin xassə və metodlarına müraciət etmək üçün obyektin adı hökmən göstərilməlidir:

```
Obyekt_tipli_dəyişənin_adı.xassə
```

və ya

```
Obyekt_tipli_dəyişənin_adı.metod.
```

Məhz bu qayda ilə biz obyektin sərlövhəsini `Delphi` adlandırdıq, eyni qayda ilə konstruktora (`Obyekti_yaratmaq`) və destruktora (`Obyekti_məhv_etmək`) müraciət etdik.

Sinif əsasında obyekt tipli yeni dəyişənlər yaratmaq mümkündür. Aşağıdakı "proqram" iki yeni düymə yaratmağa imkan verir:

```
Proqramın başlanğıcı;
Dəyişənlər
     Obyekt1 : Düymə;
     Obyekt2 : Düymə;
Kodun başlanğıcı
     Obyekt1:=Düymə.Obyekti_yaratmaq;
     Obyekt2:=Düymə.Obyekti_yaratmaq;

     Obyekt1.Sərlövhə:='Pascal';
     Obyekt2.Sərlövhə:='Delphi';

     Obyekt1.Obyekti_məhv_etmək;
     Obyekt2.Obyekti_məhv_etmək;
Kodun sonu.
```

Bu "proqramda" iki dəyişən `Düymə` tipli elan edilir. Sonra onlar inisiallaşdırılır və sərlövhələri dəyişdirilir. Nəticədə biz bir obyektdən müxtəlif

sərlövhəli iki obyekt aldıq. Hər iki düymə sərbəst işləyir, biri digərinə mane olmur, çünki, onlar üçün ayrı yaddaş sahələri ayrılmışdır.

Yaradılmış obyekt `Free` metodu ilə məhv edilməlidir. Əgər obyekt daha lazım deyildirsə, onda onu pozmaq lazımdır:

```
Obyekt1.Free;
```

Beləliklə, *obyekt sinfin nüsxəsidir*. Sinfin köməyi ilə işləyəcəyimiz obyektin mahiyyəti təsvir olunur. Məsələn, bu – siniflərin xassə, metod və ya hadisələrinin təsviri ola bilər. Obyekt nüsxədir. Forma üzərində düymə yerləşdirmək üçün, Siz, sinfi elan etməlisiniz – xassə, metod və hadisələri yaratmalısınız, düyməni forma üzərində yerləşdirdikdə isə onun nüsxəsi, başqa sözlə, obyekt yaradılır. İkinci düyməni yerləşdirdikdə daha bir nüsxə, yəni daha bir obyekt yaranır. Siniflə obyekt arasında fərq bunlardan ibarətdir.

11.6.3. Obyektyönlü proqramlaşdırmanın əsas prinsipləri

Obyektyönlü proqramlaşdırmanın əsas prinsipləri aşağıdakılardır:

- **irsi mənimsəmə;**
- **polimorfizm;**
- **inkapsulyasiya.**

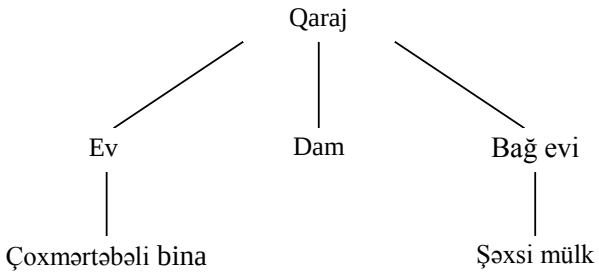
11.6.3.1. İrsi mənimsəmə

İrsi mənimsəmə ondan ibarətdir ki, mövcud əcdad–obyektdən varis–obyekt əmələ gəlir. Bu zaman varis–obyekt əcdad–obyektdən onun bütün sahə, xassə və metodlarını irsən alır. İrsi alınmış bu sahə, xassə və metodları sonralar olduğu kimi və ya modifikasiya etməklə istifadə etmək olar. Digər tərəfdən, sadəcə olaraq irsi mənimsəmənin mənası yoxdur, ona görə də yeni obyektə onun xüsusiyyətini və funksionallığını müəyyənləşdirən yeni elementlər (sahə və metodlar) əlavə edilir. İrsi mənimsəməni qavramaq üçün misala baxaq. Fərz edək ki, Siz, "qaraj" obyektini təsvir etmişsiniz. İndi isə Sizə "ev" obyektini təsvir etmək lazımdır. Əslində qaraj özü də, bir növ, birotaqlı evdir. Hər iki tikili eyni xassələrə – divar, döşəmə, tavan və s. kimi xassələrə malikdir. Ona görə də daha yaxşı olar ki, qarajı əsas obyekt kimi qəbul edərək ondan ev əmələ gətirək. Bunun üçün yeni "ev" obyekti yaradaraq, onun "qaraj" obyektindən əmələ gəlməsini təsvir etmək lazımdır. Bu yeni obyekt qarajın bütün xassələrini qəbul edəcəkdir. Bu isə o deməkdir ki, artıq divarlar, döşəmə, tavan vardır və yalnız pəncərə və interyer əlavə etmək lazımdır. Artıq Sizin iki obyektiniz vardır. Bu qaydanı tətbiq edərək it üçün dam yaratmaq olar. Bunun üçün "dam" obyekti yaradaraq onun "qaraj" obyektindən əmələ gəlməsini təsvir etmək lazımdır. Bu halda isə, qarajın ölçülərini kiçiltmək, girişi dəyişməklə onu it damına çevirmək olar. Nəticədə, şəkil 11.1 – də göstərilmiş obyektlərin ağacvarı irsi mənimsəmə iyerarxiyası alınacaqdır.

Bu misaldan iki anlayış irəli gəlir: "*əcdad*" və "*varis*". *Əcdad* – elə sinifdir ki, ondan digər siniflər əmələ gəlir. *Varis* siniflər isə digər siniflərdən əmələ gəlir. Məsələn, qaraj – ev, it damı, bağ evi və s. üçün əcdaddır. Ev, it damı, bağ evi və s. isə qarajın varisləridir.

Varis – əcdadın xassələrini mənimsəyir, ona görə də o bilməlidir ki, əcdad hansı xassələrə malikdir. Əcdad isə öz varisinin xassələrini bilməyə bilər, çünki, o, yeni sinfə hansı xassə və metodların əlavə edildiyini bilmir. Məhz obyektyönlü proqramlaşdırmada bu irsi mənimsəmə metodu qəbul edilmişdir.

Varis–obyektdə əcdad–obyektin hər hansı bir elementini pozmaq olmaz. Öz növbəsində, yeni obyektdən növbəti obyekt yaratmaq olar, nəticədə obyektlər ağacı və ya siniflər iyerarxiyası əmələ gəlir. Bu ağacın başlanğıcında **`TObject`** baza sinfi dayanır. Şəkil 10.4 – də Delphi obyektlərinin iyerarxiya fraqmenti göstərilmişdir. `TObject` baza sinfi bütün obyektlər üçün daha ümumi olan elementləri, məsələn, obyektləri yaratmaq və obyektləri pozmaq kimi əməliyyatları yerinə yetirir. Bu və ya digər obyekt obyektlər ağacında baza sinfindən nə qədər uzaqda yerləşərsə, bir o qədər səciyyəvi olur.



Şəkil 11.1. "Qaraj" obyektinin iyerarxiyası

İrsi mənimsəmənin cəlbediciliyi ondan ibarətdir ki, əgər hər hansı bir obyekt artıq müəyyən edilmiş və *sazlanmışdırsa*, həmin obyekti başqa proqramlarda da istifadə etmək olar. Bu zaman ola bilər ki, yeni məsələ əvvəlki məsələdən fərqlənsin, verilənlərin və onların emal edilmə metodlarının modifikasiya edilməsi zərurəti ortaya çıxsın. Onda proqramçı dilemma qarşısında qalır: o, yeni obyekt yaratsın, yoxsa irsi mənimsəmə mexanizmini tətbiq etməklə əvvəlki işin nəticələrini istifadə etsin. Birinci yol səmərəli deyil, çünki çox zəhmət tələb edir. İkinci halda isə işin müəyyən hissəsi artıq yerinə yetirildiyi üçün, yeni proqramın yaradılmasına az vaxt sərf olunacaqdır. Özü də bu zaman proqramçının əcdad–obyektin hissələrinin necə həyata keçirilməsini bilməsinə ehtiyac yoxdur.

11.6.3.2. Polimorfizm

Polimorfizmin mahiyyəti ondan ibarətdir ki, müxtəlif obyektlərin metodları eyniadlı, lakin müxtəlif məzmunlu ola bilər. Bu, sinif–varisdə əcdad–obyektin metodunu yenidən müəyyənləşdirməklə əldə edilir. Nəticədə isə əcdad və varis

özlərini tamamilə başqa cür aparırlar. Qaraj misalına qayıdaq. Müasir qarajların qapıları bir qayda olaraq yuxarı qaldırılmaqla açılır. Ev isə qarajdan əmələ gəldiyi üçün, onun da qapıları yuxarı açılacaqdır. Bəs nə etməli? Sadəcə olaraq qapıların açılması üçün yazılmış proseduru ev üçün yenidən yazmaq lazımdır ki, evin qapıları yana açılsın. Bu zaman ev – qarajın bütün xassələrinə malik olacaq, qapıların açılması üçün isə öz prosedurunu təqdim edəcəkdir.

Qapıların açılması üçün proseduru dəyişməyin mümkün olması üçün isə bu prosedur qarajda "*virtual*" (`virtual`) elan olunmalıdır. *Virtual prosedur* o deməkdir ki, yaradılacaq obyektdə o dəyişdirilə bilər. Polimorfizmdə başqa bir metod – *dinamik metod* (`dynamic`) da tətbiq oluna bilər. Hər iki metodun nəticələri eynidir, fərq ondadır ki, `dynamic` metod yavaş işləyir, lakin proqram kodları az yazılır, `virtual` metod isə proqramın sürətlə işləməsi üçün daha optimal kodlar yaradır. **Borland** firması `virtual` metodu tətbiq etməyi tövsiyə edir.

Polimorfizmin daha bir xüsusiyyəti vardır. Evin divarlarında tablolar ola bilər, qarajda isə belə tablolara ehtiyac yoxdur. Bu halda əcdadın metodlarına müraciət oluna bilər. Misala baxaq:

```
Qarajın divarlarını yaradan prosedur
   Başlanğıc
     Divarları yaratmaq
   Son


Evin divarlarını yaradan prosedur
   Başlanğıc
     Əcdad obyekti çağırmaq
     Divarlara tablo asmaq
   Son
```

Birinci prosedurla qarajın divarları yaradılır, ikinci prosedurla evin divarlarının yaradılmasını yenidən müəyyənləşdiririk. İlk baxışda adama elə gəlir ki, qarajın divarlarını yaratma proseduru itəcək və evin divarları yenidən yaradılacaqdır, əslində isə bu belə deyildir. Sadəcə olaraq əcdad–obyekti çağırmaq (onda əcdad bizim üçün divarları yaradacaq), sonra isə divarlara tablolar asmaq lazımdır.

11.6.3.3. İnkapsulyasiya

İnkapsulyasiya obyektlərin əsas xassəsidir. İnkapsulyasiya ondan ibarətdir ki, obyekt özündə müəyyən hissələri gizlədir ki, obyektin istifadəsi zamanı onlar o qədər də əhəmiyyət kəsb etmir. Proqramlaşdırmada qlobal dəyişənlərdən istifadə etdikdə, proqramçı bu dəyişənlərlə əlaqədar olan verilənləri emal etmək üçün nəzərdə tutulmayan prosedurlardan istifadə etdikdə, səhvlərdən sığorta olunmur. Fərz edək ki, hər hansı bir təşkilatın əməkdaşlarının əmək haqqını hesablayan adi proqram mövcuddur və bu proqramda iki massivdən istifadə

edilir. Massivin birində əmək haqqı kəmiyyətləri, digərində isə əməkdaşların telefon nömrələri saxlanır. Əgər proqramçı bu massivləri səhv salarsa, təsəvvür edirsinizmi nə baş verəcəkdir? Verilənlərin və onları emal edən prosedurların bir obyektdə birləşdirilməsi belə xoşagəlməzliklərdən qaçmağa imkan verir. Məhz inkapsulyasiya sinif daxilində verilənlərin və onları emal edən metodların (alt proqramların) birləşdirilməsidir. Bu o deməkdir ki, sahə, xassə və metodlar sinifdə inkapsulyasiya olunur (birləşdirilir və sinifdə yerləşdirilir). Bu zaman sinif müəyyən funksionallıq əldə edir.

Biz obyekt haqqında təsəvvür əldə etdikdə çoxlu misallar göstərərək onların necə işləməsinin bizi ümumiyyətlə maraqlandırmadığını qeyd etdik (zəngli saat, televizor, avtomobil, digər məişət texnikaları və s.). Proqramçıya da, hər hansı sinfi istifadə etdikdə, bilmək lazım deyildir ki, bu sinif öz imkanlarını necə yerinə yetirir. Sinfi yaratdıqda onu elə layihələndirmək lazımdır ki, bu sinfin əsas işlərini yerinə yetirən metod və xassələr gizli olsun. Bunun üçün xassə və metodlara müxtəlif müraciət hüquqları müəyyənləşdirilə bilər. Belə hüquqlar sinfin **`private`** (*gizli*), **`protected`** (*mühafizə olunan*), **`public`** (*ümumi*) və **`published`** (*aşkar və ya dərc olunmuş*) bölmələrində müəyyənləşdirilir. Sinfin açıq metodları digər siniflər tərəfindən görünür, gizli metodlar isə digər siniflər tərəfindən görünmür.

Ümumiyyətlə, obyektyönlü proqramlaşdırmada sinfin dəyişənlərinin açıq verilməsi məsləhət görülmür, lakin, əgər onlara təsir etmək lazım gələrsə, onda metod və ya xassələrdən (**`property`**) istifadə etmək olar. Başqa sözlə, dəyişənləri oxumaq və ya onların qiymətlərini dəyişdirmək üçün yalnız birbaşa təsir imkanı olmayan metodlardan istifadə etmək lazımdır.

Biz bilərəkdən obyektyönlü proqramlaşdırmanın belə populyar izahına geniş yer verdik. Çünki, bir daha xatırladırıq ki, bizim məqsədimiz məhz obyektyönlü proqramlaşdırmanın xüsusiyyətlərini öyrənməkdən ibarətdir. Biz, obyektyönlü proqramlaşdırma metodlarını nəzəri olaraq, öz sözlərimizlə proqramlaşdırmağı artıq bacarırıq. İndi isə obyektlər, siniflər, onların xassələri, hadisələri və emaletmə metodlarını populyar şəkildə qavradıqdan sonra, onların Object Pascal dilində necə təsvir edilməsini öyrənək.

11.6.4. Siniflər

Sinif yazıların xüsusi tipidir. Yazılardan fərqli olaraq, sinfin tərkibinə nəinki verilənləri təsvir edən sahələr, həm də xassə və metodlar daxil olur. *Sinfin sahələri* yazıların sahələrinə analojidir və obyekt haqqında informasiya saxlamaq üçündür. Sahələri emal etmək üçün nəzərdə tutulmuş prosedur və funksiyalara isə *metodlar* deyilir. *Xassə* isə sahə və metod arasında aralıq vəziyyətdir. Bir tərəfdən xassəni sahə kimi istifadə edərək onlara mənimsətmə

operatoru vasitəsilə qiymət mənimsətmək olar, digər tərəfdən, sinif daxilində xassənin qiymətinə müraciət sinfin metodları ilə yerinə yetirilir.

Sinfin təsviri aşağıdakı strukturdan ibarətdir:

```
Type sinfin adı = class(əcdad–sinfin adı)

    Private    // Bu sözdən sonra obyektin gizli dəyişənləri təsvir edilir

      { private declarations }  bu sətri Delphi yazır

      {  Bu hissədə yalnız TForm1
         obyektinin müraciət edə biləcəyi
         dəyişən və metodlar təsvir edilir  }


    Protected

      { protected declarations }  bu sətri Delphi yazır


        {  Bu hissədə bu obyektin və onun
        varislərinin müraciət edə biləcəyi xassə
        və metodlar təsvir edilir  təsvir edilir  }


    Public   // Bu sözdən sonra obyektin açıq dəyişənləri təsvir edilir

      { public declarations }  bu sətri Delphi yazır

        {  Bu hissədə yeni yaranmış siniflərin
        müraciət edə biləcəyi, mühafizə
        olunan  dəyişən və metodlar təsvir edilir  }


    Published

   { published declarations }  bu sətri Delphi yazır

        {  Bu hissədə yeni komponentin
        Obyektlər inspektorunda əks
        olunacaq xassə və metodları təsvir edilir  }

end;
```

Göründüyü kimi, sinfin strukturu dörd bölmədən ibarətdir. Bütün bölmələr üçün ümumi olan `declarations` – *təsvirlər* – xassə, metod və hadisələrin təsvirlərindən ibarətdir. Sinfin müxtəlif elementləri üçün müxtəlif müraciətetmə hüquqları (görünmə) müəyyən etmək olar ki, bunlar da strukturda xüsusi işçi sözlərlə işarə edilmişdir.

`Private` – bu bölmədə təsvir edilmiş xassə və metodlara yalnız bu obyekt müraciət edə bilər. Digər obyektlər bu metodları çağıra bilməz və bu xassələri yazıb–oxuya bilməz.

`Protected` – bu bölmə *mühafizə* olunan təsvirlərdən ibarət olur. Burada təsvir edilmiş xassə və metodlara yalnız bu obyekt və onun varisləri (bu obyektdən yaranmış və onun xassələrini mənimsəmiş obyektlər) müraciət edə bilər. Kənar obyektlər burada saxlanılan xassə və metodlara müraciət edə bilməz.

`Public` – proqramın istənilən yerindən görünən, hamının müraciət edə biləcəyi təsvirlərdən ibarətdir.

`Published` – bu bölmədə, yeni komponent yaradıldıqda, həmin komponentin Obyektlər inspektorunda əks olunacaq xassə və metodlarının təsvirləri yazılır. Hamının müraciət edə biləcəyi bu aşkar təsvirlərə əlavə olaraq, bu bölmə, yerinə yetirmə vaxtının tipi (*Run–Time Type Information – RTTI*) haqqında informasiya yaradır. Bu informasiyaya görə əlavənin icrası zamanı obyektin elementlərinin bu və ya digər sinfə mənsub olması yoxlanır. `Published` bölməsi həm də əlavə konstruksiya edildikdə obyektlərin xassəsinə müraciət etməyə imkan verir. Obyektlər inspektorunda obyektin dərc olunmuş xassələri görünür. Əgər `Published` yazılmazsa, o susmaya görə təsəvvür olunur və ona görə də sinfin adının göstərildiyi sətirdən sonra yerləşən bütün təsvirlər dərc olunmuş təsvir kimi qəbul ediləcəkdir.

Misal.

```
type
   TForm1 = class(TForm)

     Button1: TButton;
     Edit1: TEdit;

     procedure Button1Click(Sender: TObject);
     procedure Edit1KeyPress(Sender: TObject;
                             var Key: Char);

   private
     { Private declarations }

   public
     { Public declarations }

end;
```

Sinif təsvir edildikdən dərhal sonra, proqramda onun nüsxəsindən, başqa sözlə, sinif tipli dəyişəndən istifadə etmək olar:

```
Var Form1: TForm1;
```

`TForm` əcdad–sinfindən yaradılan `TForm1` varis–sinfin təsvirində iki prosedurun – metodların yalnız adları sadalanır, onların kodları isə `implementation` bölməsində göstərilməlidir. `TForm1` sinfi `TForm` sinfinin bütün xassələrini irsi qəbul edir, inkapsulyasiya prinsipinə görə isə strukturda `TForm` sinfinin təsvirinə ehtiyac qalmır və yeni sinfə iki metod əlavə olunur ki, nəticədə sinif funksionallıq əldə edir.

11.6.4.1. Sahələr

Sinfin *sahəsi* sinfə daxil olan verilənlərdən ibarətdir. Sahə adi dəyişən kimi təsvir olunmaqla istənilən tip ola bilər. Sahələr adətən xüsusi mühafizə olunan təsvirlər bölməsində (`private`) təsvir edilir.

Misal.

```
Type TNewClass=class(TObject)
Private
   FCode: integer;
   FSign: char;
   FNote: String;
end;
```

Baxılan bu misalda, yeni `TNewClass` sinfi `TObject` baza sinfi əsasında yaradılır və əlavə olaraq o, üç yeni `FCode`, `FSign` və `FNote` sahələrini alır. Bu sahələr uyğun olaraq tam, simvol və sətir tiplidir. Xatırladaq ki, qəbul olunmuş qaydaya görə, sahələrin adı həmişə **`F`** (*field* – "sahə" sözündən) hərfi ilə başlamalıdır.

Əgər yeni sinif `TObject` baza sinfi əsasında yaradılırsa, onda **`class`** sözündən sonra onun adını yazmamaq olar, qalan hallarda isə əcdad–sinfin adı göstərilməlidir.

Misal.

```
Type TNewClass=class
Type TBtnSound=class(TButton)
```

Yeni siniflər yaradıldıqda varis–obyekt əcdad–obyektin bütün sahələrini irsi mənimsəyir və bu zaman bu sahələrin pozulması və ya onların yenidən təyin edilməsi mümkün deyildir. Yeni sahələr isə istənilən qədər əlavə oluna bilər və beləliklə, iyerarxiyaya görə hər hansı bir sinif `TObject` sinfindən nə qədər uzaqda yerləşərsə, onun bir o qədər çox sahələri olur.

11.6.4.2. Xassələr

Sinfin sahələrində bu sinfin informasiyaları saxlanır. Sahələr adətən mühafizə olunan `Private` bölməsində yerləşdiyi üçün, proqram yolu ilə onlara birbaşa daxil olmaq mümkün deyildir. Proqram yolu ilə sahələrin qiymətlərinə müraciət etmək üçün xassələrdən istifadə olunur. Hər bir xassəyə onun qiymətlərindən ibarət sahələr və bu sahəyə daxil olmaq üçün iki metod uyğun gəlir. Proqram yolu ilə daxil oluna bilən xassələr hamının müraciət edə biləcəyi (`Public`) və dərc olunmuş (`Published`) bölmələrdə təsvir olunur, əks halda isə onları mühafizə olunan (`Protected`) bölmədə yerləşdirirlər. Xassənin formatı belədir:

property *xassənin adı* **:** < *xassənin tipi* > < *spesifikatorlar* >**;**

Xassənin əsas spesifikatorları onun qiymətlərini uyğun olaraq *oxumağa* və *yazmağa* imkan verən `read` və `write` spesifikatorlarıdır. Xassənin təsvirində bu spesifikatorlardan, heç olmazsa, birini göstərmək lazımdır. Spesifikatordan sonra sahənin və ya metodun adı göstərilməlidir. Əgər xassənin qiyməti birbaşa öz sahəsinin qiyməti ilə əlaqədardırsa, onda spesifikatorda bu sahənin adı göstərilir. Hər iki spesifikator istifadə edildikdə xassənin formatı belə olur:

property *xassənin adı***:***xassənin tipi* **read** *sahənin adı* **write** *sahənin adı***;**

Oxumaq metodu parametrsiz funksiyadır və onun tipi oxunan xassənin tipi ilə eynidir. *Yazmaq* metodu tipi xassənin tipi ilə müəyyənləşdirilən bir parametrdən ibarət prosedurdur.

Misal.

```
type TNewClass=class(TObject)

private
  FCode: integer;
  FSign: char;
  FNote: String;

Published
  property Code: integer read FCode write FCode;
  property Sign: char    read FSign write FSign;
  property Note: string  read FNote write FNote;

end;
```

Yeni əlavə olunmuş `FCode`, `FSign` və `FNote` sahələri mühafizə olunan bölmədə yerləşdiyi üçün proqramçı bu sahələrə daxil ola bilmir. Lakin, tamədədli `Code`, simvollu `Sign` və sətir tipli `Note` xassələri dərc olunmuş bölmədə yerləşdiyi üçün, proqramçı bu xassələrdən istifadə etməklə, həmin sahələrə müraciət edə bilər. Proqramçılar bu xassələrə həmçinin Obyektlər inspektorundan müraciət edə bilər.

11.6.4.3. Metodlar

Metod sinfin elementi olan alt proqramdan (prosedur və ya funksiya) ibarətdir. Metodun təsviri modulun adi alt proqramının təsvirinə oxşayır. Metodun sərlövhəsi sinfin təsvirində, onun öz kodu isə realizasiya (**implementation**) bölməsində yerləşir. Realizasiya bölməsində metodun sərlövhəsi sinfin tipini göstərən tərkibli addan ibarət olur.

Misal.

```
interface
   ...
type
   TForm1=class(TForm)
     Button1:TButton;
     procedure Button1Click(Sender: TObject);
   end;
...
implementation
...
procedure TForm1.Button1Click(Sender: TObject);
begin
  close;
end;
```

Burada **interface** bölməsində `Button1Click(Sender:TObject)` metodu təsvir olunur və `implementation` bölməsində bu metodun adında `TForm1` sinfinin adı göstərilməklə, onun kodu yerləşir; `Button1` düyməsi basıldıqda metod `Form1` formasını bağlayır.

Sinifdə elan edilmiş metod, bu metodun növündən asılı olaraq, müxtəlif üsullarla çağrıla bilər. Metodun növü xüsusi modifikatorla müəyyən edilir. Bu modifikator sinfin təsvirində metodun adından sonra, aralarında nöqtəli–vergül simvolu qoyulmaqla göstərilir. Bu modifikatorlardan bəziləri aşağıdakılardır:

- **Static** –*statik metodlar*;
- **Virtual** –*virtual metodlar*;
- **Dynamic** –*dinamik metodlar*;
- **Override** –*yenidən təyinetmə metodları*;
- **Message** –*məlumat emaletmə metodları*;
- **Abstract** –*mücərrəd metodlar*.

Static (*statik*) *metodlara* sadə prosedur və funksiyalar aiddir. Susmaya görə sinifdə elan edilmiş bütün metodlar statik metodlardır və onlar adi alt proqram kimi çağrılır.

Virtual (*virtual*) *metodlar* varis obyektlərdə yenidən müəyyənləşdirilə bilər (qarajın qapısının ev qapıları kimi açılması).

Dynamic (*dinamik*) *metodlar* – virtual metodlara analojidir, sürəti azdır, lakin proqram kodları daha yığcamdır.

Message (*məlumat emaletmə*) *metodları* – Delphi bu metodlar vasitəsilə əməliyyat sisteminin hadisələrinə reaksiya verir.

Abstract (*mücərrəd*) *metodlar* obyektdə yalnız elan olunur, lakin realizasiya olunmurlar.

Metod müəyyən qədər və ya tamamilə əcdad–metodu üstələyə bilər. Əgər varis–metod əcdad–metodu modifikasiya edərsə, onda əcdad–metodun çağrılması üçün **Inherited** metodu istifadə edilir.

Misal.

```
TBtnSound=Class(TObject)
protected
  procedure Click; override;
end;
…
procedure TBtnSoundClick;
begin
  beep;
  inherited;
end;
```

Bu misalda, `BtnSound` düyməsi üçün `Click` (mausun düyməsini basmaq) metodu yenidən müəyyənləşdirilir. Nəticədə düymə basıldıqda adət etdiyimiz əməliyyatdan başqa, sistem siqnalı (`beep`) da səslənəcəkdir. Bu misalda polimorfizm prinsipi öz əksini tapmışdır.

Metodların yenidən müəyyənləşdirilməsinə hadisələrin yenidən müəyyənləşdirilməsi də deyirlər. *Hadisə* bu və ya digər hadisəyə reaksiyanı təmin etmək üçün prosedur tipli xassədən ibarətdir. Bu xassəyə (hadisəyə) qiymətin verilməsi hadisə baş verdikdə metodun göstərilməsini bildirir. Uyğun metodlara *hadisə emalediciləri* deyilir.

Hadisənin növündən asılı olaraq Delphi–də müxtəlif tip hadisələr mövcuddur. Xəbərdaredici hadisələr üçün xarakterik olan ən sadə tip `TNotifyEvent` tipidir. Bu tip belə təsvir olunur:

```
type TNotifyEvent=
      procedure(Sender:TObject) of object;
```

Bu təsvirdəki **Sender** parametri hadisənin obyekt–mənbəyini göstərir. Daha mürəkkəb tipli hadisələrdə `Sender` parametri ilə yanaşı digər parametrlər də mövcud olur.

Hadisələr xassələr olduğu üçün, əlavənin yerinə yetirilməsi zamanı onların qiymətlərini, başqa sözlə, obyektin eyni bir hadisəyə reaksiyasını dinamik dəyişmək olar. Bu zaman, əgər hadisələrin tipləri uyğun gələrsə, bir obyekt

emaledicisini digər obyektə və ya hadisəyə təyin etmək olar. Bu, sinif göstəricisi ilə təmin edilir.

`Sender` aşkar parametrindən başqa, metoda onun sinfinin nüsxəsini çağıran göstərici də verilir. Bu göstərici **`Self`** operatorudur.

Obyektlərin yaradılması və pozulması üçün təyin olunmuş metodlara, uyğun olaraq, *konstruktor* (`constructor`) və *destruktorlar* (`destructor`) deyilir.

Hadisə və metodların təfərrüatı ilə öyrənilməsinə sonrakı bölmələrdə də baxacağıq.

11.6.5. Yerinə yetirmə vaxtının tipi haqqında informasiya

Obyektlər özlərinin tipi və irsi mənimsəmələri haqqında *RTTI* informasiyaya malikdir. Belə informasiyanı proqramın icrası zamanı əldə etmək mümkündür. Bu informasiyanı obyektin bu və ya digər tipə mənsub olmasını yoxlamaq üçün istifadə etmək olar. Onun əhəmiyyəti ondan ibarətdir ki, hər bir obyekt üzərində onun tipindən asılı olaraq yalnız müəyyən əməliyyatlar yığımını yerinə yetirmək olar.

Yuxarıda qeyd etdik ki, əksər metodlara onları çağırdıqda **`TObject`** tipli **`Sender`** parametri verilir. Bu parametrlə əməliyyat yerinə yetirmək üçün, məsələn, xassəyə qiymət vermək və ya metodu çağırmaq üçün, əvvəlcə onun tipini üzərində əməliyyat yerinə yetiriləcək obyektin tipinə çevirmək lazımdır. Belə çevirmə *aşkar* və *qeyri–aşkar* yolla yerinə yetirilir.

Object Pascal dilində tiplərlə işləmək üçün **`is`** və **`as`** operatorları tətbiq edilir. `is` operatoru aşağıdakı ifadədə istifadə olunur:

```
Obyekt is Sinif;
```

Bu ifadə ilə obyektin göstərilən sinfə və ya onun hər hansı varisinə mənsub olması yoxlanır. Əgər obyekt sinfə mənsubdursa, onda bu ifadənin nəticəsi *True*, əks halda isə *False* qiyməti olur.

`as` operatoru bir tipi o biri tipə çevirmək üçün aşağıdakı ifadədə istifadə edilir:

```
Obyekt as Sinif;
```

Bu ifadə ilə obyekt sinfin tipinə gətirilir. Bu *qeyri–aşkar* çevrilmə adlanır.

Misal.

```
procedure TForm1.Button1Click(Sender:TObject);
begin
  if (Sender is TButton) then
  (Sender as TButton).Caption:= TimeToStr(Now);
end;
```

Bu qeyri–aşkar çevirmədə `Button1` düyməsini basdıqda onun üzərində cari vaxt təsvir edilir. Cari vaxtı təsvir edən `Now` funksiyasıdır. Düymə

obyektinə və onun `Caption` xassəsinə daxil olmaq üçün `Sender` parametri istifadə olunur və o, `TButton` tipinə gətirilir (`Sender as TButton`). Bunun üçün əvvəlcə onun bu tipə gətirilə bilməsi (`Sender is TButton`) yoxlanır.

Tiplərin *aşkar* üsulla çevrilməsi isə belə konstruksiya ilə yerinə yetirilir:

```
Tip(Obyekt)
```

Misal.

```
procedure TForm1.Button1Click(Sender:TObject);
begin
  TButton(Sender).Caption:= 'Düymə';
end;
```

Burada, istifadəçi hansı komponent üzərində mausun düyməsini basarsa, onun sərlövhəsində `'Düymə'` yazılacaqdır. Bunun üçün komponentin tipi `TButton` tipinə gətirilir.

11.6.6. Vizual komponentlər kitabxanası

Obyektyönlü proqramlaşdırma texnologiyasının yarandığı ilk vaxtlarda vizuallıq barədə heç kim düşünmürdü, bu proqramçıların arzusu olaraq qalırdı. **Borland** firması o vaxtlar *Object Windows Library* – Windows obyektlər kitabxanası yaratmışdı. Windows üçün vizual örtük yaratdıqdan sonra, **Borland**, obyektyönlü proqramlaşdırma texnologiyasının konsepsiyasını təkmilləşdirmək qərarına gəldi ki, obyektlərlə vizual işləmək mümkün olsun.

Delphi–nin 6–cı versiyasına qədər yalnız vizual komponentlər kitabxanası (*Visual Component Library – VCL*) mövcud idi. 6–cı versiyada yeni *CLX* (*Borland Component Library for Cross Platform*) – kross formalı komponentlər kitabxanası meydana gəldi. *VCL* yalnız Windows sistemi mühitində işləmək üçün nəzərdə tutulduğundan, digər sistemlərdə işləyə bilmirdi.

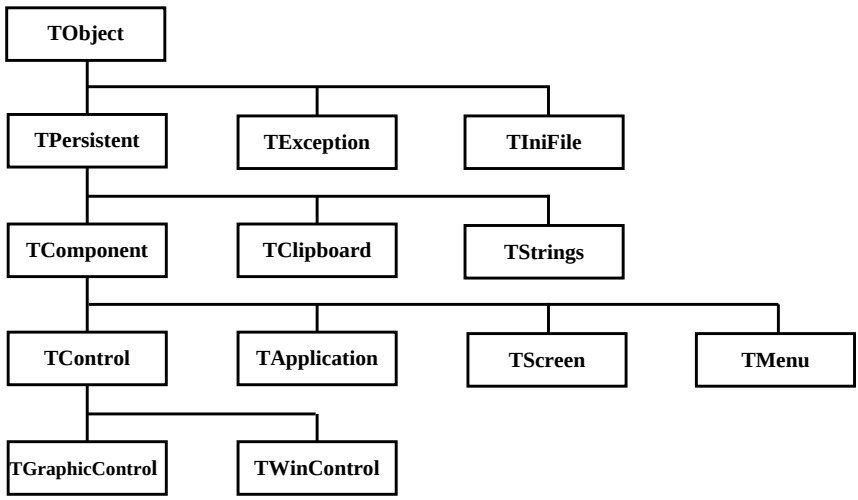
2000-ci ildə **Borland** *Linux* sistemi üçün vizual iş mühiti yaratmaq qərarına gəldi. Bu mühitin əsasını Delphi və *VCL* təşkil edirdi. Bu isə o demək idi ki, Delphi–də yazılmış kod, ona heç bir dəyişiklik edilmədən, *Linux* sistemi altında da problemsiz işləməlidir.

2001–ci ildə yeni *Kylix* sistemi yarandı ki, bu sistem də *Linux* sistemi üçün yazılmış Delphi kodlarını kompilyasiya edə bildi. Bu dəfə vizual komponentlər kitabxanası kimi *CLX* istifadə edilmişdi. Əslində bu elə *VCL* kitabxanası idi, hətta obyektlərin adları da olduğu kimi saxlanılmışdı.

Vizual komponentlər kitabxanası (*Visual Component Library – VCL*) əlavələri çox tez yaratmaq üçün çoxlu siniflərdən ibarətdir. Kitabxananın elementləri Object Pascal dilində yazıldığı üçün, o, Delphi əlavələrini yaradan inteqrallaşdırılmış iş mühiti (**IDE**) ilə birbaşa əlaqədardır. *VCL* əsasən qeyri–vizual komponentlərdən ibarətdir, lakin, burada vizual komponentlər və hətta

mücərrəd `TObject` sinfindən başlayaraq, digər siniflər də vardır. Bütün komponentlər siniflərdir, lakin bütün siniflər komponent deyildir.

VCL–in bütün sinifləri müəyyən iyerarxiyalı səviyyələrdə yerləşməklə siniflər ağacı əmələ gətirir. Obyektin mənşəyi haqqında bilik onun öyrənilməsində böyük əhəmiyyət kəsb edir, belə ki, varis–obyekt əcdad–obyektin bütün elementlərini irsi mənimsəyir. Məsələn, `Caption` xassəsi `TControl` sinfinə aiddirsə, bu xassə onun varislərində də, məsələn, `TButton` və `TCheckBox` siniflərində və onların komponentləri olan `Button` düymələrində və `CheckBox` müstəqil dəyişdiricilərində də olacaqdır. Ən vacib siniflərin iyerarxiyası şəkil 11.2 – də göstərilmişdir.



Şəkil 11.2. Object Pascal dilində siniflərin iyerarxiyası

`TObject` sinfi Object Pascal dilinin bütün siniflərinin ümumi əcdadıdır. O, iyerarxiyanın kökündə dayanır. Bu sinif mücərrəddir və bütün varis–siniflər üçün ümumi olan metodları həyata keçirir. Bu metodlardan ən vacibləri aşağıdakılardır:

- **`Create`** –*obyektin yaradılması*;
- **`Destroy`**–*obyektin pozulması*;
- **`Free`** –`Create` *metodu ilə yaradılmış obyektin pozulması*, bu zaman `Destroy` metodu da çağrılır.

Bir çox siniflərin həm də əcdad–sinifləri olan `TPersistent`, `TComponent` və `TControl` varis–siniflərinin qısa xarakteristikalarını verək. Bu siniflərə ümumi xassə, metod və hadisələr aiddir.

`TPersistent` sinfi xassələri axından yüklənən və axında saxlanan obyektlər üçün mücərrəddir. Axın mexanizmi yaddaşla (adətən disk və operativ) işləmək üçün istifadə edilir. `TObject` sinfinin metodlarına əlavə olaraq `TPersistent` sinfi `Assign` metoduna malikdir ki, bu da obyektin xassə və sahələrini bir obyektdən digərinə verməyə imkan verir.

`TComponent` sinfi bütün komponentlər üçün baza sinfidir. Öz əcdadlarının metodlarına əlavə olaraq, o, elə vasitəyə malikdir ki, onun köməyi ilə komponentlər başqa komponentləri də əldə edə bilir. Nəticədə formada istənilən komponenti yerləşdirdikdə, o, başqa komponentə aid olacaqdır (ən çox formaya). Komponenti yaratdıqda, avtomatik olaraq, ona aid olan bütün komponentlər yaranacaqdır, bu komponenti pozduqda isə onun bütün komponentləri avtomatik pozulacaqdır. `TComponent` sinfinin aşağıdakı xassələri var:

`Components` –*mənsub olan komponentlərin siyahısı*;
`ComponentCount` –*mənsub olan komponentlərin sayı*;
`ComponentIndex` –*mənsub olan komponentlər siyahısında komponentin nömrəsi*;
`ComponentState` –*cari komponentin vəziyyəti*;
`Name` –*komponentin adı*;
`Owner` –*komponentin sahibi*;
`Tag` –*komponentlə birgə saxlanan tam qiymət.*

`TComponent` sinfinin bəzi metodları bunlardır:

`Destroy Components`–*bütün mənsub olan komponentləri məhv etmək*;
`Destroying` –*mənsub olan komponenti onun məhv edilməsi haqqında xəbərdar etmək*;
`FindComponent` –`Components` *siyahısında komponenti tapmaq.*

`TControl` sinfi vizual komponentlər (idarəedici elementlər) üçün baza sinfidir. O, vizual komponentlərin fəaliyyəti, o cümlədən, onların ekranda konturlarının çəkilməsi vasitələrini təmin edir. Bütün vizual komponentlər pəncərə və qrafik tipli olurlar ki, onlar da uyğun olaraq `TWinControl` və `TGraphicControl` siniflərindən əmələ gəlir.

11.7. Modullar

Modullar sabit, dəyişən, prosedur və funksiyaların tiplərini müəyyən edən kitabxanadır. Alt proqramlar kimi modullar da standart və qeyri–standart modullara bölünür. Qeyri–standart modullar istifadəçilərin yaratdıqları modullardır. Lakin, onların hər ikisi eyni qayda ilə tərtib olunduqları üçün, tamamilə eyni struktura malik olur. Delphi–də standart modullar Turbo Pascal dilində olduğu kimi, müəyyən modullar kitabxanasından istifadə etmək məqsədilə tətbiq edilir. İstifadəçi modullarının tətbiqi isə Delphi–də xüsusi

əhəmiyyət kəsb edir. Belə ki, ola bilər ki, proqramçı Turbo Pascal dilində proqram yaradarkən ümumiyyətlə, moduldan istifadə etməsin. Bununla da, sadəcə olaraq, proqramın strukturlaşdırılma və standartlaşdırılma səviyyəsi aşağı düşəcək, proqramın başa düşülməsi və modernləşdirilməsi çətinləşəcəkdir. Lakin, istənilən halda, bu və ya digər səviyyəli proqram tərtib olunacaqdır.

Delphi–də isə *modul* layihələndirmənin ən vacib elementidir və əslində bütün proqramlaşdırma – konstruksiyalaşdırma işləri məhz modulda yerinə yetirilir. Delphi–də layihə yaradan proqramçılar modula çox vaxt "yunit" (`Unit` sözündən) də deyirlər. Proqramçı yunitdə bütün proqramlaşdırma işlərini yerinə yetirir və həmin yunit layihə faylına qoşulur. Delphi ilə ilkin tanışlıqda biz öyrəndik ki, layihə faylının birinci iki sətri aşağıdakı kimi yazılır:

```
Program Project1;

Uses Forms,
     Unit1 in 'Unit1.pas';
```

Sonrakı fəsillərdə, konkret layihələr hazırladıqda, bütün təşəbbüsümüz yalnız modulun yaradılmasından ibarət olacaqdır.

İndi isə hələlik biz modulun xüsusiyyəti və strukturunu mükəmməl öyrənək. Kompilyator modulu onun sərlövhəsinə görə tanıyır. Modulun sərlövhəsi **`Unit`** sözü ilə başlayır. Sərlövhənin formatı belədir:

`Unit` *modulun adı***;**

Misal. **`Unit`** `Unit1;`

Kompilyasiya nəticəsində icra olunan ***.exe*** tipli fayl deyil, ***.dcu*** tipli modul faylı yaranır.

Modul sərlövhədən başqa, aşağıdakı dörd bölmədən ibarət olur:

- **`Interface`** *–interfeys bölməsi*;
- **`Implementation`** *–realizasiya bölməsi*;
- **`Initialization`** *–inisializasiya bölməsi*;
- **`Finalization`** *–deinisializasiya bölməsi*.

Beləliklə, Delphi–də modul aşağıdakı struktura malik olur (şərhləri biz əlavə etmişik):

`Unit` *modulun adı*;

// İnterfeys bölməsi
`Interface`

`Uses` *modulların siyahısı*;
`Const` *sabitlərin siyahısı*;
`Type` *tiplərin təsviri*;
`Var` *dəyişənlərin elan edilməsi*;
`Procedure` *prosedurların sərlövhəsi*;

```
    Function funksiyaların sərlövhəsi;
// interfeys bölməsinin sonu


// Realizasiya bölməsi
Implementation

    Uses modulların siyahısı;
    Const sabitlərin siyahısı;
    Type tiplərin təsviri;
    Var dəyişənlərin elan edilməsi;
    Prosedurların təsviri;
    Funksiyaların təsviri;
// realizasiya bölməsinin sonu


// İnisializasiya bölməsi
Initialization

    operatorlar;
// inisializasiya bölməsinin sonu


// Deinisializasiya bölməsi
Finalization

    operatorlar;

end.
```

Modulun birinci sətri modulun adı ilə başlayır. Qeyd etdiyimiz kimi, susmaya görə, modulun adı həmişə `unit1` olur. Modulun adını əl ilə dəyişdirmək məsləhət görülmür. Onun adını dəyişdirmək üçün *File/Save As..* əmri vasitəsilə faylı yadda saxlayaraq ona fərqli ad vermək lazımdır.

Göründüyü kimi, interfeys bölməsi `Interface` sözü ilə başlayır. Bu bölmədə bu modulu istifadə edəcək bütün modul və proqramlar tərəfindən müraciət oluna biləcək identifikatorların təsvirləri yerləşir. Bu isə o deməkdir ki, bu bölmədə təsvir edilmiş bütün dəyişənlərə digər formalardan da müraciət oluna bilər. `Uses` siyahısında bu modulun istifadə edəcəyi digər modullar sadalanır. Bundan başqa, burada tiplər, sabitlər, dəyişənlər və alt proqramlar elan olunur. `Procedure` və `Function` tipli alt proqramların, formal parametrləri göstərilməklə, yalnız sərlövhələri sadalanır. Bu alt proqramların kodları isə *realizasiya* bölməsində yerləşir. İnterfeys bölməsi `Implementation` sözü ilə yekunlaşır. Məhz bu sözlə modulun ikinci vacib bölməsi – realizasiya bölməsi başlayır.

Realizasiya bölməsi `Implementation` sözü ilə başlayır. Alt proqramların təsviri (proqramın kodları), onların interfeys bölməsində sərlövhələrinin sadalanma ardıcıllığına uyğun gəlməyə də bilər. Bu bölmədə, alt proqramların sərlövhəsində, formal parametrləri göstərməmək olar, çünki, onlar və funksiyanın nəticəsi interfeys bölməsində göstərilmişdir. Realizasiya bölməsində həmçinin yalnız bu modulda istifadə olunacaq tipləri təsvir etmək, sabit və dəyişənləri elan etmək və digər alt proqramları təsvir etmək (proqramın kodunu yazmaq) olar. Bu verilənlərə bu modulu istifadə edəcək digər proqram tərəfindən müraciət oluna bilməz, ona görə də deyirlər ki, həmin verilənlər proqramda *görünmür*.

İnisializasiya bölməsi `Initialization` sözü ilə başlayır. Bu bölmədə də modulları qoşan, proqramın başlanğıcında yerinə yetirilən operatorlar yerləşir. İnisializasiya bölməsində modullar `Uses` siyahısında sadalanan ardıcıllığa uyğun yerinə yetirilir. Əvvəlki bölmələrdən fərqli olaraq, `Initialization` bölməsi vacib bölmə deyildir və modulda bu bölməni ümumiyyətlə yazmamaq olar.

Əgər modulda inisializasiya bölməsi yazılmışdırsa, onda modulda `Finalization` sözü ilə başlayan *deinisializasiya* bölməsi istifadə oluna bilər. Bu bölmə də vacib deyildir. Burada, proqram başa çatdıqda, icra olunan operatorlar göstərilir. Deinisializasiya bölməsində modullar proqramın `Uses` siyahısının sadalanma ardıcıllığının əksinə yerinə yetirilir.

Beləliklə, biz Delphi–də proqramlaşdırmanın ən vacib elementi olan modulların strukturunu öyrəndik və sonda xatırladaq ki, layihələndirmə zamanı hər bir əlavə forması üçün bir modul yaradılır. Konkret olaraq boş forma üçün modul faylı aşağıdakı kodlardan ibarətdir (qalın şriftlərlə kodlar Delphi–də olduğu kimi yazılmış, hər bir sətir və bölmələr şərhlərlə izah edilmişdir):

```
unit Unit1;      // Modulun adı

   // interfeys bölməsinin başlanğıcı
interface

uses      // Bu sözdən sonra qoşulmuş modullar sadalanır
     Windows, Messages, SysUtils, Variants, Classes,
     Graphics, Controls, Forms,  Dialogs;

Type      // Bu sözdən sonra tiplər elan edilir
     TForm1 = class(TForm)   {Yeni TForm1 obyektinin
                               təsvirinin başlanğıcı }

       // Burada komponentlər və hadisələr təsvir edilir


       Private  // Bu sözdən sonra obyektin gizli verilənləri təsvir edilir
         { Private declarations }  // bu sətri Delphi yazır
```

```
    { Bu hissədə yalnız TForm1
      obyektinin müraciət edə biləcəyi
      dəyişən və metodlar təsvir edilir }

      Public   // Bu sözdən sonra obyektin açıq verilənləri təsvir edilir
        { Public declarations }  // bu sətri Delphi yazır

      { Bu hissədə yeni yaranmış siniflərin
      müraciət edə biləcəyi, mühafizə
      olunan dəyişən və metodlar təsvir edilir }

    end;

var       // Qlobal dəyişənlər elan edilir
  Form1: TForm1;       { TForm1 obyekti tipli Form1
                         dəyişəni təsvir edilmişdir }


// Realizasiya bölməsinin başlanğıcı
implementation

{$R *.dfm}   {.dfm tipli faylların (vizual obyektlərin
               verilənlərindən ibarət fayllar) qoşulması }

end.      // end nöqtə ilə – modulun sonu.
```

On ikinci fəsil

KOMPONENTLƏRLƏ İŞ

Artıq biz bilirik ki, əlavə interfeysi yaratmaq üçün Delphi yüzdən artıq vizual komponentlər təklif edir. Bu komponentlərin əsasları komponentlər palitrasının **`Standart`**, **`Additional`** və **`Win32`** səhifələrində yerləşir. Bunlar uyğun olaraq *standart*, *əlavə* və *32–mərtəbəli* (Windows 95–ə daxil edilmiş) komponentlər adlanır.

`Standart` səhifəsində əksəriyyəti Windows sisteminin ilkin versiyalarında istifadə olunmuş aşağıdakı interfeys komponentləri yerləşir:

`Frame` **–Freymlər;**
`MainMenu` **–Əsas menyu;**
`PopupMenu` **–Peyda olan menyu ;**
`Label` **–Yazı;**
`Edit` **–Birsətirli redaktor;**
`Memo` **–Çoxsətirli redaktor;**
`Button` **–Standart düymə;**
`CheckBox` **–Müstəqil dəyişdirici (bayraq);**
`RadioButton` **–Dəyişdirici;**
`ListBox` **–Siyahı;**
`ComboBox` **–Siyahı sahəsi;**
`ScrollBar` **–Fırlatma zolağı ;**
`GroupBox` **–Qruplar;**
`RadioGroup` **–Asılı dəyişdiricilər (və ya dəyişdiricilər) qrupu;**
`Panel` **–Panel;**
`ActionList` **–Əməliyyat obyektləri siyahısı.**

Additional səhifəsində aşağıdakı komponentlər yerləşir:

BitBtn –**Şəkilli düymə;**
SpeedButton –**Cəld müdaxilə düyməsi;**
MaskEdit –**Verilənləri şablon üzrə daxil edən birsətirli redaktor;**
StringEdit –**Sətirlər cədvəli;**
DrawGrid –**Cədvəl;**
Image –**Qrafik təsvir;**
Shape –**Həndəsi fiqurlar;**
Bevel –**Faska;**
ScrollBox –**Fırlatma oblastı;**
CheckListBox –**Dəyişdiricilər siyahısı;**
Splitter –**Ayırıcı;**
StaticText –**Statik mətn;**
ControlBar –**Alətlər paneli üçün konteyner;**
ApplicationEvents –**Əlavə hadisəsi;**
Chart –**Diaqram.**

Win32 səhifəsində Windows–un 32–mərtəbəli interfeysinə aid olan komponentlər yerləşir:

PageControl –**Bloknot;**
ImageList –**Qrafik təsvirlər siyahısı;**
RichEdit –**Tam funksiyalı mətn redaktoru;**
TabControl –**Əlfəçin;**
TrackBar –**Şkala (məkik);**
ProgressBar –**İşin gedişi indikatoru;**
UpDown –**Sayğac;**
HotKey –**Cəld klavişlər kombinasiyası redaktoru;**
Animate –**Videokliplərə baxış;**
DateTimePicker –**Tarixi daxiletmə sətri;**
MonthCalendar –**Təqvim;**
TreeView –**Obyektlər ağacı;**
ListView –**Siyahı;**
HeaderControl –**Ayırıcı;**
StatusBar –**Vəziyyətlər sətri;**
ToolBar –**Alətlər paneli;**
CoolBar –**"Sərt" alətlər paneli;**
PageScroller –**Təsvirlər fırladıcısı.**

Vizual komponentlər (idarəedici elementlər) üçün `TControl` sinfi baza sinfidir. O, komponentin vəziyyətini, ölçülərini, sərlövhəsini, rəngini və digər parametrlərini idarə etməyə imkan verir. `TControl` sinfinə vizual

komponentlər üçün ümumi olan xassə, hadisə və metodlar daxildir. Vizual komponentləri iki böyük qrupa bölmək olar: pəncərəli və pəncərəsiz idarəetmə elementləri.

Pəncərəli idarəetmə elementləri müəyyən təyinatlı xüsusiləşdirilmiş pəncərədən ibarətdir. Belə elementlərə əmrlər düymələri, mətn sahələri, fırlatma zolaqları və s. aiddir. Pəncərəli elementlər üçün `TWinControl` sinfi baza sinfidir ki, bu sinif `TControl` sinfinin birbaşa varisidir.

Pəncərəli idarəetmə elementləri daxiletmə fokusu ala bilər. Elementin fokus alması iki üsulla özünü büruzə verir: mətn kursoru vasitəsilə və düzbucaqlı ilə. Adətən mətn redaktorları fokus aldıqda onların sahələrində mətn kursoru əmələ gəlir. Susmaya görə bu kursor ekranda sayrışan şaquli xətdən ibarət olur. Digər komponentlər isə fokus aldıqda qırıq xətli qara düzbucaqlı ilə təsvir olunur. Məsələn, standart `Button` düyməsi fokus aldıqda onun sərlövhəsi bu cür düzbucaqlı ilə haşiyələnir, `ListBox` siyahısı isə fokus aldıqda cari anda seçilmiş sətir başqa rənglə fərqləndirilir.

Pəncərəsiz elementlər üçün baza sinfi `TGraphicControl` sinfidir ki, o, birbaşa `TControl` sinfindən yaranmışdır. Bu elementlər fokus ala bilmir və digər interfeys elementlərinin əcdadı ola bilmir. Pəncərəsiz elementlərə misal olaraq `SpeedButton` cəld müdaxilə düymələrini göstərmək olar ki, onların vasitəsilə əlavələr üçün alətlər paneli yaratmaq daha əlverişli olur.

Əlavələrdə idarəedici element kimi daha çox istifadə olunan vizual komponentlərə baxaq. Bu komponentlər bir sıra ümumi xassə, hadisə və metodlara malikdir.

12.1. Komponentlərin xassələri

Xassələr əlavələrin yaradıldığı və yerinə yetirildiyi zaman komponentlərin xarici görünüşü və onların özünü necə aparmasını idarə etməyə imkan verir. Yuxarıda qeyd olunduğu kimi, komponentlərə Obyektlər inspektorunun köməyi ilə və ya modulda mənimsətmə operatoru vasitəsilə qiymətlər vermək olar. Qeyd edək ki, aşağıda baxacağımız xassələr nə qədər ümumi olsalar da, elə komponentlər var ki, onlar bu və ya digər xassələrə malik olmur.

`Caption` xassəsi. `TCaption` tipli `Caption` xassəsi komponentin *sərlövhəsini* müəyyənləşdirən sətirdən ibarətdir. `TCaption` tipi `TString` sətir tipinə analojidir. Sərlövhədə hər hansı simvolu nəzərə çarpdırmaq mümkündür. Bu o deməkdir ki, cəld müdaxilə klavişləri kombinasiyasından istifadə oluna bilər, başqa sözlə, *Alt* klavişini basılı saxlayaraq nəzərə çarpdırılan simvola uyğun klavişi basdıqda, mausun düyməsini basmaqla icra olunan əməliyyat yerinə yetiriləcəkdir. Klavişlər kombinasiyasını müəyyənləşdirmək üçün uyğun simvolun qarşısında **&** işarəsi yazmaq lazımdır, məsələn, `Button1` düyməsinin `Caption` xassəsi qarşısında `&Close` yazıb, proqrama `Form1.Close;` kodunu əlavə etdikdən sonra, hazır əlavədə ***Alt+C*** klavişlərini basarkən forma bağlanacaqdır. Cəld müdaxilə klavişlərindən

istifadə olunduqda Windows klaviaturanın registrlərinin vəziyyətini (əlifbanı) də nəzərə alır.

Misal. Əlavənin sərlövhəsinin dəyişdirilməsi.

Obyektlər inspektorunun *Events* səhifəsində `OnClick` hadisəsi qarşısında mausun düyməsini iki dəfə basın. Yunit ön plana keçdikdə mətn kursoru ilə göstərilən mövqeyə aşağıdakı sətri yazın:

```
Form1.Caption:='Book';
```

Yenidən Obyektlər inspektoruna qayıdın. `OnDblClick` hadisəsini tapıb, onun qarşısında mausun düyməsini iki dəfə basın (`OnDblClick` hadisəsi mausun düyməsini iki dəfə basmaq hadisəsidir) və koda aşağıdakı sətri əlavə edin:

```
Form1.Caption:='Notebook';
```

F9 klavişini basın. Hazır əlavənin sərlövhəsinə baxaraq onun üzərində mausun düyməsini bir dəfə basın. Sərlövhədə `Book` sözü yazılacaq. İndi əlavə üzərində mausun düyməsini iki dəfə basın. Sərlövhə `Notebook` sözü ilə əvəz olunacaq.

Bu, Sizin Delphi–də yazdığınız, demək olar ki, ilk proqramdır. Diqqətlə onun modulunu nəzərdən keçirin, onun strukturunda olan dəyişiklikləri Delphi–nin təklif etdiyi "boş" modulla müqayisə edin. Görəcəksiniz ki, Sizin yaratdığınız əlavənin moduluna iki prosedur – metod əlavə edilmişdir. Bu prosedurların adlarına, onların komponent və hadisə ilə əlaqəsinə, `type` bölməsində yazılışına diqqət yetirin və `implementation` bölməsindəki strukturunu dərindən öyrənin. Bütün bunlar gələcəkdə Delphi–də proqramlaşdırmanın daha asan qavranılmasına kömək edəcəkdir.

`Align` xassəsi. `TAlign` tipli `Align` xassəsi komponentləri onların yerləşdikləri konteyner üzərində *düzləndirmək* üçündür. Konteyner kimi ən çox `Form` forması və `Panel` paneli istifadə edilir. `Align` xassəsi aşağıdakı qiymətlərdən birini ala bilər:

`alNone` –*komponentlər düzləndirilmir, yəni proqramçı forma üzərində onu harada yerləşdirmişdirsə, elə orada da qalır*;

`alTop` –*komponent konteynerin yuxarı hissəsində yerləşdirilir, hündürlüyü sabit qalmaqla, eni konteynerin eninə bərabər olur (pəncərənin bütün müştəri enini tutur)*;

`alBottom` –*komponent konteynerin aşağı hissəsində yerləşdirilir, hündürlüyü sabit qalmaqla, eni konteynerin eninə bərabər olur*;

`alLeft` –*komponent konteynerin sol hissəsində yerləşdirilir, eni sabit qalmaqla, hündürlüyü konteynerin hündürlüyünə bərabər olur (pəncərənin bütün müştəri hündürlüyünü tutur)*;

`alRight` –*komponent konteynerin sağ hissəsində yerləşdirilir, eni sabit qalmaqla, hündürlüyü konteynerin hündürlüyünə bərabər olur*;

`alClient` –*komponent bütün konteyneri (bütün müştəri oblastını) tutur*.

Bu xassənin qiymətlərinə uyğun təsirinin nəticəsini əyani görmək üçün, Komponentlər palitrasının **Standart** səhifəsindən `Panel` (konteyner) komponentini forma üzərində yerləşdirin. Obyektlər inspektorunun *Properties* səhifəsində `Align` xassəsini tapıb, qiymətlər sahəsindən uyğun qiymətləri seçin. Bu xassəyə hər yeni qiymət müəyyən etdikdə konteynerin forması da dəyişəcəkdir. `Align` xassəsinə modulda kod vasitəsilə də qiymət vermək mümkündür, məsələn:

```
Panel1.Align:= alClient;
```

Məlumdur ki, bu halda konteyner, ***F9*** klavişi basıldıqdan sonra, bütün müştəri oblastını tutacaqdır. `Panel` komponenti adətən alətlər panelini yaratmaq üçün istifadə olunduğu üçün, Windows pəncərələrində yuxarı hissədə, əsas menyudan sonra yerləşdirilir.

`Color` xassəsi. `TColor` tipli `Color` xassəsi komponentin səthinin *rəngini* dəyişmək üçün istifadə edilir. Rəngləri təyin etmək üçün sabitlərdən istifadə olunur. Bu sabitlər `$000000–$FFFFFF` intervalında dəyişən *4* baytlıq, onaltılıq ədədlərdir. Əlavələrdə daha çox istifadə edilən rənglər və onlara uyğun sabitlər cədvəl 12.1–də göstərilmişdir. `Color` xassəsinə bu sabitlər Obyektlər inspektorundan müəyyənləşdirilir. Bundan başqa, komponentin rəngini daha dəqiq müəyyən etmək üçün `Color` xassəsinin qarşısında mausun düyməsini iki dəfə basmaqla açılan *Color* (*Rəng*) standart dialoq pəncərəsindən təklif olunan konkret rənglər seçmək olar.

Cədvəl 12.1. Əsas rəng sabitləri

Sabit	Rəng	Qiyməti
`clAqua`	*Açıq mavi*	`$FFFF00`
`clBlack`	*Qara*	`$000000`
`clBlue`	*Mavi*	`$FF0000`
`clFuchsia`	*Yasəməni*	`$FF00FF`
`clGray`	*Boz*	`$808080`
`clGreen`	*Yaşıl*	`$008000`
`clLime`	*Açıq yaşıl*	`$00FF00`
`clMaroon`	*Tünd qırmızı*	`$000080`
`clNavy`	*Tünd göy*	`$800000`
`clOlive`	*Zeytunu*	`$008080`
`clPurple`	*Bənövşəyi*	`$800080`
`clRed`	*Qırmızı*	`$0000FF`
`clSilver`	*Gümüşü*	`$C0C0C0`
`clTeal`	*Göy–yaşıl*	`$808000`
`clWhite`	*Ağ*	`$FFFFFF`
`clYellow`	*Sarı*	`$00FFFF`

Misal. Komponentin rənginin dəyişdirilməsi.

Forma üzərinə bir `Panel` paneli və bir `Edit` redaktoru yerləşdirin. `Panel1` komponentini seçərək `Align` xassəsinə `alTop` qiyməti verin. Sonra

Color xassəsinə clGreen qiyməti seçin. Edit redaktorunu seçin, onun Color xassəsinə clRed qiyməti verin. Nəhayət, Form1 formasının boş sahəsində mausun düyməsini basaraq Color xassəsi üçün clAqua qiyməti təyin edin. Görəcəksiniz ki, forma açıq mavi, konteyner yaşıl və Edit redaktoru isə qırmızı rənglə rənglənmişdir. Bu əməliyyatları modulda kod vasitəsilə belə yazmaq olar:

```
Panel1.Color:=clGreen;
Edit1.Color:=clRed;
Form1.Color:=clAqua;
```

Sabitlərin bu kitabda göstərilməyən digər hissəsindən standart Windows pəncərəsinin *Свойства:Экран/Оформление* səhifəsində müəyyən olunmuş pəncərə hissələrinə müvafiq rəngləri təyin etmək üçün istifadə edilir. Həmin sabitlərlə Obyektlər inspektorunda tanış olmaq olar.

Ctl3D xassəsi. Boolean tipli bu xassə komponentin *vizual görünüşünü* müəyyənləşdirir. Ctl3D xassəsinin qiyməti *False* seçilərsə, onda komponent ikiölçülü, *True* seçildikdə isə üçölçülü olur. Bu xassə bütün komponentlərə xas olmur, məsələn, Label komponenti belə xassəyə malik deyildir.

Cursor xassəsi. TCursor tipli Cursor xassəsi *mausun göstəricisinin görünüşünü* müəyyənləşdirir. Delphi–də mausun göstəricisinin iyirmidən çox əvvəlcədən müəyyənləşdirilmiş növü və onlara uyğun sabitlər mövcuddur. Bu sabitlərin ən əsasları aşağıdakılardır:

crDefault	*–göstəricinin görünüşü susmaya görə müəyyənləşdirilir (adi ox şəklində)*;
crNone	*–göstərici görünmür*;
crArrow	*–göstərici ox şəklindədir*;
crCross	*–göstərici xaç işarəsi şəklindədir*;
crHourGlass	*–göstərici qum saatı şəklindədir*.

DragCursor xassəsi. TCursor tipli DragCursor xassəsi *komponenti hərəkət etdirdikdə mausun göstəricisinin görünüşünü* müəyyənləşdirir. Bu xassənin qiymətləri Cursor xassəsinin qiymətləri ilə eynidir.

DragMode xassəsi. TDragMode tipli DragMode xassəsi *komponentin mausla yerinin dəyişdirilməsi* (*drag–and–drop* üsulu ilə) *rejimini* müəyyən edir. Bu xassə iki qiymətdən birini ala bilər: dmManual və dmAutomatic. Susmaya görə xassəyə dmManual qiyməti verilmişdir və bu o deməkdir ki, BeginDrag metodu çağrılanacan obyektin yerini dəyişmək olmaz. DragMode xassəsinə dmAutomatic qiyməti verildikdə isə istifadəçi obyektin yerini dəyişdirə bilər.

Enabled xassəsi. Boolean tipli Enabled xassəsi komponentin *aktivliyini* müəyyən edir, başqa sözlə, maus və ya klaviaturadan daxil olan məlumata komponentin reaksiyasını müəyyənləşdirir. Susmaya görə bu xassəyə

True qiyməti verilmişdir, yəni komponent aktivdir. *False* qiyməti verildikdə isə komponent aktiv olmur.

`Font` xassəsi. `TFont` tipli `Font` xassəsi vizual komponentdə təsvir olunan mətnin *şriftini* müəyyənləşdirir. Şriftin parametrlərini idarə etmək üçün bir neçə xassələr vardır ki, onlardan ən əsasları aşağıdakılardır:

`TFontName` tipli `Name` xassəsi *şriftin adını*, məsələn, *Arial*, *Courier New*, *Times New Roman* və s. bildirir. Qeyd edək ki, şriftin adını bildirən `Name` xassəsinin komponentin adını bildirən `Name` xassəsi ilə heç bir əlaqəsi yoxdur.

Misal.
```
Label1.Font.Name:='Arial';
```

`Integer` tipli `Size` xassəsi *şriftin punktlarla* (*1 punkt=1/72 düym*) *ölçüsünü* müəyyənləşdirir.

Misal.
```
Label1.Font.Size:=14;
```

`Integer` tipli `Height` xassəsi *şriftin piksellərlə ölçüsünü* müəyyənləşdirir. Əgər bu xassənin qiyməti müsbət olarsa, onda sətirlərarası interval ölçüyə daxil olur, şriftin ölçüsü mənfi olduqda isə interval ölçüyə daxil olmur.

Misal.
```
Label1.Font.Height:= -11;
Label2.Font.Size:= -Label1.Font.Height*72;
```

Burada, baxdığımız `Height` xassəsinin də komponentin hündürlüyünü bildirən `Height` xassəsi ilə heç bir əlaqəsi yoxdur.

`TFontStyle` tipli `Style` xassəsi *şrift tərzini* müəyyənləşdirir və aşağıdakı qiymətlər kombinasiyasını ala bilər:

`fsItalic` –*kursiv*;
`fsBold` –*yarımqalın*;
`fsUnderline` –*altdan xətt çəkilmiş*;
`fsStrikeOut` –*üzərindən xətt çəkilmiş*.

`TColor` tipli `Color` xassəsi *şriftin rəngini* dəyişdirir.

Misal.
```
Label1.Font.Color:= clMaroon;
Label1.Color:= clBlue;
```

Burada, `Label` yazı komponenti üçün mətnin rəngi tünd qırmızı, fonun rəngi isə mavi müəyyənləşdirilmişdir.

Şriftin parametrlərini Obyektlər inspektorundan da müəyyənləşdirmək olar.

`Canvas` xassəsi. `TCanvas` tipli `Canvas` xassəsi *şəkilçəkmə səthi* (*xolst*, *kanva*) üzərində qrafik işləri yerinə yetirmək üçündür. Xüsusi halda, `Canvas` xassəsinə şriftin parametrləri daxildir. `Canvas` xassəsinə `Form` forması, `Label` yazısı və `Image` qrafik obrazları kimi obyektlər malik olur. Şəkilçəkmə

daha çox formanın səthində yerinə yetirilir. `Canvas` xassəsi obyektin səthini bildirdiyi üçün, o, adətən `Font` (*şrift*), `Pen` (*qələm*) və `Brush` (*fırça*) kimi alətlərlə birlikdə istifadə edilir.

Misal. Formanın səthinə mətnin çıxarılması.

Biz, indiyədək həll etdiyimiz məsələlərdə `Caption` xassəsindən istifadə etməklə formanın sərlövhəsində mətn yazırdıq. İndi isə formanın üzərində mətn yazaq. Bunun üçün forma üzərində `Button` standart düyməsi yerləşdirib, onun üzərində mausun düyməsini iki dəfə basaraq yunitdə kursorla göstərilən mövqeyə aşağıdakı kodları yazın:

```
Form1.Color:=clAqua;
// Şriftin parametrlərinin təyini
Form1.Canvas.Font.Name:= 'Courier';
Form1.Canvas.Font.Style:= [fsBold]+[fsUnderline];
Form1.Canvas.Font.Size:=28;
Form1.Canvas.Font.Color:=clRed;
// Forma səthinə mətnin çıxarılması
Form1.Canvas.TextOut(50,50,'Salam,Delphi');
```

Bu kodları daha sadə formada belə yazmaq olar:

```
Form1.Color:=clAqua;
// Şriftin parametrlərinin təyini
With Form1.Canvas.Font do begin
Name:= 'Courier';
Style:= [fsBold]+[fsUnderline];
Size:=28;
Color:=clRed;
end;
// Forma səthinə mətnin çıxarılması
Form1.Canvas.TextOut(50,50,'Salam,Delphi');
```

Bu misalda, ***F9*** klavişini basdıqdan sonra, `Button` düyməsini basdıqda formanın səthi mavi rəngli olmaqla onun üzərinə qırmızı rəngli, *28* punkt ölçülü, yarımqalın və altdan xətt çəkilmiş şriftlə `Salam,Delphi` mətni çıxarılır. Forma səthinə mətn çıxarmaq üçün

`procedure TextOut(`*x,y* **`: integer; const`** *Text* **`: string);`**

proseduru tətbiq edilmişdir. Burada, *x, y* mətnin sol yuxarı küncünün koordinatlarını, *Text* isə səthə çıxarılacaq mətn sətrini müəyyənləşdirir.

`Height` və **`Width`** **xassələri.** `Integer` tipli `Height` və `Width` xassələri uyğun olaraq komponentin piksellə *hündürlüyünü və enini* müəyyənləşdirir.

Misal.

```
GroupBox1.Height:= Form1.ClientHeight;
GroupBox1.Width:= Form1.ClientWidth;
```

Burada, komponentin hündürlüyü (`Height`) və eni (`Width`) pəncərənin müştəri oblastının hündürlüyü (`ClientHeight`) və eninə (`ClientWidth`) bərabər müəyyənləşdirilir. Bu xassələrin təsirini öyrənmək üçün forma üzərinə müxtəlif komponentlər, məsələn, `Button` standart düymələri, `Panel` panelləri, `Edit` redaktorları və s. yerləşdirdikdən sonra, onları seçərək Obyektlər inspektorundan həmin xassələrə müxtəlif qiymətlər verib, ölçüləri müşahidə etmək lazımdır.

`Left` və `Top` xassələri. `Integer` tipli `Left` və `Top` xassələri üzərində yerləşdikləri konteynerə nisbətən komponentin *sol yuxarı küncünün koordinatlarını* müəyyənləşdirir. Forma özü də komponent olduğu üçün, onun da koordinatları var və bu koordinatlar monitorun ekranının sol yuxarı küncünə görə təyin olunur. `Left`, `Top`, `Height` və `Width` xassələri birlikdə komponentin koordinatı və ölçülərini müəyyənləşdirir.

`Name` xassəsi. `String` tipli `Name` xassəsi *komponentin unikal adını* müəyyənləşdirir. Bu xassəyə qiymətlər yalnız latın əlifbası simvolları və rəqəmlərlə verilə bilər.

`Hint` xassəsi. `String` tipli `Hint` xassəsi komponentin oblastında kursoru bir az ləngitdikdə *peyda olan məlumatdır*. Bu məlumatı həmişə göstərmək üçün Obyektlər inspektorunda `Boolean` tipli `ShowHint` xassəsinə *True* qiyməti vermək lazımdır. Bu xassəyə susmaya görə *False* qiyməti verildiyi üçün məlumat təsvir edilmir.

`PopupMenu` xassəsi. `TPopupMenu` tipli `PopupMenu` xassəsi mausun göstəricisini komponentin oblastında yerləşdirdikdə onun sağ düyməsini basarkən peyda olan *kontekst menyunu* göstərir. Kontekst menyunun peyda olması üçün `AutoPopup` xassəsinə *True* qiyməti vermək lazımdır, susmaya görə isə ona *False* qiyməti verilmişdir.

`ParentColor` xassəsi. `Boolean` tipli `ParentColor` xassəsi *əcdad–obyektin rəngini* bildirir. Komponentin əcdad–obyektin rəngini qəbul etdiyini və ya öz rəngində olduğunu müəyyənləşdirir. *True* və ya *False* qiyməti alır.

`ParentCtl3D` xassəsi. `Boolean` tipli `ParentCtl3D` xassəsi *əcdad–* obyektin həcmini bildirməklə, komponentin onun *görünüşünü qəbul edib–etmədiyini* müəyyənləşdirir. *True* və ya *False* qiyməti alır.

`ParentFont` xassəsi. `Boolean` tipli `ParentFont` xassəsi əcdad–obyektin şriftlərini bildirməklə, komponentin onun *şriftlərini qəbul edib–etmədiyini* müəyyənləşdirir. *True* və ya *False* qiyməti alır.

`Text` xassəsi. `TCaption` tipli `Text` xassəsi `Caption` xassəsinə analoji olaraq komponentlə əlaqədar sətirdən ibarətdir. Lakin, `Caption` xassəsindən fərqli olaraq, `Text` xassəsinin qiyməti sərlövhəni deyil, *komponentin məzmununu* göstərir. Məsələn, `Edit`, `Memo` komponentləri üçün `Text` xassəsinin qiyməti onların daxilində redaktə olunan simvol verilənləri kimi

təsvir olunur. Biz komponentləri öyrəndikdə `Text` xassəsinin tətbiqi ilə çoxlu məsələlər həll edəcəyik.

`TabOrder` xassəsi. `TTabOrder` tipli `TabOrder` xassəsi *Tab* klavişini basdıqda konteyner daxilində komponentin *fokusalma ardıcıllığını* müəyyənləşdirir. Fokusalma komponentə idarəetmənin verilməsini müəyyən edir. Komponentlərin fokusalma ardıcıllığı onların forma üzərində yerləşmə ardıcıllığına müvafiq təyin olunur. Belə ki, forma üzərində birinci yerləşdirilmiş komponent üçün `TabOrder` xassəsinin qiyməti *0*, ikinci komponent üçün *1* və s. olur. Bir konteyner daxilində yerləşən komponentlərin fokusalma ardıcıllığının başqa konteynerdə yerləşən komponentlərin fokusalma ardıcıllığına heç bir aidiyyəti yoxdur. Fokusalma ardıcıllığını dəyişdirmək üçün `TabOrder` xassəsinə müvafiq qiymətlər vermək lazımdır. `TabOrder` xassəsi sıfra bərabər olan komponent birinci fokus alır, başqa sözlə, idarəetmə birinci ona verilir.

Misal.

```
Edit1.TabOrder:=1;
ListBox1.TabOrder:=0;
Button1.TabOrder:=2;
```

Burada, `ListBox1` komponenti birinci, `Edit1` komponenti ikinci və `Button1` komponenti isə ən axırda fokus alır və fokus yenidən `ListBox` komponentinə verilir.

Fokusalma ardıcıllığını Delphi interfeysində də dəyişmək olar. Bunun üçün *EditTabOrder* (*Tabulyasiya ardıcıllığına düzəliş*) dialoq pəncərəsini ekrana çıxarmaq lazımdır. Bu pəncərə *Edit* (*Düzəliş*) menyusundan *EditTabOrder* əmrini icra etməklə ekrana çağrılır.

`ReadOnly` xassəsi. `Boolean` tipli `ReadOnly` xassəsi informasiyanın daxil edilməsi və onlara düzəlişlərin edilməsi ilə əlaqədar olan idarəetmə elementlərinə özlərində olan *mətnə düzəlişlər edilməsinə icazə verilməsini* müəyyən edir. Əgər `ReadOnly` xassəsinə *True* qiyməti verilərsə, onda mətnə düzəlişlər edilməsinə icazə verilmir – onu yalnız oxumaq olar, *False* qiyməti verildikdə isə mətnə düzəlişlər edilməsinə icazə verilir. Mətnə düzəlişlərə qadağa yalnız istifadəçiyə aiddir, proqramçı isə `ReadOnly` xassəsinin qiymətindən asılı olmayaraq mətnə həmişə düzəlişlər edə bilər, məsələn:

```
Edit1.ReadOnly:=True;
Edit1.Text:= ' Yeni mətn ';
```

Mətnə düzəlişlər qadağan olunsa belə, redaktə komponenti həmişə fokus alır. Bu zaman mətn sahəsində əmələ gələn sayrışan mətn kursoru komponentin daxilində, simvollar üzərində yerini dəyişə bilir, lakin, mətnə düzəlişlər edilmir.

Delphi–də komponentlərin çoxlu xassələri mövcuddur. Biz burada komponentlər üçün daha ümumi və xarakterik olan əsas xassələrə baxdıq. Konkret komponentlərin xüsusiyyətlərini öyrəndikdə yeni xassələrlə tanış

olacaq, onları və burada öyrəndiyimiz xassələri tətbiq etməklə məsələlər həll edəcəyik.

12.2. Hadisələr

Vizual komponentlər onlarla müxtəlif növ hadisələr yaratmaq və onları emal etmək qabiliyyətinə malikdir. Komponentlər üçün nisbətən ümumi olan hadisələri aşağıdakı kimi qruplaşdırmaq olar:

1. *İdarəedici elementin seçilməsi*;
2. *Mausun göstəricisinin hərəkət etdirilməsi*;
3. *Klaviaturanın klavişlərinin basılması*;
4. *İdarəedici elementin daxiletmə fokusu alması və ya onu itirməsi*;
5. *Obyektlərin drag–and–drop metodu ilə hərəkət etdirilməsi*.

Əksər hadisələr `TNotifyEvent` tipinə aid olmaqla, xəbərdaredici xarakterə malikdir. Xəbərdaredici hadisə belə təsvir olunur:

```
type TNotifyEvent=
            procedure(Sender:TObject) of object;
```

Göründüyü kimi, hadisə xəbərdaredicisi yalnız hadisənin mənbəyini göstərən **`Sender`** parametrindən ibarətdir və başqa informasiyaya malik deyildir. Lakin, bəzi mürəkkəb hadisələr üçün `Sender` parametri kifayət etmir və əlavə parametrlər göstərmək lazım gəlir.

İdarəedici elementi seçdikdə `TNotifyEvent` tipli **`OnClick`** hadisəsi baş verir ki, buna *basma hadisəsi* deyilir. Bu hadisə adətən komponentin üzərində düyməni basdıqda baş verir. `OnClick` hadisəsi Delphi–də ən geniş istifadə olunan hadisədir. Biz, Delphi ilə ilkin tanışlıqda bu hadisənin tətbiqi ilə sadə misala baxmışdıq. Həmin misalda bu hadisə `Button` düyməsi basıldıqda baş verirdi. İndi isə `Label` yazı komponenti ilə əlaqədar `OnClick` hadisəsinə aid misala baxaq.

Misal. `Label` komponentinin sərlövhəsində cari vaxtın göstərilməsi.

Forma üzərində `Label` komponenti yerləşdirib, Obyektlər inspektorunda `OnClick` hadisəsindən sağ tərəfdə mausun düyməsini iki dəfə basaraq, yunitə aşağıdakı kodları yazın (aydınlıq üçün burada proseduru tam veririk):

```
procedure TForm1.Label1.Click(Sender:TObject);
begin
  Label1.Caption:= TimeToStr(Time); // Bu sətri proqramçı yazır
end;
```

Layihəni yerinə yetirdikdən sonra (***F9*** klavişini basmaqla), `Label1` komponenti üzərində mausun düyməsini basdıqda bu komponentin sərlövhəsində cari vaxt təsvir olunacaqdır. Bu misalda istifadə olunan `Time` funksiyası cari vaxtı göstərir. `Caption` xassəsi isə `TCaption` (`TString`

sinfinə analoji) tipli, yəni sətir tipli olduğu üçün, cari vaxt sətir tipinə çevrilməlidir. Belə çevirməni isə `TimeToStr` funksiyası ("*vaxtı sətrə çevir*") yerinə yetirir.

İdarəedici element klavişlər kombinasiyası ilə seçildikdə `OnClick` hadisəsi baş vermir.

Mausun istənilən düyməsini basdıqda daha iki hadisə baş verir:

`OnMouseDown` –*mausun düymələri basıldıqda*;
`OnMouseUp` –*mausun düymələri buraxıldıqda*.

Hər iki hadisə `TMouseEvent` tiplidir.

Mausun düymələrini basdıqda ardıcıl olaraq aşağıdakı hadisələr baş verir: `OnMouseDown`, `OnClick` (sol düymə üçün), `OnMouseUp`.

Bundan başqa, komponent üzərində mausun düyməsini iki dəfə basdıqda, `TNotifyEvent` tipli `OnDblClick` hadisəsi baş verir. Bu zaman hadisələr aşağıdakı ardıcıllıqla baş verir: `OnMouseDown`, `OnClick`, `OnMouseUp`, `OnDblClick`, `OnMouseDown`, `OnMouseUp`.

Hadisələri modulda kodlar vasitəsilə də yaratmaq olar. Məsələn, `Button1.Click;` operatoru `Button1` düyməsinin basılmasını yerinə yetirir.

Mausun göstəricisini komponent üzərində hərəkət etdirdikdə `TMouseMoveEvent` tipli **`OnMouseMove`** hadisəsi baş verir ki, bu hadisə belə təsvir olunur:

```
type TMouseMoveEvent=procedure(Sender:TObject;
Shift:TShiftState; x,y:integer) of object;
```

Burada, `Sender` parametri mausun göstəricisinin hansı idarəetmə elementi üzərində olmasını bildirir, tam tipli *x* və *y* parametrləri isə `Sender` elementinin koordinat sistemində göstəricinin mövqeyini müəyyənləşdirir. `Shift` parametri klaviaturanın *Alt*, *Ctrl* və *Shift* klavişlərinin və mausun düymələrinin vəziyyətini göstərir. Bu parametr aşağıdakı qiymətlər kombinasiyasını ala bilər:

`ssShift` –***Shift*** *klavişi basılmışdır*;
`ssAlt` –***Alt*** *klavişi basılmışdır*;
`ssCtrl` –***Ctrl*** *klavişi basılmışdır*;
`ssLeft` –*mausun sol düyməsi basılmışdır*;
`ssMiddle` –*mausun orta düyməsi basılmışdır*;
`ssDouble` –*mausun düyməsi iki dəfə basılmışdır*.

Göstərilən bu klavişlərdən istənilən birini basdıqda `Shift` parametrinə müvafiq qiymət verilir. Məsələn, əgər *Shift* və *Ctrl* klavişləri birgə basılmışdırsa, onda `Shift` parametrinin qiyməti `[ssShift, ssCtrl]` olur. Əgər heç bir klaviş basılmamışdırsa, onda `Shift` – `[ ]` boş qiyməti qəbul edir.

Misal. Mausun koordinatlarının göstərilməsi.

Forma üzərinə `Panel` komponenti yerləşdirin. Obyektlər inspektorunda onun `Align` xassəsinə `alTop` qiyməti verin. Komponent formanın eni boyunca yuxarı hissədə yerləşəcəkdir. *Events* səhifəsində `OnMouseMove` hadisəsi qarşısındakı sahədə mausun düyməsini iki dəfə basaraq yunitə aşağıdakı kodları yazın:

```
Procedure TForm1.FormMouseMove(Sender:TObject;
          Shift:TShiftState; x,y:integer);

begin
  Panel1.Caption:= 'soldan: '+IntToStr(x)+
                   'yuxarıdan: '+IntToStr(y);
end;
```

Həmişə olduğu kimi, Siz, yalnız `begin` və `end;` operatorları arasında yerləşən sətri yazırsınız. ***F9*** klavişini basın. Əgər proqramda heç bir səhv olmazsa, onda mausun göstəricisini forma üzərində gəzdirdikdə `Panel1` konteynerinin sərlövhəsində göstəricinin koordinatları təsvir olunacaqdır. Formanın sahəsindən mausu çıxarıb, konteynerin sahəsinə keçirdikdə, onun sərlövhəsində heç nə göstərilməyəcəkdir. Çünki, `x` və `y` parametrləri yalnız formanın koordinat sisteminə aiddir. Bundan başqa, əgər forma üzərində digər komponentlər olardısa, mausun göstəricisini həmin komponentlərin üzərində yerləşdirdikdə göstəricinin koordinatları yenə də təsvir edilməyəcəkdir.

İndi isə yazdığımız sətri izah edək. Sərlövhədə "`soldan:`" və "`yuxarıdan:`" sözləri, onların qarşılarında isə `x` və `y` dəyişənlərinin qiymətləri təsvir ediləcəkdir. Prosedurdan göründüyü kimi, `x` və `y` dəyişənləri tam tiplidir, `Caption` xassəsi isə sətir tiplidir. Ona görə də `x` və `y` dəyişənlərinin qiymətlərini sətrə çevirmək lazımdır ki, bu məqsədlə `IntToStr` ("*tamı sətrə çevir*") funksiyası tətbiq edilmişdir (bu funksiya kitabın sonundakı *Əlavədə* izah edilmişdir).

Prosedurda göstərilən `Shift` parametrini öyrənmək üçün daha bir misala baxaq.

Misal. `OnMouseMove` hadisəsində `Shift` parametri.

Yuxarıdakı misala uyğun yunitə bir neçə operator da əlavə edin ki, prosedur belə təsvir edilsin:

```
Procedure TForm1.FormMouseMove(Sender:TObject;
          Shift:TShiftState;x,y:integer);

begin
  Panel1.Caption:=
  'soldan: '+IntToStr(x)+' yuxarıdan: '+IntToStr(y);
  if ssShift in Shift then Panel1.Caption:=
  Panel1.Caption+'Shift klavişi basılmışdır ';
  if ssCtrl in Shift then Panel1.Caption:=
  Panel1.Caption+'Ctrl klavişi basılmışdır ';
  if ssAlt in Shift then Panel1.Caption:=
```

```
  Panel1.Caption+'Alt klavişi basılmışdır ';
end;
```

Mausun göstəricisi formanın üzərində olarkən, *Shift*, *Ctrl* və ya *Alt* klavişlərindən hər hansı birini basdıqda, bu hadisə sərlövhədə qeyd ediləcəkdir. Bu proqramda, gördüyünüz kimi, `if` şərt operatorundan və `in` mənsubluq əməliyyatından istifadə edilmişdir.

Misal. `OnMouseDown` hadisəsinin öyrənilməsi.

Obyektlər inspektorunun *Events* səhifəsində `OnMouseDown` hadisəsinin qarşısındakı boş sahədə mausun düyməsini iki dəfə basaraq yunitə aşağıdakı kodları yazın:

```
procedure TForm1.FormMouseDown(Sender:TObject;Button:
        TMouseButton;Shift:TShiftState; x,y:integer);
begin
  if Button=mbLeft then
     Canvas.TextOut(x,y,'maus burada olmuşdur ')
  else Form1.Refresh;
end;
```

Bu misalda, `if` operatoru mausun sol düyməsinin (`mbLeft`) basılmasını yoxlayır və əgər düymə basılmışdırsa, onda mausun göstəricisinin mövqeyinə "`maus burada olmuşdur`" mətni çıxarılır. Əks halda isə, yəni mausun digər (sağ və orta) düymələri basıldıqda forma təmizlənir (`refresh` metodu ilə). Qeyd edək ki, `Button` parametri üç qiymətdən birini ala bilər:

`mbLeft` –*sol düymə*;
`mbMiddle` –*orta düymə*;
`mbRight` –*sağ düymə*.

Klaviatura ilə işləyərkən, klavişi basdıqda **`OnKeyPress`** və **`OnKeyDown`** hadisələri, klavişi buraxdıqda isə **`OnKeyUp`** hadisəsi baş verir. Klavişi basdıqda hadisələr aşağıdakı ardıcıllıqla baş verir: `OnKeyDown`, `OnKeyPress` və `OnKeyUp`.

`TKeyPressEvent` tipli `OnKeyPress` hadisəsi, hər bir hərf–rəqəm klavişlərini basdıqda baş verir və klaviş basıldıqda tələb olunan reaksiyaya uyğun emal olunur. `TKeyPressEvent` tipi belə təsvir olunur:

```
type TKeyPressEvent=procedure(Sender:TObject;
                              var Key:char) of object;
```

Burada, simvol tipli `Key` parametri basılan klavişin **ASCII** kodunu göstərir. Əgər bu parametrə sıfır qiyməti (`#0`) verilərsə, bu o deməkdir ki, klavişin basılması ləğv edilir.

`OnKeyPress` hadisəsi idarəedici klavişlərə məhəl qoymur, *Caps Lock* və *Shift* klavişləri ilə müəyyən edilən registrlərin vəziyyətini isə nəzərə alır. Yalnız *Tab* klavişi basıldıqda `OnKeyPress` və `OnKeyUp` hadisələri baş vermir.

Misal. `OnKeyPress` hadisə emaledicisi.

Nəticələri üzərində təsvir etmək üçün, formada `Panel` paneli yerləşdirərək onun `Align` xassəsinə `alTop` qiyməti verin. Forma üzərində mausun klavişini basıb, Obyektlər inspektorunun *Events* səhifəsində, `OnKeyPress` hadisəsi qarşısında, mausun düyməsini iki dəfə basaraq yunitə aşağıdakı kodları yazın:

```
procedure TForm1.FormKeyPress(Sender:TObject;
                              var Key:char);
begin
  Panel1.Caption:= Key;
end;
```

Layihəni yerinə yetirin. İndi klaviaturada hansı simvol klavişini bassanız, panelin sərlövhəsində həmin klavişə uyğun simvol təsvir ediləcəkdir (registr nəzərə alınmaqla). Lakin, *Probel* və *Enter* klavişlərinin təsviri görünməyəcək, *Delete*, *Insert*, *Esc*, *F1–F12* və s. kimi klavişlər isə ümumiyyətlə təsvir olunmayacaqdır. Bu klavişlərin təsvirini görmək üçün növbəti misala baxaq. Bunun üçün ***Alt+F4*** klavişlərini basaraq yenidən Delphi–yə qayıdın.

Misal. Bütün simvolların təsvir edilməsi.

Əvvəlki misalın layihəsini bağlamadan formanı seçib, Obyektlər inspektorunda `OnKeyDown` hadisəsi qarşısında mausun düyməsini iki dəfə basaraq aşağıdakı kodları yunitə əlavə edin:

```
Procedure TForm1.FormKeyDown(Sender:TObject;
          var Key:word;Shift:TShiftState);
begin
  Form1.Caption:= IntToStr(Key);
end;
```

Biz bilərəkdən nəticələri formanın sərlövhəsi üzərinə çıxarırıq ki, paneldə – basılan klavişin simvolunu, formada isə ədədi qiymətlərini (kodlarını) görə biləsiniz. İndi yuxarıda baxılan misaldan fərqli olaraq, nəinki simvol klavişlərini, hətta bütün klavişlərin (*Tab* klavişindən başqa) ədədi qiymətlərini görə bilərsiniz.

Misal. `OnKeyDown` hadisəsində `Shift` parametrinin öyrənilməsi.

Sonuncu misalda gördük ki, `OnKeyDown` hadisəsində `Shift` parametri də iştirak edir. Bu parametrin funksiyasını öyrənək. Bunun üçün *File/Close All* (*Fayl/Hamısını bağlamaq*) əmri ilə köhnə layihəni bağlayın. *File/New* (*Fayl/Yeni*) əmrini icra edərək obyektlərin saxlandığı yerdən *Application* (*Əlavə*) seçin. Forma üçün Obyektlər inspektorunda `OnKeyDown` hadisəsi qarşısında mausun düyməsini iki dəfə basaraq aşağıdakı kodları yazın:

```
procedure TForm1.FormKeyDown(Sender:TObject;
         var KeyDown:word;Shift:TShiftState);
begin
  if (Shift=[ssCtrl]) and (chr(Key)='1') then
```

```
  MessageDlg('"Ctrl və 1" klavişləri basılmışdır ',
  mtConfirmation, [mbOk],0);
end;
```

Panelin üzərində ***Ctrl+1*** klavişlərini basdıqda "Ctrl və 1" klavişləri basılmışdır məlumatından ibarət **Confirm** dialoq pəncərəsi ekranda təsvir olunacaqdır. Bu pəncərə MessageDlg proseduru ilə ekrana çıxarılır.

Misal. Simvolun daxil edilməsinə qadağa.

Yeni layihədə boş formaya aid OnKeyPress hadisəsi qarşısında mausun düyməsini iki dəfə basaraq aşağıdakı kodları yazın:

```
procedure TForm1.FormKeyPress(Sender:TObject;
                              var Key:char);
begin
  if Key= '!' then
  begin
    key:= #0;
    Caption:= 'Bu simvolu daxil etmək olmaz! ';
  end
  else Caption:= Key;
end;
```

İstənilən simvolu basdıqda o, sərlövhədə təsvir olunacaq, "**!**" simvolunu basdıqda isə Key parametri sıfra bərabər edilir, sanki klaviş basılmamışdır və sərlövhədə Bu simvolu daxil etmək olmaz! mətni təsvir olunacaqdır.

Pəncərəli idarəetmə elementi fokus aldıqda TNotifyEvent tipli **OnEnter** hadisəsi baş verir. Bu hadisə element üzərində mausun düyməsini və ya *Tab* klavişini basdıqda yaranır. Element fokusu itirdikdə isə **OnExit** hadisəsi baş verir.

Misal. Komponentin fokusalma və fokus itirmə hadisə emalediciləri.

Forma üzərinə Edit komponenti yerləşdirin. Obyektlər inspektorunda OnEnter hadisəsi qarşısında mausun düyməsini iki dəfə basaraq yunitə bu kodları yazın:

```
procedure TForm1.Edit1Enter(Sender:TObject);
begin
  Form1.Caption:= (Sender as TControl).Name+
                        'fokus almışdır! ';
end;
```

F12 klavişini basmaqla formaya qayıdaraq, Edit komponentini seçin. Obyektlər inspektorunda OnExit hadisəsi qarşısında mausun düyməsini iki dəfə basaraq yunitə aşağıdakı kodları əlavə edin:

```
procedure TForm1.EditExit(Sender:TObject);
```

```
begin
  Form1.Caption:= TEdit(Sender).Name+'Aktiv deyil!';
end;
```

Hazır layihədə mausun düyməsini `Edit` daxiletmə sahəsində basdıqda formanın sərlövhəsində `fokus almışdır!` mətni, forma üzərindəki digər komponentlər üzərində basdıqda isə `Aktiv deyil!` mətni təsvir ediləcəkdir. Eyni hadisə *Tab* klavişini basdıqda da təkrar olunacaqdır. `Sender` parametrinin `Name` (*Ad*) xassəsinə müraciət iki üsulla yerinə yetirilmişdir: birinci prosedurda `Sender` parametri `as` operatoru ilə qeyri–aşkar üsulla `TControl` tipinə gətirilir, ikinci prosedurda isə bu parametr aşkar üsulla `TEdit` tipinə gətirilir.

Komponentlərin *drag–and–drop* (*"dartmaq və yerləşdirmək"*) metodu ilə yerlərini dəyişdikdə iki element istifadə edilir: mənbə və qəbuledici. *Mənbə* – hərəkət etdirilən obyektdən, *qəbuledici* isə həmin obyektin yerləşdiriləcəyi idarəetmə elementindən ibarət olur. Komponentləri hərəkət etdirdikdə ardıcıl olaraq aşağıdakı hadisələr baş verir:

`OnStartDrag` –*yerdəyişmənin başlanğıcında mənbə yaradır*;

`OnDragOver` –*obyekti üzərinə gətirdikdə qəbuledici tərəfindən çağrılır.* Bu zaman hərəkət etdirmə parametri (`State`) obyektin qəbuledicinin sahəsinə daxil olmasını, onun üzərində hərəkət etməsini və onun sahəsini tərk etməsini göstərir;

`OnDragDrop` –*obyekti üzərində yerləşdirdikdə qəbuledici tərəfindən çağrılır*;

`OnEndDrag` –*yerdəyişmə əməliyyatı başa çatdıqda qəbuledici tərəfindən yaradılır.*

Drag–and–drop texnologiyası ilə obyektlərin yerini dəyişdirdikdə adətən, iki – `OnDragDrop` və `OnDragOver` hadisələrini emal etmək kifayətdir. Yeri dəyişdirilən obyekt–mənbənin `DragMode` xassəsinə `dmAutomatic` qiyməti vermək lazımdır ki, yerdəyişmənin başlanması avtomatik yerinə yetirilsin. `OnDragOver` hadisə emaledicisinə aşağıdakı parametrlər ötürülür: **`Source`** – obyekt–mənbə, **`Sender`** – obyekt–qəbuledici, **`x, y`** – mausun göstəricisinin cari koordinatları, **`State`** – yerdəyişmənin vəziyyəti və **`Accept`** – yerdəyişmə əməliyyatının təsdiqedilmə əlaməti. Bu hadisə emaledicisində yerdəyişmə əməliyyatının mümkünlüyü təhlil olunur: əgər yerdəyişmə mümkündürsə, onda **`Accept`** əlamətinə *True* qiyməti, əks halda isə *False* qiyməti verilir.

`OnDragDrop` hadisə emaledicisində yeri dəyişdirilən obyektin qəbulu və emalı yerinə yetirilir.

Misal. `Label` yazı komponentinin forma hüdudlarında hərəkət etdirilməsi.

Forma üzərinə `Label` komponenti yerləşdirin. Bu komponentin `DragMode` xassəsinə `dmAutomatic` qiyməti verin. Formanı seçib,

`OnDragOver` hadisəsinin qarşısında mausun düyməsini iki dəfə basaraq aşağıdakı kodları yazın:

```
procedure TForm1.FormDragOver(Sender, Source:TObject;
      x,y:integer;State:TDragState; var Accept;Boolean);
begin
  if Source=Label1 then Accept:=True else Accept:=False;
end;
```

Formaya qayıdın və Obyektlər inspektorunda `OnDragDrop` hadisəsinin qarşısında mausun düyməsini iki dəfə basaraq əlavə edin:

```
procedure TForm1.FormDragDrop(Sender,Source:TObject;
                                    x,y:integer);
begin
  Label1.Left=x; Label1.Top:=y;
end;
```

F9 klavişini basdıqdan sonra, mausla `Label1` komponentinin yerini dəyişdirə bilərsiniz.

12.3. Metodlar

Vizual komponentlərlə əlaqədar çoxlu metodlar mövcuddur ki, onlar obyektləri yaratmağa, məhv etməyə, təsvir etməyə, gizlətməyə, onların konturlarını çəkməyə və s. imkan verir. Vizual komponentlər üçün daha ümumi olan bir neçə metoda baxaq.

`SetFocus` proseduru. **`SetFocus`** proseduru pəncərəli idarəetmə elementinə *daxiletmə fokusu* verir. Əgər idarəetmə elementi həmin vaxt fokus ala bilmirsə, səhv baş verir. Ona görə də fokus verməzdən əvvəl, həmin elementin fokus ala biləcəyini yoxlamaq məqsədəuyğundur. Belə yoxlama **`CanFocus`** funksiyası ilə yerinə yetirilir. Bu funksiya `Boolean` tiplidir, əgər onun qiyməti *True* olarsa, onda element fokus ala bilər, *False* olduqda isə element fokus ala bilməz. İdarəetmə elementi o vaxt fokus ala bilməz ki, o qoşulmamış vəziyyətdə olsun və onun `Enabled` xassəsinin qiyməti *False* olsun.

Misal. `Edit` mətn sahəsinə fokus verilməsi.

```
if Edit1.CanFocus then Edit1.SetFocus;
```

Burada, `Edit1` birsətirli mətn redaktoruna fokus verməzdən (`SetFocus`) əvvəl onun fokus ala biləcəyi (`CanFocus`) `if` operatoru ilə yoxlanır.

`Clear` metodu. **`Clear`** metodu komponentin mətndən ibarət məzmununu *pozmaq* üçündür.

Misal. `Edit` mətn sahəsinin təmizlənməsi.

```
Edit1.Clear;
```

`Refresh` metodu. `Refresh` metodu idarəetmə elementini *yeniləşdirmək* üçündür. Yeniləşmə elementin təsviri və onun konturlarının pozulmasından ibarətdir. Yuxarıda həll etdiyimiz misallardan birində bu metodun tətbiqinin nəticəsini müşahidə etdik.

`Perform` metodu. **`Perform`** metodu pəncərəli idarəetmə elementinə *məlumat göndərmək* üçündür. Bu funksiyanın forması belədir:

```
function Perform(Msg : Cardinal;
                 WParam, LParam : LongInt) : LongInt;
```

Burada, *Msg* – göndərilən məlumat, *WParam* və *LParam* isə məlumat haqqında əlavə informasiyalardan ibarətdir. İdarəetmə elementinə istinad edən parametri daxil etmək lazım gəlmədiyi üçün, `Perform` metodundan istifadə etmək `SendMessage` metodundan daha asandır.

12.4. Mətnlərin təsviri

Mətn (yarlık) sərlövhəyə malik olmayan idarəedici elementləri işarə etmək üçün tətbiq edilir. Mətnə ən sadə misal olaraq Windows sistemində *Пуск* (*Start*) düyməsini basdıqda açılan *Əsas menyunun* bəndlərini misal göstərmək olar. Hər bir bəndin adı elə mətndir. Bu mətnə yazı, nişan və ya yarlık deyirlər. *Mətnləri təsvir etmək* üçün Delphi **`Label`** komponentini təklif edir. Yazı, layihə yerinə yetirildikdən sonra, istifadəçi tərəfindən dəyişdirilə bilməyən *sadə mətndən* ibarətdir.

Komponentin xassələrini öyrənmək üçün **`Standard`** səhifəsindən `Label` komponentini forma üzərində yerləşdirin. Obyektlər inspektorunda onun `Caption` xassəsi qarşısında `Label1` adını pozaraq yazın: `Mən Delphi sistemini öyrənirəm`. Bu mətni daxil etmək üçün `Font` xassəsi üzərində mausun düyməsini basdıqda peyda olan üç nöqtə təsvirli düyməni iki dəfə basın. Açılan `Font` (*Şrift*) dialoq pəncərəsindən şrifti, onun ölçüsünü, rəngini, tərzini və s. seçə bilərsiniz. Mətn daxil edilən kimi o formada təsvir olunacaqdır. Ola bilər ki, mətn komponentin sahəsinə sığışmasın. Bu halda komponenti seçərək mausla onun ölçüsünü dəyişdirə bilərsiniz. Bundan başqa, komponent daxilində mətni düzləndirmək olar. Bunun üçün `TAligment` tipli `Aligment` xassəsindən istifadə edilir ki, bu xassə aşağıdakı qiymətlərdən birini ala bilər:

`taLeftJustify` – *sol tərəfə görə düzləndirmə*;
`taCenter` – *mətnin mərkəzdə yerləşdirilməsi*;
`taRightJustify` – *sağ tərəfə görə düzləndirmə*.

Əgər mətn komponentin eninə sığışmazsa, onu *sətirdən–sətrə* keçirmək olar. Bunun üçün `WordWrap` xassəsinə *True* qiyməti vermək lazımdır.

Mətn komponentin daxilinə yerləşmədikdə, mətnin uzunluğuna uyğun olaraq komponentin *ölçüsünü avtomatik dəyişmək* olar. Bunun üçün Obyektlər inspektorunda `AutoSize` xassəsinə *True* qiyməti vermək lazımdır. Lakin, unutmayın ki, `AutoSize` xassəsinə *True* qiyməti verildikdə `Aligment` və `WordWrap` xassələri komponentə təsir göstərməyəcəkdir. Baxdığımız bu xassələri komponent üzərində əyani sınaqdan keçirin.

Yazı *şəffaf* və ya *rəngli* ola bilər. Bu `Boolean` tipli `Transparent` xassəsi ilə müəyyənləşdirilir. Yazının rəngi `Color` xassəsi ilə təyin olunur. Yazını şəffaf etmək üçün `Transparent` xassəsinə *True* qiyməti vermək lazımdır. Şəffaf yazı adətən şəkil üzərində, məsələn, xəritə üzərində mətn yazdıqda lazım gəlir ki, mətn təsviri örtməsin.

Yazını digər elementin sərlövhəsi kimi istifadə etdikdə yazı komponenti ilə həmin element arasında *assosiativ əlaqə* yaratmaq lazımdır. `Label` komponenti pəncərəli element olmadığı üçün fokus ala bilmir. Lakin, onu klavişlər kombinasiyası ilə seçdikdə, fokus onunla əlaqədə olan elementə verilə bilər. Assosiativ əlaqə yaratmaq üçün `TFocusControl` tipli `FocusControl` xassəsindən istifadə olunur.

Misal. `Label` komponenti ilə `Edit` komponenti arasında əlaqə yaratmaq.

Əgər `Label` komponenti `Edit` sətir redaktorunun sərlövhəsi kimi istifadə olunarsa, onda buna uyğun kod belə yazılır:

```
Label1.FocusControl:= Edit1;
```

Bilirik ki, klavişlər kombinasiyası sərlövhədə seçilmiş simvolun qarşısında *ampersand* **(&)** işarəsi qoyulmaqla müəyyənləşdirilir. Bu zaman `Label` komponenti `ShowAccelChar` xassəsinə malik olur ki, o sərlövhədə **&** işarəsinin necə interpretasiya olunduğunu müəyyənləşdirir. Əgər bu xassə *True* qiyməti alarsa, onda **&** işarəsi klavişlər kombinasiyasını müəyyənləşdirir. Əks halda, bu xassəyə *False* qiyməti verildikdə isə klavişlər kombinasiyası işləməyəcəkdir və `FocusControl` xassəsinin qiymətindən asılı olmayaraq komponentlər arasında assosiativ əlaqə mövcud olmayacaqdır.

`Label` komponentini mausla seçdikdə isə onunla əlaqəli olan elementin fokus alması üçün `OnClick` hadisə emaledicisi yaratmaq lazımdır.

Misal. `Label` komponentinin seçilməsi.

Forma üzərinə `Label` və `Edit` komponentləri yerləşdirin. `Label` komponentini seçib, `OnClick` hadisəsi qarşısında mausun düyməsini iki dəfə basaraq bu kodları yazın:

```
procedure TForm1.Label1Click(Sender:TObject);

begin
  if Edit1.CanFocus then Edit1.SetFocus;
end;
```

Hazır layihə üzərində `Label1` komponentini seçdikdə `Edit1` mətn sahəsində mətn kursoru peyda olacaq, yəni o daxiletmə fokusu alacaqdır.

Misal. Forma üzərinə yazının çıxarılması.

Formada `Label` komponenti yerləşdirib onun sərlövhəsində, yuxarıda qeyd etdiyimiz kimi, `Mən Delphi sistemini öyrənirəm` mətnini yazın. Obyektlər inspektorunda `AutoSize` xassəsinə *True* qiyməti verin, `Font` xassəsindən isə şriftin adını, ölçüsünü, rəngini və s. parametrləri seçin. Formada `Button` standart düyməsi yerləşdirərək onun sərlövhəsində `Bağlamaq!` sözü yazıb, `OnClick` hadisəsi qarşısında mausun düyməsini iki dəfə basaraq modulda `Close;` operatoru yazın (bu prosedurla Siz artıq tanışsınız). ***F9*** klavişini basdıqdan sonra, hazır layihədə yazı komponenti üçün daxil etdiyiniz mətni görəcəksiniz və `Bağlamaq!` düyməsini basdıqda forma bağlanacaqdır. Forma üzərində olan bu mətnə nə düzəliş etmək, nə də onun yerini dəyişdirmək mümkündür. Sonralar yazıdan daha məqsədəuyğun funksiyalar üçün istifadə edəcəyik.

12.4.1. İnformasiyanın daxil və redaktə edilməsi

İnformasiyanın daxil və redaktə edilməsi formanın xüsusi sahə və oblastlarında yerinə yetirilir. İnformasiyanın daxil edilməsi və zərurət yarandıqda onlara düzəlişlərin edilməsi üçün, Delphi, **`Edit`**, **`MaskEdit`**, **`Memo`** və **`RichEdit`** komponentlərini təklif edir. `MaskEdit` komponenti mətni şablon üzrə daxil etməyə, `RichEdit` komponenti isə `Memo` komponentinin yerinə yetirdiyi funksiyalara əlavə olaraq, mətni formatlaşdırmağa imkan verən redaktorlardır. Biz `Edit` və `Memo` komponentlərini öyrənəcəyik.

12.4.1.1. Birsətirli redaktor

Birsətirli redaktor forma üzərində mətn sahələri yaratmağa imkan verdiyi üçün ona *mətn sahələri* də deyirlər. Mətn sahələri Windows pəncərələrində ən çox rast gəlinən elementlərdir (faylı yadda saxladıqda, axtardıqda onun adının daxil edilməsi, mətn redaktorlarında sözlərin axtarılması, dəyişdirilməsi sahələri və s.). Ona görə də Delphi bir neçə birsətirli komponent təklif edir ki, bunlardan ən çox istifadə olunanı **`Edit`** komponentidir.

`Edit` komponenti informasiyanı klaviaturadan daxil etməyə və müxtəlif simvolları redaktə etməyə imkan verir. Bu zaman idarəetmə klavişləri ilə mətn kursorunu sətir üzərində hərəkət etdirmək, *Delete* və *Backspace* klavişləri ilə simvolları pozmaq və mətnin hissələrini seçmək və s. kimi əməliyyatları yerinə yetirmək olar. Yeri gəlmişkən qeyd edək ki, `Edit` komponenti *Enter* və *Esc* klavişlərinə məhəl qoymur.

`Edit` komponenti `Caption` xassəsinə malik deyildir. Onun əsas xassəsi `Text` xassəsidir ki, `Caption` xassəsindən fərqli olaraq, bu xassə sərlövhəni deyil, *komponentin məzmununu* (sətirdə olan mətni) bildirir.

Mətn sahələri adətən bir sətrin daxil edilməsi üçün nəzərdə tutulduğundan onların hündürlüyü çox da böyük olmur. Lakin, şriftin hündürlüyü və mətnin uzunluğuna mütənasib olaraq *komponentin ölçüsünün avtomatik* olaraq dəyişməsi üçün `AutoSize` xassəsindən istifadə etmək lazımdır (`Label` komponentində olduğu kimi).

Redaktə sətrində *simvollar registrini dəyişdirmək* üçün `TEditCharCase` tipli `CharCase` xassəsi mövcuddur ki, bu da aşağıdakı üç qiymətdən birini ala bilər:

`ecLowerCase` –*mətnin simvolları aşağı registr simvollarına çevrilir*;
`ecNormal` –*simvollar registri dəyişmir*;
`ecUpperCase` –*mətnin simvolları yuxarı registr simvollarına çevrilir*.

Forma üzərində `Edit` komponenti yerləşdirib, müxtəlif registrlərdə daxil edilmiş simvollar yığımından ibarət mətn yazaraq bu xassələrin təsirini özünüz yoxlayın. Bundan başqa, simvollar registrini dəyişdirmək üçün **`AnsiLowerCase`** və **`AnsiUpperCase`** funksiyaları da istifadə oluna bilər. Bu funksiyalar *Əlavədə* izah edilmişdir.

Misal. Mətn sahəsi daxilində simvollar registrinin dəyişdirilməsi.

Forma üzərinə iki `Edit` komponenti və `Button` düyməsi yerləşdirin. `Button` düyməsini seçərək `OnClick` hadisəsini aktivləşdirib aşağıdakı kodları yazın:

```
procedure TForm1.Button1Click(Sender:TObject);
begin
  Edit1.Text:= AnsiLowerCase(Edit1.Text);
  Edit2.Text:= AnsiUpperCase(Edit2.Text);
end;
```

Hazır layihədə `Edit1`, `Edit2` mətn sahələrinə müxtəlif registrli mətnlər daxil edin. `Button1` düyməsini basdıqda `Edit1` sahəsinə daxil edilən mətn kiçik hərfli mətnə, `Edit2` sahəsinə daxil edilən mətn isə baş hərflərdən ibarət mətnə çevriləcəkdir.

Birsətirli redaktorda *şifrə* də daxil etmək olar. Bunun üçün `Char` tipli `PasswordChar` xassəsindən istifadə etmək lazımdır. Obyektlər inspektorunda susmaya görə bu xassəyə `#0` qiyməti verilmişdir, yəni şifrə istifadə edilmir. Şifrəni kod vasitəsilə də daxil etmək olar:

```
Edit1.PasswordChar:= '*';
Edit1.Text:= 'Delphi';
```

Bu halda mətn sahəsində `******` sətri peyda olacaq, əslində isə `Text` xassəsinin qiyməti `Delphi` mətnindən ibarətdir.

Misal. Formanın sərlövhəsinin dəyişdirilməsi.

Forma üzərinə `Label`, `Edit` və `Button` komponentləri endirərək onları şəkil 12.1 – də olduğu kimi yerləşdirin. Obyektlər inspektorunda `Label1` komponentinin sərlövhəsini – `Yeni sərlövhəni daxil et:`, `Button1` komponentinin sərlövhəsini isə `Sərlövhəni dəyiş` adlandırın, `Edit1` komponentinin `Text` xassəsindəki `Edit1` mətnini pozub `Microsoft Excel 2002` yazın. `Button1` düyməsinin `OnClick` hadisəsi üçün aşağıdakı kodları yazın:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Form1.Caption:= Edit1.Text;
end;
```

Şəkil 12.1. Formanın sərlövhəsinin dəyişdirilməsi

Proqramı işə salın. Düyməni basdıqda formanın sərlövhəsi `Microsoft Excel 2002` olacaqdır. `Edit` mətn sahəsində `Microsoft Word 2002` yazıb düyməni yenidən basın. Sərlövhə uyğun olaraq bu mətnlə əvəz olunacaqdır. Beləliklə, Siz, hər dəfə `Edit` sahəsində yeni mətn yazıb düyməni basdıqda formanın sərlövhəsi dəyişəcəkdir.

Mətn sahəsindən daxil edilən mətnlərə nəzarət etmək də mümkündür. Bunun üçün klavişlərin basılması hadisə emaledicilərindən, məsələn, **`OnKeyPress`** emaledicisindən istifadə etmək olar.

Misal. Mətn sahəsindən daxil edilən informasiyaya nəzarət.

Forma üzərində yalnız `Edit` komponenti yerləşdirərək `OnKeyPress` hadisəsi qarşısında mausun düyməsini iki dəfə basaraq aşağıdakı kodları yazın:

```
procedure TForm1.Edit1KeyPress(Sender:TObject;
                               var Key:char);

begin
  if not(key in ['0'..'9']) then
  begin
    Form1.Caption:= 'Siz simvol klavişini basmışsınız ';
    Key:= #0;
  end
    else Form1.Caption:= Key;
end;
```

Bu modulda `if` operatoru yerləşən sətri belə də yazmaq olar:

```
if (Key<'0') or (Key>'9') then
```

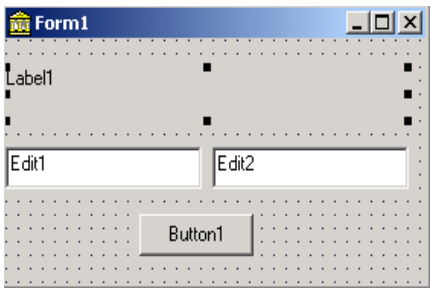
Proqramı işə buraxın. Bu proqramın yerinə yetirdiyi funksiya klaviaturadan yalnız rəqəmlərin daxil edilməsinə icazə verməkdir. Burada `if` operatoru basılan klavişi (`Key`) yoxlayır, əgər o, baxılan çoxluğa (`0,1,…,9` rəqəmləri) daxil deyilsə (`if not(key in ['0'..'9'])`), formanın sərlövhəsində istifadəçiyə xəbərdarlıq edilir və `Key` parametrinə sıfır qiyməti verir (sanki heç bir klaviş basılmamışdır). Rəqəm klavişləri basıldıqda isə ədəd sərlövhədə təsvir olunur.

`Edit` komponenti bir sətirdən ibarət olduğu üçün, mətndə sətrin sonu işarəsi (`#13` kodu) olmur və ona görə də bu komponent *Enter* klavişinə məhəl qoymur. `Edit` komponentinin *Enter* klavişinə reaksiya verməsi üçün kodları proqramçı özü yazmalıdır. Bu məqsədlə nümunə üçün aşağıdakı metoddan istifadə oluna bilər:

```
procedure TForm1.Edit1KeyPress(Sender:TObject;
                               var Key:Char);

begin
  if Key= #13 then begin
  Key:= #0;
  Button1.SetFocus;
end;
```

Burada, *Enter* klavişi basıldıqdan sonra, idarəetmə `Button1` düyməsinə verilir (fokus alır).

`Edit` komponentinin tətbiqinə aid daha bir neçə misala baxaq.



Şəkil 12.2. Vurma əməliyyatı yerinə yetirən kalkulyator

Misal. Vurma əməliyyatı yerinə yetirən kalkulyatorun hazırlanması.

Forma üzərində komponentləri şəkil 12.2 – də göstərildiyi kimi yerləşdirin. `Label1` komponentinin sərlövhəsini pozun, `Button1` düyməsinin sərlövhəsini `Vurma` adlandırın, `Edit` komponentlərinin isə `Text` xassələrini pozun. `Button1` düyməsi üçün `OnClick` hadisə emaledicisi yaradın. Bu məsələnin proqramının tam mətni aşağıdakı kimi olacaqdır:

```
unit Unit1;

interface

uses
   Windows,Messages,SysUtils,Classes,Graphics,
   Controls,Forms,Dialogs,StdCtrls,ExtCtrls;

type
 TForm1 = class(TForm)
   Edit1: TEdit;
   Edit2: TEdit;
   Button1: TButton;
```

```
    Label1: TLabel;
    procedure Button1Click(Sender: TObject);

private
  { Private declarations }

public
  { Public declarations }

end;

var
 Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
Var z: LongInt;  z1,z2,s: String;
begin
 z1:= Edit1.Text;
 z2:= Edit2.Text;
 z:= StrToInt(z1)*StrToInt(z2);
 s:= IntToStr(z);

 With label1.Font do
  begin
    Name:= 'Courier';
    Size:= 16;
    Color:= clRed;
    Style:= [fsBold];
  end;

 Label1.Caption:=S;
end;
end.
```

Burada, `z1`, `z2` və `z` dəyişənləri tam tipli (`LongInt`), `s` isə sətir tipli (`String`) elan edilir. Ona görə də bu proqram yalnız tam ədədlərin hasilini hesablayacaqdır və onluq kəsr ədədlər daxil etmək olmaz. `Edit1` və `Edit2` mətn sahələrindən daxil edilən ədədlər sətir tipdən tam ədədlərə çevrilərək (`StrToInt`) vurulur və hasil – `z` yenidən (bu dəfə tərsinə) tam ədəddən sətir tipə çevrilir (`IntToStr`, *Əlavəyə* bax). `Button` düyməsini basdıqda nəticə qırmızı rəngli, *16* punktluq, yarımqalın, `Courier` şrifti ilə `Label` yazısı üzərində təsvir edilir.

İndi isə həmin məsələni vuruqları bir mətn sahəsindən daxil etməklə həll edək.

Misal. Vuruqları bir mətn sahəsindən daxil edən kalkulyator.
Məsələnin xüsusiyyəti ondan ibarətdir ki, biz yuxarıdakı proqramda

```
z1:= Edit1.Text;
z2:= Edit1.Text;
```

yazaraq eyni bir mətn sahəsindən növbə ilə müxtəlif ədədlər daxil etdikdə, `z1` və `z2` dəyişənləri həmişə bir–birinə bərabər olacaqdır. Artıq bildiyiniz kimi, ənənəvi proqramlaşdırmada, məsələn, Turbo Pascal dilində

```
read(z1);
read(z2);
```

yazıldıqda dəyişənlərə müxtəlif qiymətlər daxil edilir. Burada isə belə deyildir. Odur ki, eyni bir sahədən iki müxtəlif qiymət daxil etmək üçün redaktorun *Enter* klavişinə reaksiyavermə prosedurundan və dəyişənin (`z1`) qlobal tipli elan edilməsindən istifadə edəcəyik. Beləliklə, həll edəcəyimiz məsələnin yuniti belə olacaqdır:

```
unit Unit1;

interface

uses
   Windows,Messages,SysUtils,Classes,Graphics,
   Controls,Forms,Dialogs,StdCtrls,ExtCtrls;

type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Button1: TButton;
    Label1: TLabel;
    procedure Button1Click(Sender: TObject);
    procedure Edit1KeyPress(Sender: TObject;
                            var Key: Char);
  private
    { Private declarations }

  public
    { Public declarations }
  end;

var

   Form1: TForm1;
   z1:LongInt;    // Qlobal dəyişən

implementation

{$R *.DFM}
```

```
// Klaviaturanın Enter klavişinə reaksiyası
procedure TForm1.Edit1KeyPress(Sender:TObject;
                                  var Key: Char);
begin
  if Key= #13 then
  begin
    Key:= #0;
    Edit1.SetFocus;
    z1:= StrToInt(Edit1.Text);
    Edit1.Clear;
  end;
end;

procedure TForm1.Button1Click(Sender: TObject);
Var
   z2,z:LongInt;s:String;
begin
  z2:= StrToInt(Edit1.Text);
  z:= z1*z2;
  s:= IntToStr(z);
  With label1.Font do
  begin
    Name:= 'Courier';
    Size:= 16;
    Color:= clRed;
    Style:= [fsBold];
  end;
  Label1.Caption:=S;
end;

end.
```

Burada, `OnKeyPress` hadisə emaledicisində, `Edit1` komponentinə daxiletmə fokusu verilir, birinci vuruq daxil edilir, *Enter* klavişi basıldıqdan sonra mətn sahəsi təmizlənir (ikinci vuruğun daxil edilməsi üçün hazırlanır).

Digər komponentləri öyrəndikdə biz nisbətən daha mükəmməl kalkulyator hazırlayacağıq.

12.4.1.2. Çoxsətirli redaktor

Çoxsətirli redaktorla işləmək üçün **`Memo`** komponentindən istifadə olunur. Bu redaktor `Edit` birsətirli redaktorunun bütün imkanlarına malik olmaqla, ondan fərqli olaraq bir yox, çoxlu sətirlərdən ibarət olur.

Çoxsətirli redaktorun *bütün məzmununa* müdaxilə etmək üçün `String` tipli `Text` xassəsindən istifadə olunur. Bu halda `Memo` komponentinin bütün məzmunu bir sətir kimi təsvir olunur. Sətrin sonu simvolu iki `#13#10` kodlarından ibarət olur ki, bu da *Enter* klavişinin basılmasını təsvir edir.

Memo komponentinin məzmunu olan mətnin *ayrı–ayrı sətirlərinə* müraciət etmək üçün isə onun TStrings tipli Lines xassəsindən istifadə olunur. Delphi–də TStrings sinfi məxsusi olaraq sətirlər üzərində əməliyyatlar yerinə yetirmək üçün yaradılmışdır. Bu sinifdə sətirlərin nömrələnməsi sıfırdan başlayır.

Misal.

```
Memo1.Lines[5]:= 'İnformatika';
Memo2.Lines.Clear;
Memo3.Lines.Add('Riyaziyyat');
```

Burada, Memo1 redaktorunun altıncı sətrinə yeni İnformatika qiyməti mənimsədilir, Memo2 redaktorunun məzmunu pozulur və Memo3 redaktorunun mətninin sonuna Riyaziyyat sözündən ibarət yeni sətir əlavə olunur. Sətrin əlavə edilməsi

Add(Const *s*:**String):Integer;**

funksiyası ilə yerinə yetirilmişdir.

Memo komponentinin məzmununu hər hansı mətn faylından da *yükləmək* və mətn faylında *yadda saxlamaq* mümkündür. Bunun üçün TStrings sinfinin

LoadFromFile(const *FileName*:**String);**

və

SaveToFile(const *FileName*:**String);**

metodlarından istifadə olunmalıdır. Burada, *FileName* parametri oxunacaq və ya yadda saxlanılacaq faylın adını bildirir.

Misal.

```
Memo1.Lines.LoadFromFile('c:\Рабочий стол\Мои
                         документы\fac.txt');
Memo2.Lines.SaveToFile('c:\ Рабочий стол\Мои
                         документы \kafedra.txt');
```

Redaktorda olan mətnin ixtiyari sətirlərinə baxmaq üçün orada *fırlatma zolaqları* yerləşdirmək lazımdır. Bu, TScrollStyle tipli ScrollBars xassəsi ilə yerinə yetirilir və o aşağıdakı qiymətlərdən birini ala bilər:

ssNone – *fırlatma zolaqları yoxdur (susmaya görə)*;
ssHorizontal – *redaktorun aşağı hissəsində üfqi fırlatma zolağı var*;
ssVertical – *redaktorun sağ tərəfində şaquli fırlatma zolağı var*;
ssBoth – *hər iki fırlatma zolağı var*.

Redaktorun daxilində yerləşən mətni *düzləndirmək* üçün TAligment tipli Aligment xassəsindən istifadə edilir. Bu xassə aşağıdakı qiymətlərdən birini ala bilər:

taLeftJustify – *sol tərəfə görə düzləndirmə (susmaya görə)*;
taCenter – *mərkəzə görə düzləndirmə*;
taRightJustify – *sağ tərəfə görə düzləndirmə*.

`Edit` birsətirli redaktorundan fərqli olaraq `Memo` redaktoru *Enter* klavişinə məhəl qoyur. Bu zaman *yeni sətrin daxil edilməsi* üçün `WantReturns` xassəsinə *True* qiyməti verilməlidir. Əks halda, `WantReturns` xassəsinə *False* qiyməti verildikdə, redaktor *Enter* klavişinə deyil, *Ctrl+Enter* klavişlərinin basılmasına reaksiya verir.

Redaktə komponentləri bir–birinə çox oxşar olduğu üçün, onlara bir sıra ümumi xassə, hadisə və metodlar xasdır.

Redaktorun məzmununda istənilən dəyişiklik edildikdə `TNotifyEvent` tipli **`OnChange`** hadisəsi baş verir ki, bu hadisə daxil edilən informasiyanın yoxlanılması və ona nəzarət edilməsi üçün istifadə oluna bilər. Bundan başqa, redaktorun *mətninə düzəlişlər edildikdə* **`Modified`** xassəsi *True* qiyməti alır. Bu xassə adətən faylın mətninə düzəlişlər edildikdən sonra, onların yadda saxlanılması haqqında istifadəçiyə xəbərdarlıq göndərilməsi məqsədilə istifadə olunur, məsələn:

```
if Memo1.Modified then
   ShowMessage('Fayl yadda saxlanılmamışdır!');
```

Redaktora daxil ediləcək simvolların *maksimal sayını* göstərmək üçün `Integer` tipli `MaxLength` xassəsindən istifadə olunur. Daxil ediləcək simvolların sayına məhdudiyyət qoyulmadıqda bu xassəyə sıfır qiyməti (susmaya görə) verilir.

Mətnin seçilmiş fraqmentləri ilə işləmək üçün redaktor `AutoSelect`, `SelStart`, `SelLength` və `SelText` xassələrinə malikdir.

`Boolean` tipli `AutoSelect` xassəsi redaktə elementi daxiletmə fokusu aldıqda mətnin *avtomatik olaraq seçilməsini* müəyyənləşdirir (susmaya görə *True* qiyməti alır).

`String` tipli `SelText` xassəsi mətnin *seçilmiş fraqmentini* müəyyən edir, mətn seçilmədikdə onun qiyməti boş sətir olur.

`Integer` tipli `SelStart` və `SelLength` xassələri, uyğun olaraq, sətirdə seçilmiş *fraqmentin başlanğıc mövqeyini* (birinci simvolun nömrəsi sıfra bərabər olur) və *uzunluğunu* göstərir. `SelStart` və `SelLength` xassələri qarşılıqlı əlaqədə olduğuna görə, mətn fraqmentini proqram yolu ilə seçdikdə, əvvəlcə `SelStart` xassəsinə, sonra isə `SelLength` xassəsi ilə mətnin uzunluğuna qiymət vermək lazımdır.

Misal.

```
Memo1.SelStart:= 4;
Memo1.SelLength:= 15;
Memo1.SelText:= 'Delphi';
```

Burada, *5*–ci simvoldan başlayaraq *15* simvol `Delphi` mətni ilə əvəz olunur.

Seçilmiş mətn fraqmentləri üzərində əməliyyatlar aparmaq üçün bu xassələrdən başqa, `SelectAll`, `CopyToClipBoard` və `CutToClipBoard` metodları da mövcuddur.

SelectAll metodu redaktə elementində olan *bütün mətni seçir*;

CopyToClipBoard və **CutToClipBoard** metodları, uyğun olaraq, seçilmiş mətni *mübadilə buferinə köçürür* və *kəsib mübadilə buferində yerləşdirir*.

Misal.

```
Edit1.SelectAll;
Edit1.CutToClipboard;
```

`Edit` redaktorundan mətn pozularaq mübadilə buferinə yerləşdirilir.

Mübadilə buferi ilə işləmək üçün **PasteFromClipBoard** metodu da vardır. Bu metodla mübadilə buferində yerləşən mətn redaktorda *kursorla göstərilən yerə yerləşdirilir* və ya redaktorda seçilmiş mətn fraqmenti buferdəki mətnlə *əvəz edilir*.

12.5. Siyahılar

Siyahı mətn sətirlərindən ibarət qarşılıqlı əlaqəli, nizamlanmış elementlər yığımıdır. Windows sistemində siyahılardan geniş istifadə olunur (*Font* dialoq pəncərəsində şriftin adı, tərzi, ölçüsü, rəngi və s.). Bu sistem üçün aşağıdakı siyahılar xarakterikdir:

Açılan siyahı pəncərədə *bükülmüş sətirdən* ibarət olur. Bu sətirdə yerləşən düymə üzərində mausun düyməsini basdıqda siyahı açılır və bu siyahıdan istənilən bəndi seçmək olar. Siyahı büküldükdə seçilmiş bənd bir sətirdə təsvir olunur.

Siyahıdan ibarət açılan sahə – açılan siyahıya oxşayır, lakin, ondan fərqli olaraq, siyahıya klaviaturadan yeni qiymət (bənd) əlavə etmək olar. Bu siyahıya *kombinasiyalı siyahı* da deyirlər. Bu iki idarəetmə elementi Delphi–nin təklif etdiyi `ComboBox` komponenti ilə yaradılır.

Sadə siyahı – ekranda dərhal görünən *bir neçə sətirdən* ibarət olur. Bu element `ListBox` komponenti ilə yaradılır.

12.5.1. Sadə siyahı

Sadə siyahılarda mətnlərdən ibarət sətirlər düzbucaqlı sahədə yerləşir. Sadə siyahıları yaratmaq üçün **Standart** səhifəsindəki **ListBox** komponentindən istifadə olunur.

Əgər sətirlərin sayı görünmə sahəsində yerləşə biləcəyindən çoxdursa, onda siyahıda fırlatma zolağı əmələ gəlir. *Fırlatma zolaqları və sütunların sayı* `Integer` tipli `Columns` xassəsinin qiymətindən asılıdır. Əgər onun qiyməti *0* olarsa, onda sətirlər bir sütunda yerləşəcək və zərurət yaranarsa, şaquli fırlatma zolağı avtomatik əmələ gələcək və ya itəcəkdir. Əgər `Columns` xassəsinin qiyməti *1*–dən böyük və ya *1*–ə bərabər olarsa, onda hökmən üfqi fırlatma zolağı olacaq və sütunların sayı xassənin qiyməti qədər olacaqdır. Siyahıda hər

iki fırlatma zolağının olması üçün `Columns` xassəsinə *0* qiyməti vermək lazımdır. Bu zaman şaquli fırlatma zolağı, zərurət yaranarsa, peyda olacaqdır. Üfqi fırlatma zolağını yaratmaq üçün isə `SendMessage` metodu ilə siyahıya `LB_SetHorizontalExtent` məluməti göndərmək lazımdır.

Misal. İki fırlatma zolağı olan siyahı.

```
procedure TForm1.FormCreate(Sender:TObject);
begin
  ListBox1.Columns:=0;
  SendMessage(ListBox1.Handle,
              LB_SetHorizontalExtent,1000,0);
end;
```

Burada, `Columns` xassəsinə *0* qiyməti verməklə şaquli fırlatma zolağı yaradılır. Üfqi fırlatma zolağı isə `SendMessage` metodu ilə yaradılır. Bu metodda birinci parametr `ListBox1` komponenti ilə əlaqə yaradır (`Handle`), ikinci parametr üfqi fırlatma zolağını yaradır, üçüncü parametrlə üfqi fırlatma zolağının həmişə təsvir edilməsi müəyyənləşdirilir: əgər bu parametr siyahının ölçüsündən böyük olarsa, üfqi fırlatma zolağı həmişə görünəcəkdir. Dördüncü parametr burada lazım olmadığı üçün sıfra bərabər edilmişdir.

Sadə siyahının *üslubu* `TListBoxStyle` tipli `Style` xassəsi ilə müəyyənləşdirilir: Bu xassə aşağıdakı qiymətləri ala bilər:

`lbStandart` –*standart üslub (susmaya görə)*;
`lbOwnerDrawFixed` –`ItemHeight` *xassəsi ilə müəyyən olunmuş eyni hündürlüklü elementlərdən ibarət siyahı*;
`lbOwnerDrawVariable` –*müxtəlif hündürlüklü elementlərdən ibarət siyahı*.

Siyahı *haşiyə daxilində* də ola bilər. Bu `TBorderStyle` tipli `BorderStyle` xassəsi ilə təyin olunur və bu xassə aşağıdakı qiymətləri ala bilər:

`bsNone` –*haşiyə yoxdur*;
`bsSingle` –*haşiyə var (susmaya görə).*

12.5.2. Kombinasiyalı siyahı

Kombinasiyalı siyahı redaktə sahəsini və siyahını birləşdirir. İstifadəçi qiyməti siyahıdan seçə və ya redaktə sahəsindən birbaşa daxil edə bilər. Kombinasiyalı siyahıları yaratmaq üçün Delphi **`ComboBox`** komponentini təqdim edir. Bu komponentlə yaradılan siyahı bükülmüş (bir sətirdən ibarət) və ya açıq ola bilər. Sadə siyahıdan fərqli olaraq, kombinasiyalı siyahıda üfqi fırlatma zolağı olmur. Kombinasiyalı siyahının *xarici görünüşünü* və onun özünü necə aparmasını `TComboBoxStyle` tipli `Style` xassəsi müəyyənləşdirir. Bu xassə aşağıdakı qiymətləri ala bilər:

csDropDown –*redaktə sahəsi olan açılan siyahı (susmaya görə)*. İstifadəçi qiyməti siyahıdan seçə bilər, bu zaman o redaktə sahəsində təsvir edilir və ya o, informasiyanı birbaşa daxiletmə sahəsindən daxil edə bilər;

csSimple –*daimi açılan siyahılı redaktə sahəsi*;

csDropDownList –*siyahıdan element seçməyə imkan verən açılan siyahı*;

csOwnerDrawFixed –ItemHeight *xassəsi ilə müəyyən olunmuş eyni hündürlüklü elementlərdən ibarət siyahı*;

csOwnerDrawVariable–*müxtəlif hündürlüklü elementlərdən ibarət siyahı*.

Style xassəsinə sonuncu iki qiyməti verdikdə proqramçı Delphi–nin qrafikçəkmə imkanlarından istifadə edərək siyahının elementlərinin konturlarını özü çəkməlidir.

Kombinasiyalı siyahının aşağıdakı xassələri də vardır:

Integer tipli DropDownCount xassəsi açılan siyahıda eyni zamanda təsvir olunan *sətirlərin sayını* müəyyənləşdirir. Bu xassənin qiyməti Items xassəsinin Count alt xassəsinin qiyməti ilə əlaqədardır. Belə ki, DropDownCount xassəsinin qiyməti Count xassəsinin qiymətindən böyük olarsa, onda açılan siyahıda avtomatik olaraq şaquli fırlatma zolağı əmələ gəlir. DropDownCount xassəsinin qiyməti susmaya görə *8*–ə bərabərdir.

Boolean tipli DroppedDown xassəsi siyahının *açıq* və ya *bükülü* olduğunu müəyyən edir. Əgər bu xassənin qiyməti *True* olarsa, siyahı açılmış vəziyyətdə olur. Əgər Style xassəsinin qiyməti csSimple olarsa, onda bu xassə heç nəyə təsir etmir. DroppedDown xassəsinə proqram yolu ilə də qiymət vermək olar:

```
ComboBox3.DroppedDown:=False;
```

Siyahıda elementləri *əlifba sırası* ilə düzmək üçün Sorted xassəsinə *True* qiyməti vermək lazımdır. Bu xassə dinamik deyil, statik təsirə malikdir, yəni əlifba sırası ilə düzülmüş siyahıya yeni sətir əlavə edildikdə, o ya daxil edildiyi mövqedə qalır, ya da siyahının sonuna əlavə edilir. Bu zaman siyahını əlifba sırası ilə düzmək üçün Sorted xassəsinə əvvəlcə *False*, sonra isə *True* qiyməti vermək lazımdır:

```
ListBox1.Sorted:=False;
ListBox1.Sorted:=True;
```

Adi halda siyahıda yalnız bir sətri seçmək olar. *Bir neçə sətri seçmək üçün* MultiSelect xassəsinə *True* qiyməti vermək lazımdır. Bunu həm Obyektlər inspektorundan, həm də kod vasitəsilə icra etmək olar, məsələn:

```
ListBox1.MultiSelect:=True;
```

MultiSelect xassəsinə *True* qiyməti verildikdə bir neçə sətrin seçilməsi üsulunu ExtendedSelect xassəsi müəyyənləşdirir. Bu xassəyə *True* qiyməti verildikdə (məsələn, ListBox1.ExtendedSelect:=True;) siyahıda

elementləri kursorla idarəetmə klavişləri (sola, sağa, aşağı və yuxarı), *Shift* və *Ctrl* klavişləri ilə seçmək olar. Lakin, unutmayın ki, bu iki xassə yalnız sadə siyahıya aiddir. `ComboBox` siyahısında eyni zamanda yalnız bir elementi seçmək mümkün olduğu üçün, onun `MultiSelect` və `ExtendedSelect` xassələri yoxdur.

12.5.3. Siyahıların `Items` xassəsi

Sadə və kombinasiyalı siyahıların bir sıra oxşar cəhətləri olduğundan onlar çoxlu ümumi xassə, metod və hadisələrə malikdir. Siyahıların ən əsas xassəsi **`Items`** xassəsidir ki, bu xassənin də öz növbəsində çoxlu xassə və metodları vardır. `TStrings` tipli `Items` xassəsi elementləri sətirlərdən ibarət olan massiv olmaqla, siyahıda elementlərin miqdarını və onların məzmununu müəyyən edir. `TStrings` sinfi mücərrəd sinif olmaqla, Delphi–də məxsusi olaraq sətirlərlə işləmək üçün yaradılmışdır. `Items` xassəsinin nümunəsində `TStrings` sinfinin əsas xassə və metodlarına baxaq.

Bu sinfin varisləri kimi `ListBox.Items`, `Memo.Lines`, `RichEdit.Lines`, `ComboBox.Items`, `TStringsList` və s. göstərmək olar. Bütün bu xassələr mahiyyətcə eyni, eynitipli və qarşılıqlı əvəz olunandır. Məsələn, bir siyahını birbaşa başqa siyahıya mənimsətmək olar:

```
ListBox1.Items:= Memo.Lines;
```

Həm `Items`, həm də `Lines` xassələri `TStrings` sinfinin varisləri olmaqla eyni tiplidir. Lakin, unutmayaq ki, belə mənimsətmə zamanı `ListBox` siyahısında olan köhnə elementlər pozulacaqdır.

Misal. `TStrings` siyahısının yaradılması.

Forma üzərinə `ListBox` və `Button` düymələri yerləşdirin. Düymənin sərlövhəsini `Add` ("əlavə etmək") adlandırın. `OnClick` hadisəsini aktivləşdirin və yunitə bu kodları yazın:

```
procedure TForm1.Button1Click(Sender: TObject);
Var MyList:TStrings; // MyList adı sərbəst seçilmişdir
begin
  MyList:= TStringList.Create;
  try
    With MyList do
       begin
         Add(' Riyaziyyat ');
         Add(' İnformatika ');
         Add(' Fizika ');
       end;
  ListBox1.Items.Assign(MyList);
  finally
    MyList.Free;
```

```
  end;
  end;
end.
```

Burada, **Create** metodu ilə MyList siyahısı yaradılır. Siyahının elementləri Add metodu ilə əlavə edilir.

Add (const *s* **: string) : integer; funksiyası** – *s* parametri ilə verilən sətri (mətni) siyahının sonuna *əlavə edir* və nəticə kimi siyahıda yeni elementin vəziyyətini müəyyən edir. Yeri gəlmişkən qeyd edək ki, elementi əlavə etmək üçün Insert metodu da tətbiq oluna bilər.

Insert (*index* **:integer; const** *s* **: string); funksiyası** – *s* parametri ilə verilən sətri *index* parametri ilə göstərilən nömrəli mövqeyə əlavə edir.

MyList siyahısı yaradıldıqdan sonra, Assign metodu ilə ListBox1 komponentinə mənimsədilir.

Assign (*source* **: TPersistent); proseduru** – uyğun tipli bir obyekti digər obyektə mənimsədir. Baxılan nümunə MyList siyahısını ListBox1 komponentinə köçürür. Nəhayət, proqramın sonunda Create metodu ilə yaradılan siyahının tutduğu yaddaş azad edilir (Free).

Bu proqramda siyahılara heç bir aidiyyəti olmayan try və finally operatorları istifadə edilmişdir. Bu operatorları biz sonra öyrənəcəyik, hələlik isə onunla kifayətlənək ki, bu operatorların köməyi ilə digər operatorlarda hər hansı anlaşılmazlıqlar olsa belə, proqram yerinə yetiriləcəkdir. try…finally bloku end; operatoru ilə yekunlaşmalıdır.

Siyahının ayrı–ayrı sətirlərinə Items massivinin nömrəsi ilə müraciət etmək olar. Sətirlərin nömrəsi sıfırdan başladığı üçün *1*–ci sətrə müraciət Items[0], *2*–ci sətrə müraciət Items[1] və s. kimi yerinə yetirilir. Siyahıda olmayan sətrə müraciət etmək olmaz. Məsələn, siyahı *20* sətirdən ibarətdirsə, *21*–ci və sonrakı sətirlərə müraciət səhvə gətirəcəkdir.

Siyahıda elementlərin sayı Integer tipli Count xassəsi ilə təyin olunur. Siz Obyektlər inspektorunda bu xassəni görməyəcəksiniz. Çünki, bu xassə yalnız oxumaq üçündür, onun qiymətini daxil etmək olmaz. Siyahıda olan elementlərin sayı avtomatik olaraq bu xassəyə mənimsədilir. Birinci elementin nömrəsi *0* olduğu üçün sonuncu elementin sıra nömrəsi Count–1 olur.

Maus və klaviatura vasitəsilə istifadəçi siyahının *ayrı–ayrı sətirlərini seçə bilər*. Bunun üçün Integer tipli ItemIndex xassəsindən istifadə etmək lazımdır. Proqram yolu ilə sətri seçdikdə proqramçı bu xassəyə özü qiymət verməlidir, məsələn,

```
ComboBox1.ItemIndex:= 4;
```

kodu ilə siyahıda *5*–ci sətir seçiləcəkdir. Əgər ItemIndex xassəsinin qiyməti (*–1*) olarsa, bu o deməkdir ki, siyahıda heç bir sətir seçilməyəcəkdir, əgər *0* olarsa, siyahıda birinci sətir seçiləcəkdir və nəhayət bu xassənin qiyməti

sətirlərin faktiki sayından çox qiymət alarsa, onun qiyməti (*–1*) kimi qəbul ediləcəkdir.

`Integer` tipli `SelCount` xassəsi siyahıda *seçilmiş elementlərin sayını* təyin edir. Seçilmiş elementlərin *nömrəsinə* isə `Boolean` tipli `Selected(index:integer)`; xassəsi ilə baxmaq olar. Bu zaman `index` nömrəli sətir seçilmişdirsə, onda `Selected` xassəsinin qiyməti *True*, seçilmədikdə isə *False* olur.

`Equals (`*strings*** `: TStrings) : Boolean;` funksiyası** – iki siyahını müqayisə etmək üçün tətbiq edilir. Əgər hər iki siyahının məzmunu eynidirsə, onda bu funksiyanın qiyməti *True*, əks halda isə *False* olur. İki siyahı o vaxt eyni olur ki, siyahıların uzunluqları bərabər olsun və bütün elementlər üst–üstə düşsün.

`Delete (`*index*** `: integer);` proseduru** – *index* parametri ilə göstərilən nömrəli elementi *pozur*.

Misal.
```
ComboBox1.Items.Delete(4);
```

`Clear;` – proseduru bütün elementləri pozaraq siyahını *təmizləyir*.

`Move (`*CurIndex, NewIndex*** `: integer);` proseduru** – *CurIndex* nömrəli elementi *NewIndex* nömrəli *mövqeyə* yerləşdirir.

`IndexOf (const `*s*** `: string) : integer;` proseduru** – siyahıda *s* sətrinin *olmasını* yoxlayır. Əgər siyahıda belə bir sətir tapılarsa, həmin sətrin nömrəsi, əks halda isə (*–1*) qiyməti göstərilir.

Siyahıları *yadda* saxlamaq və ya onları mətn tipli fayllardan *yükləmək* üçün, redaktə komponentlərində olduğu kimi, uyğun olaraq **`SaveToFile`** və **`LoadFromFile`** prosedurları tətbiq olunur. Biz indiyədək baxdığımız misal və izahatlarda siyahıların yalnız proqram yolu ilə yaradılmasından və onlara əlavələrin edilməsindən bəhs etdik. Halbuki, siyahıları *String List Editor* (*Sətirlər redaktoru*) redaktoru vasitəsilə də konstruksiya etmək olar. Bu redaktoru çağırmaq üçün Obyektlər inspektorunda `TStrings` tipli xassələrin (məsələn, `Items`) qiymətlər oblastında mausun düyməsini iki dəfə basmaq lazımdır. Sətir redaktoru siyahıya yeni sətirlər əlavə etməyə, onları pozmağa və məzmununu dəyişməyə imkan verir.

12.6. Siyahı və mətn redaktorlarının tətbiqinə aid misallar

Misal. Sadə siyahıya elementlərin əlavə edilməsi və onların seçilməsi.

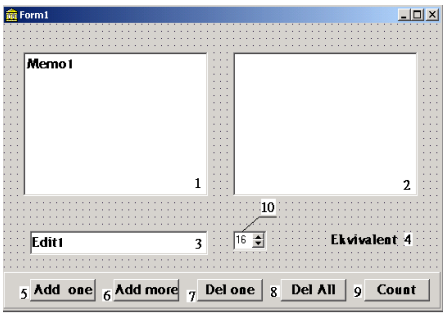
Forma üzərində `ListBox`, `Button` və `Label` komponentləri yerləşdirin. `Button` düyməsinin sərlövhəsini `Əlavə et…` adlandırıb, `OnClick` hadisəsini aktivləşdirərək yazın:

```
procedure TForm1.Button1Click(Sender: TObject);
Var i:LongInt;          // Dəyişən ixtiyari ola bilər
begin
  for i:=0 to 100 do
  ListBox1.Items.Add('Sətir №'+IntToStr(i));
  SendMessage(ListBox1.Handle,
              LB_SetHorizontalExtent,1000,0);
  ListBox1.ItemIndex:=0;
end;
```

Bu prosedurla sadə siyahıya `Sətir №0`, `Sətir №1`, ..., `Sətir №100` elementləri əlavə edilir. Siyahıya hər iki fırlatma zolağı yerləşdirilir və siyahıda birinci (*0* nömrəli) element seçilir. Layihə başa çatmamış, elə bu mərhələdə, ***F9*** klavişini basmaqla, bunları yoxlaya bilərsiniz. İndi isə `ListBox1` komponenti üzərində mausun düyməsini iki dəfə basaraq yunitə aşağıdakı kodları əlavə edin:

```
procedure TForm1.ListBox1Click(Sender: TObject);
begin
  Label1.Caption:= IntToStr(ListBox1.ItemIndex)+'    '+
                   ListBox1.Items[ListBox1.ItemIndex];
end;
```

F9 klavişini basaraq layihəni yerinə yetirin. Siz `Əlavə et…` düyməsini hər dəfə basdıqda siyahıya *101* sətir əlavə olunacaq, maus və ya klaviatura ilə bu elementlərdən hər hansı birini seçdikdə `Label` yazısı üzərində onun nömrəsi və məzmunu təsvir olunacaqdır.



Şəkil 12.3. Bir neçə komponentin qarşılıqlı təsirlərinin təşkili

Misal. Sadə siyahının redaktə komponentləri ilə əlaqəsi.

Forma üzərində şəkil 12.3 – də göstərilən komponentləri yerləşdirin. Burada aşağıdakı komponentlər təsvir edilmişdir: *1*–`Memo1`; *2*–`ListBox1`; *3*–`Edit1`; *4*–`Label1`; *5*–*9*– `Button1`– `Button5`; *10* – **`Samples`** səhifəsindən **`SpinEdit1`**. Düymələrin yerini dəyişməməsi üçün onları `Panel1` paneli üzərində yerləşdirin. `Panel1` komponentinin `Align` xassəsinə `alBottom` qiyməti verin.

`Edit1` komponentini seçib, *Events* səhifəsindən `OnExit` hadisəsinin qarşısında mausun düyməsini iki dəfə basaraq, Kod redaktoruna aşağıdakı proqramı yazın:

```
procedure TForm1.Edit1Exit(Sender: TObject);
```

```
Var
   s:String;
begin
  s:= Edit1.Text;
  s:= Trim(s);
  if s= ''then
  begin
    ShowMessage('Edit-də mətn yoxdur!');
    Button1.Enabled:=False;
  end
    else Button1.Enabled:=True;
end;
```

Bu prosedur `Edit` komponenti fokusu itirdikdə onun sahəsində mətnin olmasını yoxlayır: əgər sahədə mətn olarsa, heç nə baş verməyəcək (sadəcə `Button1` aktiv olacaq), mətn sahəsi boş olduqda isə bu barədə məlumat veriləcək və `Button` düyməsi qoşulmayacaqdır (`Button1.Enabled:= False;`). Bu onun üçün edilir ki, siyahıya boş sətir daxil edilməsin. *Probel* klavişini basdıqda siyahıya boş sətrin daxil edilməsinin qarşısını almaq üçün `Trim` funksiyasından (*Əlavəyə* bax) istifadə edilmişdir**.**

Redaktorun sahəsində dəyişiklərə nəzarət etmək üçün `OnChange` hadisə emaledicisini yaradaq. Əslində bu hadisəni `OnExit` hadisəsi ilə əlaqələndirmək olar. Bunun üçün `Edit` komponentini seçib `OnChange` hadisəsi qarşısında mausun düyməsini iki dəfə basmaq yox, siyahıdan artıq mövcud olan `Edit1Exit` hadisəsini seçmək lazımdır. Bununla da, `OnChange` hadisəsi baş verdikdə, `OnExit` hadisə emaledicisində baş verən əməliyyatlar yerinə yetiriləcəkdir. Biz, burada, bir neçə hadisənin bir prosedurla necə yerinə yetirilməsini izah etdik.

Adi qaydada isə `OnChange` hadisə emaledicisini belə yaratmaq olar. `Edit1` komponentini seçib, həmişəki kimi, `OnChange` hadisəsi qarşısında mausun düyməsini iki dəfə basaraq koda əlavə edin:

```
procedure TForm1.Edit1Change(Sender: TObject);
begin
  Edit1.OnExit(Parent);
end;
```

Burada, faktiki olaraq bir hadisədə başqa bir hadisə emaledicisi çağrılmışdır. Parametr kimi **`Parent`** (əcdad), **`Self`** (obyektin özü) və **`Nil`** (heç nə) istifadə oluna bilər (layihənin başa çatmasını gözləmədən ***F9*** klavişini basaraq bu iki prosedurun gördüyü işi yoxlaya bilərsiniz).

İndi isə `SpinEdit1` komponentini seçək. Qısaca bu komponentlə tanış olaq. Xarici görünüşü və funksional imkanlarına görə bu komponent özündə `UpDown` sayğacını və onunla assosiativ əlaqədə olan `Edit` komponentlərini birləşdirir. Bu komponent üçün əsas xarakterik xassələr `Integer` tipli `Value` (qiymət), `MinValue` (minimal qiymət), `MaxValue` (maksimal qiymət),

`Increment` (addım), `Boolean` tipli `ReadOnly` xassələri və `OnChange` hadisəsidir. Beləliklə, `SpinEdit` komponentinin `Value` xassəsinə *16* qiyməti verin. Bu o deməkdir ki, siyahı tərtib edildikdə əlavə olunan sətirlərin sayı *16* olacaqdır (kodla bunu dəyişmək də olar).

`Button1` (`Add one`) düyməsi üzərində mausun düyməsini basaraq yunitə əlavə edin:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  ListBox1.Items.Add(Edit1.Text);
end;
```

Yunitə yazılmış bu yeganə sətir `Edit1` komponentindən daxil edilən mətni `ListBox1` siyahısına əlavə edir. Bu zaman siyahıya eyni sətirlər daxil edilə bilər. Bunun qarşısını almaq üçün həmin proseduru belə yaratmaq daha məqsədəuyğun olar:

```
procedure TForm1.Button1Click(Sender: TObject);
Var
   s:String;
begin
  if ListBox1.Items.Count=0 then
  begin
    ShowMessage('Siyahı boşdur, sətri daxil edin!');
    ListBox1.Items.Add(Edit1.Text);
    s:= ListBox1.Items[ListBox1.Items.Count-1];
    Edit1.SetFocus;
    Exit;
  end
    else s:= ListBox1.Items[ListBox1.Items.Count-1];
  if s= Edit1.Text then
    begin
      ShowMessage('Sətir təkrarlanır!');
      Exit;
    end;
  ListBox1.Items.Add(Edit1.Text);
end;
```

Burada, `Add one` düyməsini basdıqda yeni daxil edilən sətir özündən əvvəlki sətirlə müqayisə edilir. Əgər sətirlər eyni olarsa, `Exit` proseduru çağrılır və kodun yerinə yetirilməsi dayandırılır. Başlanğıc anda, yəni siyahıda element olmadıqda (`Count=0`), `Items` massivinin indeksində qeyri–müəyyənlik olmaması üçün belə yoxlama boş siyahı üçün də yerinə yetirilir, bu haqda istifadəçi məlumatlandırılır və mətnin daxil edilməsi üçün `Edit1` komponentinə fokus verilir.

`Button2` (`Add More`) düyməsi üzərində mausun düyməsini iki dəfə basaraq koda əlavə edin:

```
procedure TForm1.Button2Click(Sender: TObject);
Var
   I:LongInt;
begin
  for i:=0 to SpinEdit1.Value do
  ListBox1.Items.Add(IntToStr(i)+'_'+Edit1.Text);
end;
```

`Add More` düyməsini basdıqda `SpinEdit` komponentində müəyyən edilmiş sətirlərin sayı qədər eyni sətir siyahıya əlavə edilir. Əyanilik üçün hər sətrin nömrəsi (`…IntToStr(i)`) də siyahıya daxil ediləcəkdir.

`Button3` (`Del one`) düyməsini iki dəfə basaraq yazın:

```
procedure TForm1.Button3Click(Sender: TObject);
Var
  CurrentIndex:LongInt;   // Dəyişənin adı sərbəst seçilmişdir
begin
  CurrentIndex:= ListBox1.ItemIndex;
  if ListBox1.ItemIndex= -1 then Exit;
  ListBox1.Items.Delete(CurrentIndex)
end;
```

`Del one` düyməsini basdıqda bu prosedur siyahıda seçilmiş sətri pozur. Əgər heç bir sətir seçilməmişdirsə, onda `Exit` proseduru çağrılır.

`Button4` (`Del All`) düyməsini basdıqda `ListBox` siyahısında olan elementlər pozulmalıdır, bu kod belə yazılacaqdır:

```
procedure TForm1.Button4Click(Sender: TObject);
begin
  ListBox1.Clear;
end;
```

`Button5` (`Count`) düyməsi üçün bu kodu yazın:

```
procedure TForm1.Button5Click(Sender: TObject);
Var
   m,l:LongInt;
begi

  m:= Memo1.Lines.Count;
  l:= ListBox1.Items.Count;
  ShowMessage('Memo Komponentində '+IntToStr(m)+
  ' sətir var '+#13#10+'ListBox Komponentində'
  +IntToStr(l)+' sətir var ');

end;
```

Kodun təsvirindən onun yerinə yetirdiyi funksiya aydın olduğu üçün əlavə izaha ehtiyac görmürük.

Nəhayət, sonuncu Label (Ekvivalent) komponenti üzərində mausun düyməsini iki dəfə basaraq bu proseduru yaradın:

```
procedure TForm1.Label1Click(Sender: TObject);
begin
  Memo1.Lines:= ListBox1.Items;
end;
```

Ekvivalent yazısı üzərində mausun düyməsini basdıqda ListBox1 komponentində olan elementlər Memo komponentinə köçürüləcəkdir.

Misal. İki sadə siyahı arasında əlaqənin təşkili.

Ms Windows–da, xüsusən *Ms Excel* və *Ms Access*–də bir çox hallarda bir siyahıdan müəyyən əlamətlərə görə elementlər seçilərək digər siyahıda yerləşdirilir və ya geri qaytarılır. Bu məsələni proqramlaşdıraq. Bunun üçün formaya iki ListBox, iki Label və iki Button düymələri yerləşdirin (şəkil 12.4).

Şəkil 12.4. Siyahılar arasında əlaqə

Məsələnin mahiyyəti ondan ibarətdir ki, birinci siyahı fənlərin adları ilə doldurulur, layihə işə salındıqdan sonra ikinci siyahı təmizlənir. Hər iki siyahıda bir neçə element seçilə bilər. Sağa sərlövhəli düymə basıldıqda birinci siyahıdan seçilən element ikinci siyahıya köçürülür. Sola sərlövhəli düyməni basdıqda isə ikinci siyahıda seçilmiş element birinci siyahıya köçürülür. Bu köçürmələri mausla da (*drag–and–drop* texnologiyası ilə) yerinə yetirmək olar.

Layihədə komponentlərin sərlövhələrini şəklə uyğun müəyyənləşdirin (Form – Fənn və imtahanlar, Button1–Sağa, Button2–Sola). Button1 düyməsinin Name xassəsinə btnRight, Button2 düyməsinin Name xassəsinə btnLeft adları müəyyən edin (komponentlərin Name xassəsinə adları yalnız latın hərflərindən istifadə etməklə təyin etmək olar). Name xassəsi istifadə olunduqda prosedurların sərlövhəsində komponentin öz adı deyil, Name xassəsində göstərilən ad yazılır, məsələn,

```
procedure TForm1.btnRightClick(Sender:TObject);
```

ListBox1 komponentini seçib, Obyektlər inspektorunda Items xassəsinin qarşısında mausun düyməsini basaraq açılan *String List Editor* pəncərəsindən fənlərin adlarını daxil edin (bu adları koddan da daxil etmək olar, bu halda hər bir fənn üçün prosedurda ListBox1.Items.Add(

'İnformatika'); və s. yazmaq lazımdır). Formanın boş sahəsində mausun düyməsini iki dəfə basaraq yunitə aşağıdakı kodları yazın.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Label1.FocusControl:=ListBox1;
  Label2.FocusControl:=ListBox2;
  ListBox1.Sorted:=False;          // Düzləndirmə qadağan
  ListBox2.Sorted:=False;          // edilir
  ListBox1.MultiSelect:=True;      // Bir neçə elementin
  ListBox2.MultiSelect:=True;      // seçilməsinə icazə verilir
  ListBox1.ExtendedSelect:=True;   // Klaviatura ilə elementin
  ListBox2.ExtendedSelect:=True;   // seçilməsinə icazə verilir
  ListBox2.Clear;
  ListBox1.DragMode:=dmAutomatic;  //Mausla elementlərin
                                   // yerlərinin dəyişdirilməsi
  ListBox2.DragMode:=dmAutomatic;  //əməliyyatını avtomatik
                                   //başlamağa icazə  verilir
end;
```

Kodlara əlavə edilmiş şərhlər və hər bir xassənin indiyədək verilmiş ətraflı izahı bu prosedurun yerinə yetirdiyi əməliyyatları dərk etməyə imkan verir.

Button1 düyməsi üzərində mausun düyməsini iki dəfə basaraq seçilmiş elementləri ikinci siyahıya köçürən proseduru yaradın:

```
procedure TForm1.btnRightClick(Sender: TObject);
Var
   i:Integer;
begin
  for i:= ListBox1.Items.Count-1 downto 0 do
  if ListBox1.Selected[i] then
  begin
    ListBox2.Items.Add(ListBox1.Items[i]);
    ListBox1.Items.Delete(i);
  end;
end;
```

Bu prosedur icra olunduqdan sonra, Sağa düyməsini basdıqda, ListBox1 siyahısında seçilmiş elementlər ListBox2 siyahısına köçürülür və birinci siyahıdan həmin element pozulur. Dövrün (for) təşkilində elementlərin araşdırılması sonuncu elementdən (Count–1) başlayır. Bu ona görə belə edilir ki, element pozulacaq, lakin, dövrlərin sayı dəyişməyəcəkdir. Bu isə səhvə gətirəcəkdir. Elementin seçilməsi Selected xassəsi ilə yoxlanır.

Sola düyməsini basdıqda ikinci siyahıda seçilmiş elementlər birinci siyahıya köçürülməlidir. Ona görə də analoji prosedur Sola düyməsi üçün yaradılmalıdır. Bu düymə üzərində mausun düyməsini iki dəfə basaraq

yuxarıdakı prosedurda ListBox1 əvəzinə ListBox2 və tərsinə – ListBox2 əvəzinə ListBox1 yazmaq lazımdır.

İndi isə elementlərin mausla köçürülməsi prosedurlarını yaradaq. Element ikinci siyahıya köçürüldüyü üçün, qəbuledici komponent kimi ListBox2 siyahısını seçib OnDragOver hadisəsi qarşısında mausun düyməsini iki dəfə basaraq bu kodları yazın:

```
procedure TForm1.ListBox2DragOver(Sender,
          Source:TObject;X,Y:Integer;State:TDragState;
          var Accept:Boolean);
begin
  if Source= ListBox1 then Accept:=True
  else Accept:=False;
end;
```

Bu prosedur mausla elementin yerini dəyişdirməyə icazə verilməsini müəyyən edir.

Mausun düyməsini buraxdıqda komponentin ikinci siyahıda yerləşdirilməsi hadisə emaledicisini (OnDragDrop) yaradaq. Yenə də ListBox2 komponentini seçərək bu kodları yazın:

```
procedure TForm1.ListBox2DragDrop(Sender,
          Source:TObject;X,Y:Integer);
Begin
  With Source as TListBox do
  begin
    ListBox2.Items.Add(Items[ItemIndex]);
    Items.Delete(ItemIndex);
  end;
end;
```

Bu prosedur icra olunduqda həmişə sonuncu seçilmiş element köçürüləcəkdir. Çünki, burada elementin seçilməsində Selected xassəsi deyil, ItemIndex xassəsi istifadə edilmişdir.

İndi isə DragOver və DragDrop hadisə emaledicilərini ListBox1 komponenti üçün yaratmaq lazımdır. Burada da müvafiq prosedurların kodlarında ListBox1 əvəzinə ListBox2 və tərsinə – ListBox2 əvəzinə ListBox1 yazmaq lazımdır.

F9 klavişini basaraq layihəni yerinə yetirin və nəticələri yoxlayın. Görəcəksiniz ki, elementlərin düymələrlə və mausla yerlərinin dəyişdirilməsi əməliyyatı bir–birindən fərqli qaydada yerinə yetirilir.

Mausla və düyməni basmaqla elementlərin yerlərinin dəyişdirilməsinin eyni qayda ilə yerinə yetirilməsi üçün, DragDrop hadisə emaledicilərində, uyğun düymələr üçün, OnClick hadisə emaledicisinin kodlarını yazmaq lazımdır. Bu prosedur iki formada yazıla bilər:

```
procedure TForm1.ListBox2DragDrop(Sender,
          Source:TObject;X,Y:Integer);
begin
  btnRight.Click;     // O biri düymə üçün btnLeft.Click;
end;
```

və ya

```
procedure TForm1.ListBox2DragDrop(Sender,
          Source:TObject;X,Y:Integer);
begin
  btnRightClick(Sender);
end;
```

Beləliklə, siyahılar arasında element mübadiləsini icra edən proqramın tam mətni aşağıdakı kimi olacaqdır:

```
unit Unit1;

interface

uses
   Windows,Messages,SysUtils,Classes,Graphics,
   Controls,Forms,Dialogs,StdCtrls;

Type
  TForm1 = class(TForm)
    ListBox1: TListBox;
    ListBox2: TListBox;
    btnRight: TButton;
    btnLeft: TButton;
    Label1: TLabel;
    Label2: TLabel;
    procedure FormCreate(Sender:TObject);
    procedure btnRightClick(Sender:TObject);
    procedure btnLeftClick(Sender:TObject);
    procedure ListBox2DragOver(Sender,Source:TObject;
    X,Y:Integer;State: TDragState; var Accept:Boolean);
    //procedure ListBox2DragDrop(Sender,Source:TObject;
                                X,Y:Integer);
    procedure ListBox1DragOver(Sender,Source:TObject;
    X,Y:Integer;State: TDragState;var Accept:Boolean);
    procedure ListBox2DragDrop(Sender, Source: TObject;
                                X,Y:Integer);
    procedure ListBox1Click(Sender:TObject);
    //procedure ListBox1DragDrop(Sender,Source:TObject;
                                X,Y:Integer);

  private
    { Private declarations }
```

```
  public
    { Public declarations }

  end;

var
   Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender:TObject);
begin

 Label1.FocusControl:=ListBox1;
 Label2.FocusControl:=ListBox2;
 ListBox1.Sorted:=False;          // Düzləndirmə qadağan
 ListBox2.Sorted:=False;          // edilir
 ListBox1.MultiSelect:=True;      // Bir neçə elementin
 ListBox2.MultiSelect:=True;      // seçilməsinə icazə verilir
 ListBox1.ExtendedSelect:=True;   // Klaviatura ilə elementin
 ListBox2.ExtendedSelect:=True;   // seçilməsinə icazə verilir
 ListBox2.Clear;
 ListBox1.DragMode:=dmAutomatic;  // Mausla elementlərin
                                  //yerlərinin dəyişdirilməsi
 ListBox2.DragMode:=dmAutomatic;  // əməliyyatını avtomatik
                                  //başlamağa icazə verilir
end;


procedure TForm1.btnRightClick(Sender:TObject);
Var
   i:Integer;
begin
 for i:= ListBox1.Items.Count-1 downto 0 do
 if ListBox1.Selected[i] then
 begin
   ListBox2.Items.Add(ListBox1.Items[i]);
   ListBox1.Items.Delete(i);
 end;
end;


procedure TForm1.btnLeftClick(Sender:TObject);
Var
   i:Integer;
```

```
begin
  for i:= ListBox2.Items.Count-1 downto 0 do
  if ListBox2.Selected[i] then
  begin
    ListBox1.Items.Add(ListBox2.Items[i]);
    ListBox2.Items.Delete(i);
  end;
end;


procedure Form1.ListBox2DragOver(Sender,Source:
           TObject;X,Y:Integer;State:TDragState;
           var Accept: Boolean);
begin
  if Source= ListBox1 then Accept:=True
  else Accept:=False;
end;


{procedure TForm1.ListBox2DragDrop(Sender,
            Source:TObject;X,Y:Integer);
begin
  With Source as TListBox do
    begin
      ListBox2.Items.Add(Items[ItemIndex]);
      Items.Delete(ItemIndex);
    end;
end; }


procedure TForm1.ListBox1DragOver(Sender,Source:
           TObject;X,Y:Integer;State:TDragState;
           var Accept:Boolean);
begin
  if Source= ListBox2 then Accept:=True
  else Accept:=False;
end;


{procedure TForm1.ListBox1DragDrop(Sender,
            Source:TObject;X,Y:Integer);
begin
  With Source as TListBox do
    begin
       ListBox1.Items.Add(Items[ItemIndex]);
       Items.Delete(ItemIndex);
    end;
end; }
```

```
procedure TForm1.ListBox2DragDrop(Sender,
            Source:TObject;X,Y:Integer);
Begin
  // btnRight.Click; və ya
  btnRightClick(Sender);
end;


procedure TForm1.ListBox1Click(Sender:TObject);
begin
  // btnLeft.Click; və ya
  btnLeftClick(Sender);
end;
end.
```

Proqramın mətnində elementlərin maus və düymə vasitəsilə fərqli və eyni qayda ilə yerinə yetirilməsi kodlarının hər iki variantı göstərilmişdir. Bu prosedurlar eyni sərlövhəli, müxtəlif məzmunlu olduqları üçün onları eyni zamanda icra etmək mümkün deyildir. Ona görə də birinci hala uyğun prosedurlar kursivlə göstərilərək şərh simvolları (**{ }**) daxilinə salınmışdır.

12.7. Düymələrlə iş

Düymələr idarəedici elementlər olaraq müəyyən funksiyaları yerinə yetirmək üçün əmrlər vermək məqsədilə istifadə olunur. Ona görə də onları çox vaxt əmrlər düymələri də adlandırırlar. Delphi aşağıdakı düymələri təklif edir:

- `Button` *standart düyməsi*;
- `BitBtn` *şəkilli düyməsi*;
- `SpeedButton` *cəld müdaxilə düyməsi.*

Bu düymələrin zahiri görünüşü və funksional imkanları çox az fərqlənir.

12.7.1. Standart düymə

`Button` standart düyməsi pəncərəli idarəetmə elementidir və onun üzərində yerinə yetirdiyi funksiyanın mahiyyətinə uyğun yazı ola bilər. Bu düyməyə xüsusi müzakirə mövzusu kimi baxanadək, biz artıq onunla tanış olmuşuq və demək olar ki, həll etdiyimiz bütün məsələlərdə onu tətbiq etmişik. Bizə artıq məlumdur ki, `Button` düyməsi üçün əsas hadisə mausun düyməsini basdıqda baş verən `OnClick` hadisəsidir. Bu zaman düymə onun yerinə yetirəcəyi hadisəyə uyğun görkəm alır (yəni basılır) və düyməni buraxan kimi bu hadisə dərhal yerinə yetirilir. Mausun düyməsini basmaqla, `Caption`

xassəsində müəyyən edilmiş klavişlər kombinasiyasını basmaqla və nəhayət *Enter* və ya *Probel* klavişlərini basmaqla `Button` düyməsini basmaq olar. Bundan başqa, *Esc* klavişini basdıqda da `OnClick` hadisəsi baş verə bilər.

Enter və *Probel* klavişləri ilə yalnız fokus almış düymə (adı qırıq xətli düzbucaqlı ilə əhatə olunmuş) basılır. Əgər düymə yox, başqa pəncərəli element, məsələn, `Edit` və ya `Memo` komponenti fokus almışdırsa, onda `Default` xassəsi *True* qiyməti almış düymə susmaya görə seçilmiş olur; bu düymə qara düzbucaqlı ilə əhatələnir.

Esc klavişi ilə adətən dialoq pəncərələrindəki `Cancel` (*imtina*) düyməsi basılır. Düymənin *Esc* klavişinə məhəl qoyması üçün onun `Cancel` xassəsinə *True* qiyməti vermək lazımdır.

Dialoq pəncərələrini bağlamaq məqsədilə düyməni tətbiq etdikdə, onun `TModalResult` tipli `ModalResult` xassəsindən istifadə etmək olar. Bu xassə aşağıdakı qiymətləri ala bilər: `mrNone`, `mrOk`, `mrCancel`, `mrAbort`, `mrRetry`, `mrIgnore`, `mrYes`, `mrNo`, `mrAll`, `mrNoToAll`, `mrYesToAll`. `ModalResult` xassəsinə susmaya görə `mrNone` qiyməti mənimsədilir. Əgər bu xassəyə `mrNone` qiymətindən fərqli istənilən qiymət mənimsədilərsə, onda forma `Close` metodu çağrılmadan avtomatik olaraq bağlanacaqdır.

Misal. Mausdan qaçan düymə.

Biz elə proqram yazacağıq ki, mausu düyməyə yönəltdikdə o, mausdan qaçacaqdır. Bunun üçün forma üzərində yeganə komponent – `Button1` yerləşdirərək, onun üçün `OnMouseMove` hadisəsini yaradaq. Bu məsələnin yuniti aşağıdakı kodlardan ibarət olacaqdır:

```
unit Unit1;

interface

uses
     Windows,Messages,SysUtils,Variants,Classes,
     Graphics,Controls,Forms,Dialogs,StdCtrls;

type
   TForm1 = class(TForm)
     Button1: TButton;
     procedure Button1MouseMove(Sender: TObject;
            Shift: TShiftState; X, Y: Integer);
   private
     { Private declarations }
   public
     { Public declarations }
   end;

var
  Form1: TForm1;

implementation
```

```
{$R *.dfm}

procedure TForm1.Button1MouseMove(Sender: TObject;
            Shift: TShiftState; X,Y: Integer);

Var
     index:integer;
begin
  index:=random(4);

  case index of
    0: Button1.Left:=Button1.Left+Button1.Width;
    1: Button1.Left:=Button1.Left-Button1.Width;
    2: Button1.Top:=Button1.Top+Button1.Height;
    3: Button1.Top:=Button1.Top-Button1.Height;
  end;

  if Button1.Left<0 then Button1.Left:=0;
  if Button1.Left+Button1.Width>Form1.Width then
     Button1.Left:=Form1.Width-Button1.Width;
  if Button1.Top<0 then Button1.Top:=0;
  if Button1.Top+Button1.Height>Form1.Height then
     Button1.Top:=Form1.Height-Button1.Height;

end;
end.
```

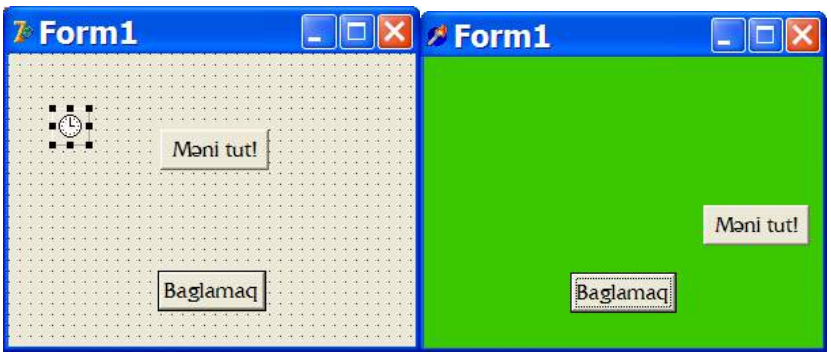
Əgər indiyədək oxuduqlarınızı yaxşı başa düşmüşsünüzsə, onda yəqin ki, bu proqramı qavramaq Sizin üçün çətinlik törətməz. Bu proqramda tam tipli `index` dəyişəni `0≤index<4` aralığında təsadüfi qiymətlər alır. Sonra, `Case` operatorunda düymənin koordinatları (`Left` – sol, `Top` – yuxarı) müəyyən edilir. Sonrakı `if` operatorlarında isə düymənin formanın hüdudlarından (`Form1.Width, Form1.Height`) kənara çıxmamasına nəzarət edilir. ***F9*** klavişini basmaqla layihəni yerinə yetirin. Siz , düyməni basmağa nə qədər cəhd etsəniz də, o, Sizdən qaçacaqdır.

Misal. Əyləncəli oyun.

Bu misalda proqramlaşdırılacaq oyunun mahiyyəti ondan ibarətdir ki, proqramın pəncərəsi hüdudlarında düymə təsadüfi qaydada peyda olacaq və cəld yerini dəyişəcəkdir. Bu düyməni "tutmaq", yəni mausun düyməsini onun üzərində basmaq lazımdır. Düymənin gəzişməsi taymerdən (vaxt ölçən) gələn siqnala görə baş verəcəkdir.

Proqramın pəncərəsini tərtib etmək üçün üç obyekt lazımdır. Onlardan başlıcası yerini dəyişən düymədir. Digər düymə isə proqramdan çıxmaq üçün lazımdır. Üçüncü obyekt taymerdir. Forma üzərində o, saat nişanı ilə təsvir olunur. Taymer vizual komponent olmadığı üçün, layihə yerinə yetirildikdən sonra görünməyəcək, lakin, "kadr arxasından" öz işini görəcəkdir.

Beləliklə, forma üzərində `Button1` düyməsi yerləşdirin (şəkil 12.5). Bu düymənin sərlövhəsini (`Caption`) `Məni tut!` adlandırın. Onun `TabStop` xassəsinə *False* qiyməti verin. Çünki, bu oyun klaviatura ilə oynanılmır. Oyunun başlanğıcında düymə görünməməlidir, ona görə də `Visible` xassəsinə də *False* qiyməti verin. Düymənin yerdəyişmə koordinatlarını tam təsadüfi seçmək olar. Lakin, biz qəbul edək ki, düymə *9* yerdə peyda olacaqdır, yəni təsəvvür edin ki, formada *9* xana var və bu xanalarda düymə peyda ola bilər. Əsas məsələlərdən biri düymənin və xanaların ölçülərini (piksellə) düzgün seçməkdir. Düymənin `Height` (*hündürlük*) xassəsinə *30*, `Width` (*en*) xassəsinə isə *80* qiyməti daxil edin. Qalan ölçüləri isə şəkil 12.6 – ya uyğun seçək. Forma üzərində mausun düyməsini basaraq `Form1` komponentinin `ClientWidth` (*daxili en*) xassəsinə *300*, `ClientHeight` (*daxili hündürlük*) xassəsinə isə *200* qiyməti, `BorderStyle` (*haşiyə*) xassəsinə `bsSingle` (*haşiyə var*) qiyməti verin.



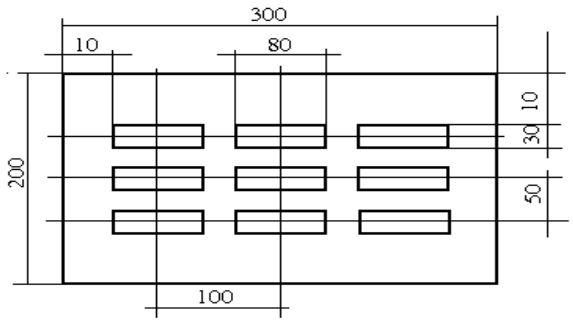
Şəkil 12.5. "Gəzişən düymə" oyununun layihəsi və nəticəsi

`Button2` düyməsinin sərlövhəsini `Bağlamaq` adlandıraraq `Default` xassəsinə *True* qiyməti verin. Bu halda düymə həmişə fokus altında olacaqdır. Düymənin ölçü və koordinatlarını isə belə təyin edin: `Left`–*110*, `Top`–*160*, `Width`–*80*, `Height`–*30*.

`System` səhifəsindən **`Timer`** komponentini seçərək onu forma üzərində, düymələrdən kənarda yerləşdirin. Taymerin əsas xassəsi `Interval` xassəsidir. Bu xassə taymerin qoşulma intervalını bildirir. İnterval saniyənin mində biri, yəni millisaniyələrlə verilir. Baxdığımız məsələ üçün `Interval` xassəsinə *500* qiyməti verin. Bu o deməkdir ki, düymə hər yarım saniyədən bir yerini dəyişəcəkdir. ***F9*** klavişini basın. Hazır forma üzərində Siz yalnız `Bağlamaq` düyməsini görəcəksiniz.

İndi isə proqramlaşdırma ilə məşğul olaq. Taymerlə bağlı yeganə **`OnTimer`** hadisəsi baş verir. Bu hadisə baş verdikdə düymə öz yerini dəyişməlidir. Düymələri şəkil 12.7 – də olduğu kimi nömrələyək. Düymələrin nömrəsi üzərinə *3* ədədi əlavə etsək, növbəti sətrə, *1* ədədi əlavə etdikdə isə növbəti sütuna keçirik. Bu qayda bizə imkan verir ki, düymələrin vəziyyətini belə hesablayaq:

```
Top:=10+50*(i div 3);
Left:=10+100*(i mod 3);
```



Şəkil 12.6. Xanaların sxemi

0	1	2
3	4	5
6	7	8

Şəkil 12.7. Düymələrin nömrələri

Burada, `i` dəyişəni ilə (`var i:integer;`) xananın nömrəsi işarə edilmişdir. Xananın nömrəsi `i` təsadüfi qaydada seçilməlidir:

```
i:=Random(9);
Button1.Visible:=True;
```

Sonuncu operator düyməni görünməyə məcbur edir.

`k` dəyişəni (`k:LongInt`) daxil edərək analoji qayda ilə formanın rəngini təsadüfi dəyişdirə bilərik:

```
k:=Random(65535);
Color:=k;
```

İndi isə `Button1Click` prosedurunu yaradaq. Bu hadisə yalnız o vaxt baş verə bilər ki, gəzişən düymə üzərində mausun düyməsini basmağa nail olaq. Məhz bu anda da oyun başa çatmalıdır. Bu əməliyyatlar prosedurda belə təsvir oluna bilər:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Button1.Caption:='QƏLƏBƏ!';
  Button1.Enabled:=False;
  Timer1.Enabled:=False;
end;
```

`Button2` düyməsi üçün `OnClick` hadisə emaledicisi yalnız formanı bağlamaqdan ibarət olacaqdır.

Növbəti prosedur təsadüfi ədədlərin təkrar olunmasının qarşısını almalıdır. Bu məqsədlə `Form1` forması üçün `OnCreate` proseduru yaradaraq `Randomize;` prosedurunu yazmaq lazımdır. Beləliklə, bu əyləncəli oyunun tam modulu belə görünəcəkdir:

```
unit Unit1;

interface

uses
   Windows,Messages,SysUtils,Classes,Graphics,
   Controls,Forms,Dialogs,ExtCtrls,StdCtrls;

type
 TForm1 = class(TForm)
   Button1: TButton;
   Button2: TButton;
   Timer1: TTimer;
   procedure Timer1Timer(Sender: TObject);
   procedure Button1Click(Sender: TObject);
   procedure Button2Click(Sender: TObject);
   procedure FormCreate(Sender: TObject);

  private
    { Private declarations }

  public
    { Public declarations }

 end;

var
   Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.Timer1Timer(Sender:TObject);
Var i:Integer;
    k:Longint;
begin

 i:=Random(9);
 k:=Random(65535);

 With Button1 do
    begin
       Visible:=True;
       Top:=10+50*(i div 3);
       Left:=10+100*(i mod 3);
       Color:=k;
    end;

end;
```

```
procedure TForm1.Button1Click(Sender:TObject);
begin
  Button1.Caption:=' QƏLƏBƏ! ';
  Button1.Enabled:=False;
  Timer1.Enabled:=False;
end;

procedure TForm1.Button2Click(Sender:TObject);
begin
  Form1.Close;
end;

procedure TForm1.FormCreate(Sender:TObject);
begin
  Randomize;
end;

end.
```

12.7.2. Şəkilli düymə

Şəkilli düymə Delphi–də `TBitBtn` sinifli **`BitBtn`** komponenti ilə təsvir olunur. Bu düymə `TButton` sinifli `Button` standart düyməsindən yaranmışdır. Şəkilli düymənin standart düymədən fərqi ondadır ki, düymənin üzərində sərlövhə ilə yanaşı şəkil də təsvir olunur. Düymədə şəklin təsvirini `TBitMap` tipli `Glyph` xassəsi müəyyənləşdirir. Susmaya görə düymənin şəkli olmur, ona görə də `Glyph` xassəsinin qiyməti `nil` olur. Şəkil üç ayrı–ayrı təsvirlərdən ibarət ola bilər. Düymənin üzərinə bu təsvirlərdən hansının çıxarılması düymənin aşağıdakı üç vəziyyətindən asılıdır:

- *düymə basılmadıqda birinci təsvir əks olunur (susmaya görə)*;
- *düymə aktiv olmadıqda və seçilə bilmədikdə ikinci təsvir əks olunur*;
- *düymə basıldıqda üçüncü təsvir əks olunur.*

Düymə üçün şəkillər *Image Editor* redaktoru ilə yaradılır. Delphi `BitBtn` düyməsi üçün əvvəlcədən şəkillər də müəyyənləşdirmişdir. Bu şəkillər `TBitBtnKind` tipli `Kind` xassəsi ilə seçilir. Bu xassə aşağıdakı qiymətləri ala bilər:

`bkCustom`–şəkli istifadəçi özü seçir, ilkin olaraq düymədə şəkil olmur;
`bkOk` –düymədə yaşıl rəngli ✓ işarəsi və **`Ok`** yazısı olur. Bu düymə üçün `Default` xassəsinə *True* qiyməti, `ModalResult` xassəsinə isə `mrOk` qiyməti verilir;
`bkCancel`–düymədə qırmızı rəngli **`X`** (xaç) işarəsi və **`Cancel`** sözü var. Burada, `Cancel` xassəsinə *True*, `ModalResult` xassəsinə `mrCancel` qiyməti mənimsədilir;
`bkYes` –düymədə yaşıl rəngli ✓ işarəsi və **`Yes`** yazısı var;
`bkNo` –düymədə qırmızı rəngli, üstündən xətt çəkilmiş çevrə (⃠) və **`No`** yazısı var;
`bkHelp` –düymədə göy–yaşıl rəngli **`?`** işarəsi və **`Help`** yazısı var;
`bkClose` –düymədə çıxışı göstərən qapı şəkli və **`Close`** yazısı var. Bu düyməni basdıqda forma avtomatik olaraq bağlanır;
`bkAbort` –düymədə qırmızı rəngli **`X`** (xaç) işarəsi və **`Abort`** yazısı var;
`bkRetry` –düymədə yaşıl rəngli təkraretmə əməliyyatı işarəsi (🔄) və **`Retry`** yazısı var;
`bkIgnore`–düymədə qəbul etməmək işarəsi (“dönüb gedən adam” şəkli) və **`Ignore`** yazısı var;
`bkAll` –düymədə yaşıl rəngli ✓ işarəsi və **`YesToAll`** yazısı var.

Əvvəlcədən müəyyənləşdirilmiş düymələr üçün `Glyph` xassəsini dəyişmək məsləhət görülmür. Çünki, bu halda düymə onun üçün nəzərdə tutulmuş funksiyanı yerinə yetirməyəcəkdir. Düymənin səthində yazıya nisbətdə təsvirin yerləşməsini `TButtonLayout` tipli `Layout` xassəsi müəyyənləşdirir. Bu xassə aşağıdakı qiymətləri ala bilər:

`blGlyphLeft` –*təsvir yazıdan solda (susmaya görə)* ✓ **Close**

`blGlyphRight` –*təsvir yazıdan sağda* **Close** ✓

`blGlyphTop` –*təsvir yazıdan yuxarıda* ✓ **Close**

`blGlyphBottom` –*təsvir yazıdan aşağıda* **Close** ✓

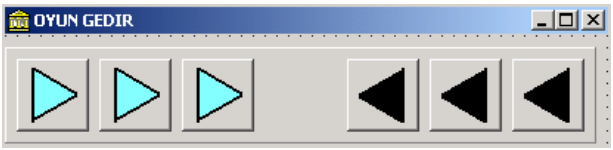
Bunlardan başqa, `BitBtn` düyməsi üçün `Margin` və `Spacing` (hər ikisi `integer` tipli) xassələri var. Bu xassələr uyğun olaraq təsvir və yazıları düymənin kənarlarına görə nizamlamaq və təsvirlə yazı arasındakı məsafəni (piksellə) müəyyənləşdirmək üçündür. Susmaya görə, hər iki xassənin qiyməti (*–1*)–ə bərabərdir, yəni təsvir və yazı düymənin mərkəzinə nisbətən simmetrik yerləşmişdir.

Şəkillər şəkilçəkmə redaktorlarında hazırlanır və *.bmp* tipli fayllarda saxlanır. Şəkilləri hazırlamaq üçün Delphi–nin tərkibinə *Image Editor* şəkilçəkmə redaktoru daxil edilmişdir.

Misal. Eduard Lyukun başsındıran məsələsi.

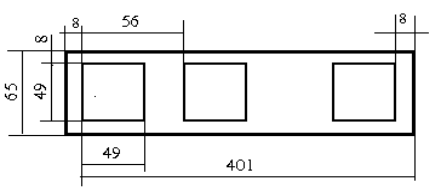
Bir çox başsındıran məsələlərinin müəllifi kimi tanınan fransız riyaziyyatçısı Eduard Lyukun ən geniş yayılmış əyləncəli məsələlərindən biri belədir. Oyun taxtası üzərində ardıcıl düzülmüş yeddi xana var. Sol tərəfdəki üç xanada ağ, sağ tərəfdəki üç xanada isə qara dama yerləşir. Mərkəzi (*4*–cü) xana boşdur. Məsələ ondan ibarətdir ki, oyunun qaydasına uyğun gedişlər etməklə, damaların yerini dəyişdirmək lazımdır. Qayda isə belədir: 1) ağ damalar yalnız sağa, qara damalar isə yalnız sola hərəkət edə bilər; 2) yalnız boş xanaya hərəkət etmək olar; 3) qonşu xanaya keçmək üçün yalnız bir xananın üstündən "tullanıb" keçmək olar.

Bu məsələdə damalar əvəzinə şəkilli düymələrdən istifadə edəcəyik. Düymələri fərqləndirmək üçün üzərində ağ və qara rəngli, hərəkət istiqamətini göstərən üçbucaqlı təsvirlərdən istifadə edək (şəkil 12.8).



Şəkil 12.8. Şəkilli düymələrin yerləşdirilməsi

Forma üzərinə bir panel və hələlik bir `BitBtn` düyməsi yerləşdirin. Panel və düymələrin ölçülərini şəkil 12.9–da olduğu kimi müəyyən edin. `Panel` komponentini seçərək Obyektlər inspektorunda `Height` (*hündürlük*) xassəsinə *65*, `Width` (*en*) xassəsinə isə *401* piksel qiyməti daxil edin. `Panel` komponentini formanın yuxarı hissəsində elə yerləşdirin ki, panellə onun arasında çox az məsafə qalsın. `BitBtn` düyməsini seçərək onun `Height` və `Width` xassələrinə *49* qiyməti daxil edin. Düymənin panel üzərində vəziyyətini dəqiqləşdirmək üçün `Left` (*soldan*) və `Top` (*yuxarıdan*) xassələrinə *8* qiyməti verin. Digər düymələri isə mübadilə buferinin köməyi ilə yerləşdirək (düymələrin hamısının eyni olması üçün). *Ctrl+C*, sonra isə *Ctrl+V* klavişlərini basın. Paneldə ikinci düymə yerləşəcək. *Ctrl+V* klavişlərini dörd dəfə də basın. İndi paneldə bütün düymələr olacaqdır. Hər yeni düyməni seçərək onların `Top` xassəsinə *8* qiyməti, `Left` xassəsinə isə uyğun olaraq *64, 120, 232, 288* və *344* (şəkil 12.9) qiymətləri daxil edin.



Şəkil 12.9. Şəkilli düymələrin ölçüləri

File/Save All (*Fayl/Hamısını yadda saxla*) əmrini icra edərək layihəni yadda saxlayın. İndi isə düymələr üzərində şəkil çəkək. Şəkillər çox sadə olduğu üçün *Paint* redaktorundan istifadə edək. Şəklin ölçüsü üçün *45*x*45* piksel seçib, böyütmə rejimində ağ və qara rəngli üçbucaqlar çəkin. Onları uyğun olaraq *Left.bmp* və *Right.bmp*

fayllarında yadda saxlamaqla, layihəni yadda saxladığınız qovluqda yerləşdirin. İndi isə bu şəkilləri növbə ilə düymələr üzərində yerləşdirək. Bunun üçün düymənin `Caption` xassəsində sərlövhəni pozun. Obyektlər inspektorunda `Glyph` (*Nişan*) xassəsinin qarşısındakı üç nöqtə təsvirli düyməni basın. Açılan *PictureEditor* (*Şəkil redaktoru*) pəncərəsində *Load* (*Yükləmək*) düyməsini basın. *Load Picture* (*Şəklin yüklənməsi*) dialoq pəncərəsində yuxarıdakı faylları seçərək *Open* (*Açmaq*), sonra isə *Ok* düyməsini basın. Seçilmiş şəkil düymənin üzərində təsvir olunacaqdır. İndi isə məsələni proqramlaşdıraq.

İstifadəçi oyuna başladıqda hər hansı bir düyməni seçəcəkdir, yəni düymələrin biri üzərində mausun düyməsini basacaqdır ki, bu da `OnClick` hadisəsinə uyğun gəlir. Biz düymələri fərqləndirə bilməliyik, həmçinin bilməliyik ki, düymələr və boş xana harada yerləşir. Bütün bu kəmiyyətləri biz obyekt–düymənin xassələrində yadda saxlayacağıq. Belə xassə kimi komponentlərə xas olan `Tag` (*tam ədəd*) xassəsindən istifadə edək. Proqramçı bu xassədən özü istədiyi məqsəd üçün istifadə edə bilər. Bu xassədə biz iki tam ədəd: düymənin yerləşdiyi xananın nömrəsini və düymənin rəngi əlamətini (hərəkət istiqaməti rənglə təyin olunur) bildirən ədədi saxlayacağıq. Bu iki ədədi bir ədəd kimi göstərmək olar: xananın nömrəsini *2*–yə vurub üzərinə ya *0* (ağ rəng), ya da *1* (qara rəng) əlavə etmək. `Tag` xassəsindəki qiyməti tərsinə araşdırmaqla, yəni onu *2*–yə tamədədli bölməklə xananın nömrəsini (qismət) və düymənin rəngini (qalıq) tapa bilərik. Beləliklə, düymələri ardıcıl olaraq seçməklə, `Tag` xassəsinə aşağıdakı tam ədədləri daxil edin: *2* (birinci xana – ağ), *4* (ikinci xana – ağ), *6* (üçüncü xana – ağ), *11* (beşinci xana – qara), *13* (altıncı xana – qara), *15* (yeddinci xana – qara). Göründüyü kimi hansı xananın boş olması məlum deyil. Bunun üçün ***F12*** klavişini basmaqla, yuniti ön plana keçirərək, `Var` bölməsinə

```
n:integer=4;
```

operatorunu daxil edin, yəni dördüncü xana boş elan edilir. Formada birinci düyməni seçərək onun üçün `OnClick` hadisə emaledicisini yaradaq. Bu emaledicidə prosedura `Sender` parametri ötürülür. Proqrama xüsusi olaraq göstərmək lazımdır ki, baxılan halda `Sender` obyekti düymədir. Bunun üçün `as` operatorundan istifadə olunur, yəni `Sender as TBitBtn`. Məhz hansı düymənin basılmasını `Tag` xassəsi ilə bilmək olar. Bu xassəyə müraciət (`Sender as TBitBtn`).`Tag` kimi yazılmalıdır. Proqramda `(Sender as TBitBtn)` konstruksiyasının təkrar olunmaması üçün `With` operatorunu tətbiq edəcəyik. Düymənin yerləşdiyi xananın nömrəsini

```
i:=(Sender as TBitBtn).Tag div 2;
```

kimi, düymənin rəngini isə

```
c:=(Sender as TBitBtn).Tag mod 2;
```

kimi hesablamaq olar.

Əgər gediş mümkündürsə, onda düymə `i` nömrəli xanadan `n` nömrəli xanaya keçməlidir. Beləliklə, yerdəyişməni və gedişin uzunluğunu

```
k:=n-i;
ak:=abs(k);
```

operatorları ilə hesablaya bilərik. Yeni əlavə edilmiş bu dəyişənləri prosedurun adı ilə `begin` operatoru arasında elan edin:

```
Var i,c,k,ak:Integer;
```

Oyunun qaydasına görə, gedişin maksimal uzunluğu ikidən çox ola bilməz, ona görə də

```
if ak<3 then
```

yazmalıyıq. Bu halda ağ düymələr yalnız sağa, qara düymələr isə sola hərəkət edə bilər, yəni

```
if ((c=0) and (k>0)) or ((c=1) and (k<0)) then
```

İndi isə gediş etmək lazımdır. `k` dəyişəni düymənin yerdəyişməsini bildirir. Gediş etdikdə `Tag` xassəsinin qiyməti yerdəyişmənin iki mislinə bərabər olmalıdır, düymənin yerini müəyyən etmək üçün isə `Left` (*Soldan*) xassəsinin üzərinə düymələr arasındakı məsafənin (*56*) `k` kəmiyyətinə hasilini əlavə etmək lazımdır:

```
Tag:=Tag+2*k;
Left:=Left+56*k;
```

İndi boş xana düymənin yerləşdiyi əvvəlki xana olacaqdır, yəni

```
n:=i;
```

Məlumdur ki, hər bir düymə *4* xanada hərəkət edə bilər. Deməli, ağ və qara düymələr birlikdə *24* xanada yerini dəyişə bilər. Bundan başqa, düymələr yalnız bir istiqamətdə hərəkət etdiyindən, bütün gedişlərin uzunluqları cəmi *24* – dən az olarsa, deməli məsələ həll edilməmişdir. Ona görə də *n* dəyişənini elan etdiyimiz hissəyə

```
Finish:integer=24;
```

əlavə edin (`n` və `Finish` dəyişənləri qlobal dəyişənlərdir). Hər gedişdə bu dəyişənin qiyməti azalmalıdır, ona görə də `n:=i;` operatorundan sonra

```
Finish:=Finish-ak;
```

yazın. Həllin başa çatması isə belə yazılacaqdır:

```
if Finish=0 then
begin
  Form1.Caption:='QƏLƏBƏ! ';
  Panel1.Color:=clFuchsia;
  Panel1.Enabled:=False;
end;
```

Burada, formanın sərlövhəsi və panelin rəngi dəyişdirilir, panelin qoşulması qadağan edilir.

`BitBtn1Click` prosedurunu bütün düymələr üçün təyin etmək lazımdır. Bunun üçün növbə ilə hər bir düyməni seçərək, Obyektlər inspektorunda `OnClick` hadisəsi qarşısındakı düyməni basmaqla `BitBtn1Click` prosedurunu seçmək lazımdır.

Beləliklə, bu məsələnin tam proqramı belə olacaqdır:

```
unit Unit1;

interface

uses
   Windows,Messages,SysUtils,Classes,Graphics,
   Controls,Forms,Dialogs,StdCtrls,Buttons,ExtCtrls;

type
   TForm1 = class(TForm)
     Panel1: TPanel;
     BitBtn1: TBitBtn;
     BitBtn2: TBitBtn;
     BitBtn3: TBitBtn;
     BitBtn4: TBitBtn;
     BitBtn5: TBitBtn;
     BitBtn6: TBitBtn;
     procedure BitBtn1Click(Sender: TObject);

  private
    { Private declarations }

  public
    { Public declarations }

  end;

var
   Form1: TForm1;
   n:Integer=4;
   Finish:Integer=24;

implementation

{$R *.DFM}

procedure TForm1.BitBtn1Click(Sender:TObject);
Var
   i,c,k,ak:Integer;
begin
  With Sender as TBitBtn do
    begin
      i:=Tag div 2;
      c:=Tag mod 2;
```

```
      k:=n-i;
      ak:=abs(k);
      if ak<3 then
      if((c=0)and(k>0))
      or((c=1)and(k<0)) then
      begin
        Tag:=Tag+2*k;
        Left:=Left+56*k;
        Finish:=Finish-ak;
        n:=i;
      end;
      if Finish=0 then
      begin
        Form1.Caption:=' QƏLƏBƏ! ';
        Panel1.Color:=clFuchsia;
        Panel1.Enabled:=False;
      end;
    end;
end;
end.
```

F9 klavişini basıb layihəni işə salın və məsələni həll edin.

12.7.3. Cəld müdaxilə düyməsi

Cəld müdaxilə düyməsi Delphi–də **`SpeedButton`** komponenti ilə təsvir olunur. Görünüşü və funksional imkanlarına görə bu düymə şəkilli düyməyə çox oxşayır. Lakin, ondan fərqli olaraq, bu düymə `TGraphicControl` sinfindən əmələ gəlmişdir və pəncərəsiz idarəetmə elementidir. Ona görə də bu düymə fokus ala bilmir və adətən alətlər paneli yaratmaq üçün istifadə olunur. O biri düymələrdən fərqli olaraq, `SpeedButton` düyməsi dəyişdirici kimi də istifadə oluna bilər. Ona görə bu düymə adi və basılmış vəziyyətlərdən başqa üçüncü – çökdürülmüş və ya seçilmiş vəziyyətdə də ola bilər. Düymənin *seçilməsi* `Boolean` tipli `Down` xassəsi ilə müəyyən olunur. Əgər onun qiyməti *True* olarsa, düymə seçilmiş olur, *False* olduqda isə seçilmir.

Cəld müdaxilə düymələri qruplaşdırıla bilər və hər bir düymə müəyyən qrupa mənsub ola bilər. Qruplaşdırılmış düymələr avtomatik olaraq öz təsirlərini razılaşdırırlar, yəni bir düymənin seçilməsi o birinin seçilməsini ləğv edir. Düymənin *qrupa mənsub olması* `Integer` tipli `GroupIndex` xassəsi ilə müəyyənləşdirilir.

`AllowAllUp` xassəsi mausun klavişini təkrar basdıqda *seçilmiş* düymənin *seçilməmiş* vəziyyətə qaytarılmasını müəyyənləşdirir. Əgər bu xassənin qiyməti *True* olarsa, seçmə ləğv edilir, *False* olduqda isə seçmə qrupa daxil olan başqa düymənin seçilməsi ilə ləğv edilir. Susmaya görə `AllowAllUp` xassəsinin qiyməti *False* olur.

Əgər düymə qrupa daxil deyilsə, yəni `GroupIndex=0` olarsa, onda həmin düymə dəyişdirici kimi işləyə bilməz və seçilmiş vəziyyətdədir. Düymənin sərbəst işləməsi üçün bir düymədən ibarət qrup yaradılır. Onda bu düymə üçün `AllowAllUp` xassəsinə *True* qiyməti, `GroupIndex` xassəsinə isə unikal nömrə mənimsədilir.

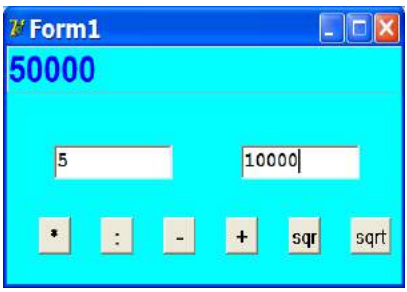
`SpeedButton` düyməsinin səthində üç yox, dörd ayrı–ayrı təsvir ola bilər. Ona görə də bu düymə üçün `NumGlyph` xassəsinin maksimal qiyməti *4*–ə bərabərdir.

Misal. Kalkulyator nümunəsinin hazırlanması.

Biz yalnız vurma əməliyyatı yerinə yetirən kalkulyator nümunəsinin necə hazırlanması prinsipini artıq bilirik. İndi isə bir neçə hesab əməllərini yerinə yetirən kalkulyator nümunəsinə baxaq. Əlbəttə, bu kalkulyator da tam mükəmməl kalkulyator olmayacaq, lakin, gələcəkdə sizin müstəqil olaraq belə kalkulyatoru yarada bilməyiniz üçün əsas ola bilər. Forma üzərinə iki `Edit`, altı `SpeedButton` və bir `Panel` komponentləri yerləşdirin (şəkil 12.10).

`Panel1` komponentini seçərək onun `Aligment` xassəsinə `taLeftJustify`, `Align` xassəsinə `alTop` qiyməti verin və sərlövhəsini pozun. `Edit1` və `Edit2` komponentlərinin `Text` xassəsini pozun. Düymələrin sərlövhəsini şəkildəki kimi dəyişin.

Əvvəlki misalı tam təfsilatı ilə izah etdiyimizdən, burada əlavə izahata ehtiyac görməyərək, məsələnin hazır modulunu Sizə təqdim edirik.



Şəkil 12.10. Kalkulyator

```
unit Unit1;

interface

uses
   Windows,Messages,SysUtils,Classes,Graphics,
   Controls,Forms,Dialogs,Buttons,StdCtrls,ExtCtrls;


type
   TForm1 = class(TForm)
     Panel1: TPanel;
     Edit1: TEdit;
     Edit2: TEdit;
     SpeedButton1: TSpeedButton;
     SpeedButton2: TSpeedButton;
     SpeedButton3: TSpeedButton;
     SpeedButton4: TSpeedButton;
     SpeedButton5: TSpeedButton;
     SpeedButton6: TSpeedButton;
     procedure SpeedButton4Click(Sender: TObject);
     procedure Edit1KeyPress(Sender: TObject;
```

```
                              var Key: Char);
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton5Click(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
    procedure SpeedButton6Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);

  private
    { Private declarations }

  public
    { Public declarations }

  end;

var
   Form1: TForm1;
   y,z: Double;

implementation

{$R *.DFM}

procedure TForm1.SpeedButton4Click(Sender: TObject);
begin
  y:=StrToFloat(Edit1.Text);
  z:=StrToFloat(Edit2.Text);

  With Panel1.Font do
    begin
      Name:='Arial';
      Size:=24;
      Style:=[fsBold];
      Color:=clRed;
    end;

  Panel1.Caption:=FloatToStr(y*z);
  Panel1.Color:=clLime;
end;


procedure TForm1.Edit1KeyPress(Sender:TObject;
                               var Key: Char);
begin
if not(Key in ['0'..'9',DecimalSeparator,
       chr(VK_BACK)]) then
Key:= #0;
end;

procedure TForm1.SpeedButton1Click(Sender: TObject);
```

```
begin
  y:=StrToFloat(Edit1.Text);
  z:=StrToFloat(Edit2.Text);
  With Panel1.Font do
    begin
      Name:='Arial';
      Size:=24;
      Style:=[fsBold];
      Color:=clRed;
    end;
  Panel1.Caption:=FloatToStr(y/z);
  Panel1.Color:=clLime;
end;


procedure TForm1.SpeedButton5Click(Sender: TObject);
begin
  y:=StrToFloat(Edit1.Text);
  z:=StrToFloat(Edit2.Text);
  With Panel1.Font do
  begin
    Name:='Arial';
    Size:=24;
    Style:=[fsBold];
    Color:=clRed;
  end;
  Panel1.Caption:=FloatToStr(y-z);
  Panel1.Color:=clLime;
end;


procedure TForm1.SpeedButton3Click(Sender: TObject);
begin
  y:=StrToFloat(Edit1.Text);
  z:=StrToFloat(Edit2.Text);
  With Panel1.Font do
    begin
      Name:='Arial';
      Size:=24;
      Style:=[fsBold];
      Color:=clRed;
    end;
  Panel1.Caption:=FloatToStr(y+z);
  Panel1.Color:=clLime;
end;


procedure TForm1.SpeedButton6Click(Sender: TObject);
begin
  y:=StrToFloat(Edit1.Text);
```

```
  Edit2.Enabled:=False;
  With Panel1.Font do
    begin
      Name:='Arial';
      Size:=24;
      Style:=[fsBold];
      Color:=clRed;
    end;

  Panel1.Caption:=FloatToStr(y*y);
  Panel1.Color:=clLime;
end;


procedure TForm1.SpeedButton2Click(Sender: TObject);
begin
  y:=StrToFloat(Edit1.Text);
  Edit2.Enabled:=False;
  With Panel1.Font do
    begin
      Name:='Arial';
      Size:=24;
      Style:=[fsBold];
      Color:=clRed;
    end;
  Panel1.Caption:=FloatToStr(Sqrt(y));
  Panel1.Color:=clLime;
end;

end.
```

12.8. Dəyişdiricilər

Dəyişdiricilərin köməyi ilə istifadəçi lazım olan parametrləri seçmək imkanı əldə edir. Dəyişdiricilərə demək olar ki, Windows–un bütün pəncərələrində rast gəlmək mümkündür. Dəyişdiricilər iki növ olur: müstəqil qeyd olunmuş və asılı qeyd olunmuş. *Müstəqil qeyd olunmuş* dəyişdiriciyə sadəcə olaraq bayraq da deyirlər. Bayraqlar iki vəziyyətdə – qoşulmuş və qoşulmamış vəziyyətlərdə olur. *Asılı qeyd olunmuş* dəyişdiricilərə isə sadəcə olaraq dəyişdiricilər deyirlər. Onlar da həmin iki vəziyyətdə olur, lakin bayraqlar təklikdə işlədikləri halda, dəyişdiricilər tək işləyə bilmir. Dəyişdiricilərdən biri həmişə qoşulmuş vəziyyətdə olur. Bu halda digər dəyişdiricilər qoşula bilmir.

Dəyişdiricilərlə işləmək üçün Delphi `CheckBox`, `RadioButton` və `RadioGroup` komponentləri təklif edir. `CheckBox` və `RadioButton` dəyişdiriciləri `Button` düyməsinin əmələ gəldiyi `TButtonControl` sinfindən yaranmışdır.

12.8.1. Müstəqil qeyd olunmuş dəyişdirici

Bu dəyişdirici **`CheckBox`** komponenti ilə yaradılır. Dəyişdirici sərlövhədən ibarət düzbucaqlı şəklindədir. Düzbucaqlı daxilində mausun sol düyməsini basdıqda ✓ işarəsi əmələ gəlir. Bu halda dəyişdirici qoşulmuş hesab olunur və deyirlər ki, "*bayraq*" qoyulmuşdur. Düzbucaqlı boş olduqda deyirlər ki, bayraq atılmışdır, yəni istifadəçi həmin parametrdən imtina edir.

Bayrağın *vəziyyətini* **`Checked`** xassəsi müəyyən edir. Susmaya görə onun qiyməti *False*–dır, yəni bayraq atılmışdır.

İstifadəçi bayrağın vəziyyətini mausla dəyişdirə bilər. Belə ki, əgər bayraq atılmışdırsa, mausun düyməsini basdıqda bayraq qoyulur və əksinə, bayraq qoyulmuşdursa, mausun düyməsini basdıqda bayraq atılır. Buna müvafiq olaraq `Checked` xassəsinin qiyməti də dəyişir. Əgər `CheckBox` komponenti fokus almış vəziyyətdə olarsa, onda bayrağı *probel* klavişini basmaqla da qoymaq və ya atmaq olar. `Checked` xassəsinə kod vasitəsilə də qiymət vermək olar:

```
CheckBox1.Checked:=true;
CheckBox2.Checked:=false;
```

Əgər `Enabled` xassəsinə *False* qiyməti verilərsə, onda bayrağın dəyişdirilməsi mümkün olmur, məsələn, `CheckBox1.Enabled:=false;`

Müstəqil qeyd olunmuş dəyişdiricinin üçüncü vəziyyəti imtina olunmuş vəziyyətdir. Bu vəziyyəti `AllowGrayed` xassəsi idarə edir. Əgər bu xassənin qiyməti *True* olarsa, mausun klavişini basdıqda bayraq üç vəziyyət arasında dövrü dəyişir: qoşulur, qoşulmur və imtina olunur. İmtina olunmuş vəziyyətdə düzbucaqlı daxilində ✓ işarəsi olmasına baxmayaraq dəyişdirici boz rəngli olur.

Bayrağın üç vəziyyətindən birini seçmək və onu təhlil etmək üçün, `TCheckBoxState` tipli `State` xassəsindən istifadə olunur. Bu xassə aşağıdakı qiymətləri ala bilər:

`cbUnChecked` –*bayraq atılmışdır*;
`cbChecked` –*bayraq qoyulmuşdur*;
`cbGrayed` –*bayraq qadağan olunmuşdur*.

Dəyişdiricinin hansı vəziyyətə keçməsindən asılı olmayaraq, onun vəziyyətini dəyişdikdə `OnClick` hadisəsi baş verir.

Misal. Vurma cədvəlinin tərtib edilməsi.

Vurma cədvəlini proqramlaşdırmaq çox asan məsələdir. Lakin, biz adi vurma cədvəli tərtib etməyəcəyik. Biz vuruqları klaviaturadan deyil, şkala adlanan idarəedici elementdən daxil edəcəyik.

Qiymətlər diapazonu ilə işləmək üçün Delphi *şkala* adlanan **`TrackBar`** komponentini təklif edir ki, onun köməyi ilə qiymətlər diapazonundan tam ədədləri seçmək mümkündür. Bu komponent də Windows sistemində geniş istifadə olunur. Buna ən sadə misal olaraq audio–qurğuların səs gücləndirici şkalasını göstərmək olar. Bu komponentin xassələrini məsələnin həlli prosesində öyrənəcəyik.

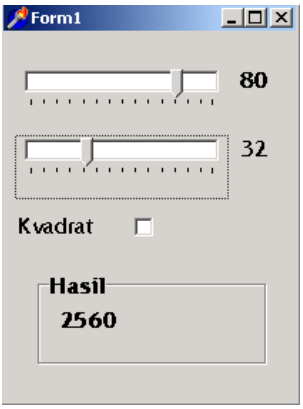
Vurma cədvəlində iki vuruq olduğuna görə, bizə iki şkala komponenti lazım olacaqdır. Ona görə də forma üzərinə **Win32** səhifəsindən iki `TrackBar`, **Standart** səhifəsindən isə üç `Label`, bir `CheckBox` və bir `GroupBox` komponenti yerləşdirin (şəkil 12.11).

Şkala üzərində hərəkət edən (maus və ya idarəetmə klavişləri ilə) məkiyin mövqeyi vuruqların qiymətini müəyyən edir. Bu qiymətləri şkalaların sağ tərəfində yerləşdirilmiş yazı komponentləri üzərində təsvir etdirəcəyik. Hər iki şkala tamamilə eyni işləməlidir. Ona görə də hər iki şkalanın xassələrinə eyni qiymətlər verəcəyik. Şkalaları növbə ilə seçərək Obyektlər inspektorunda aşağıdakı xassələrin qiymətlərini müəyyənləşdirin.

`Orientation` xassəsi – şkalanın *üfqi* və ya *şaquli* vəziyyətdə olmasını müəyyən edir. Bu xassəyə `ttHorizontal` qiyməti verin.

`Min` (*Minimum*) xassəsi – şkalanın *minimal* qiymətini müəyyən edir. Bu xassəyə *2* qiyməti daxil edin.

`Max` (*Maksimum*) xassəsi – şkalanın *maksimal* qiymətini müəyyən edir. Bu xassəyə *99* qiyməti daxil edin. Bu halda ikirəqəmli ədədlərin hasili hesablanacaq. Çoxrəqəmli ədədlər üçün bu xassənin qiymətini dəyişmək lazımdır (*999*, *9999* və s.).



Şəkil 12.11. Vurma cədvəli

`Position` (*Mövqe*) xassəsi – məkiyin *cari mövqeyini* bildirir. Məkiyi hərəkət etdirdikdə onun qiyməti avtomatik olaraq dəyişir. `Position` xassəsi ilə məkiyin başlanğıc vəziyyətini müəyyən etmək olar. Bu qiymət `Min` və `Max` diapazonunda olmalıdır. Başlanğıc anda məkiyin sol kənarda yerləşməsi üçün bu xassəyə də *2* qiyməti daxil edin.

`LineSize` (*Dəyişmə addımı*) xassəsi – məkiyi idarəetmə klavişləri ilə (sağa, sola, aşağı və yuxarı) hərəkət etdirdikdə *dəyişmə addımını* müəyyən edir. Bu xassəyə *1*, yəni minimal qiymət daxil edin.

`PageSize` (*Dəyişmə addımı*) xassəsi – məkiyi *PageUp* və *PageDown* klavişləri ilə hərəkət etdirdikdə *dəyişmə addımını* müəyyən edir. Bu xassəyə ixtiyari, məsələn, *7* qiyməti verin.

`Frequency` (*Şkala tezliyi*) xassəsi – şkalada bölgülərin *yerləşmə sıxlığını* müəyyən edir. Bu xassəyə də *7* qiyməti verin. Bu zaman məkik bir bölgüdən digər bölgüyə atılacaqdır.

İndi isə **GroupBox** qrup komponentini seçin. Bu komponent də yenidir və funksional əlaqəli idarəedici *elementləri əyani təsvir etmək* üçündür (konteynerdir). `GroupBox` komponenti düzbucaqlı haşiyədən və onun sol yuxarı küncündə yerləşən sərlövhədən ibarətdir. Bu komponentin `Caption` xassəsini `Hasil` adlandırın. Onun üzərindəki `Label3` komponentinin `Aligment` xassəsinə `taLeftJustify` qiyməti verin.

Yazı komponentlərinin (`Label`) hamısının `AutoSize` xassəsinə *False* qiyməti verin. `Label1` və `Label2` komponentlərini şkalalar qarşısında yerləşdirib `Caption` xassəsinə *2* qiyməti, `Label3` komponentini isə qrup üzərində yerləşdirib, `Caption` xassəsinə *4* qiyməti daxil edin (başlanğıc qiymətlərə uyğun olaraq). Hər üç yazı komponentinin `Aligment` xassəsinə `taRightJustify` qiyməti seçərək, onları sağ tərəfə düzləndirin.

`CheckBox1` bayraq komponentini şkaladan aşağıda yerləşdirərək onun `Caption` xassəsinə `Kvadrat` sözü yazın, `Aligment` xassəsi üçün isə `taLeftJustify` qiymətini seçin. `CheckBox` komponenti ədədi kvadrata yüksəltmək üçün nəzərdə tutulmuşdur.

İndi isə proqramlaşdırma ilə məşğul olaq. Şkalalar iki rejimdə işləməlidir: bayraq atıldıqda onlar bir–birindən asılı olmayaraq işləyir, bayraq qoyulduqda isə hər iki şkalada eyni qiymət olmalıdır. Şkalada qiyməti dəyişdikdə `OnChange` hadisəsi baş verir. Ona görə də `TrackBar1` komponentini seçib, Obyektlər inspektorunda bu hadisənin qarşısında mausun düyməsini iki dəfə basaraq, modula bu hadisə emaledicisini əlavə edin:

```
procedure TForm1.TrackBar1Change(Sender: TObject);
begin
  Label1.Caption:= IntToStr(TrackBar1.Position);
  Label3.Caption:= IntToStr(TrackBar1.Position*
                            TrackBar2.Position);
end;
```

Yəni, `Label1` yazısında məkiyin mövqeyinə uyğun qiymət, `Label3` yazısında isə hasil – nəticə təsvir olunur. Analoji proseduru `TrackBar2` komponenti üçün də yaradın:

```
procedure TForm1.TrackBar2Change(Sender: TObject);
begin
  Label2.Caption:= IntToStr(TrackBar2.Position);
  Label3.Caption:= IntToStr(TrackBar1.Position*
                            TrackBar2.Position);
end;
```

Xatırladaq ki, modulda təkrarlanan sətirləri mübadilə buferindən istifadə edərək daxil etmək daha asan olar.

F9 klavişini basaraq məsələnin birinci mərhələsini – iki ədədin hasilinin hesablanmasını yoxlayın.

İndi isə bayrağın qoyulması halı üçün proqram tərtib edək. Bunun üçün bayraq üzərində mausun düyməsinin basılması `OnClick` hadisə emaledicisi yaradılmalı və bu prosedurda cəmi bir sətir yazılmalıdır:

```
TrackBar2.Position:=TrackBar1.Position;
```

Bu halda hər iki şkalada məkiyin mövqeyi və uyğun olaraq vuruqlar eyni olmalıdır. “Əgər bayraq qoyulmuşdursa”, məntiqi

```
if CheckBox1.Checked then
```

kimi yazılır. “Bu halda vuruqlar eyni olmalıdır” məntiqi isə

```
TrackBar2.Position:=TrackBar1.Position;
```

və

```
TrackBar1.Position:=TrackBar2.Position;
```

kodları ilə təsvir olunur.

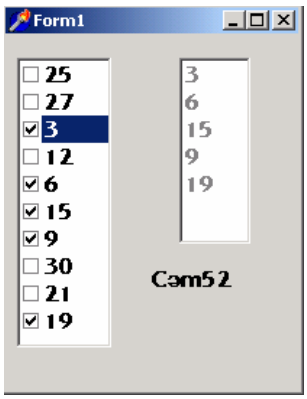
Bu kodları sadəcə olaraq hər iki prosedura əlavə etmək lazımdır. Onda proqramın tam mətni belə olacaqdır:

```
unit Unit1;

interface

uses
   Windows,Messages,SysUtils,Classes,Graphics,
   Controls,Forms,Dialogs,StdCtrls,ComCtrls;

type
   TForm1 = class(TForm)
     TrackBar1: TTrackBar;
     TrackBar2: TTrackBar;
     CheckBox1: TCheckBox;
     GroupBox1: TGroupBox;
     Label1: TLabel;
     Label2: TLabel;
     Label3: TLabel;
     procedure TrackBar1Change(Sender: TObject);
     procedure TrackBar2Change(Sender: TObject);
     procedure CheckBox1Click(Sender: TObject);

  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
   Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.TrackBar1Change(Sender: TObject);
begin

  Label1.Caption:= IntToStr(TrackBar1.Position);
  Label3.Caption:= IntToStr(TrackBar1.Position*
                            TrackBar2.Position);
```

```
  if CheckBox1.Checked then
     TrackBar2.Position:=TrackBar1.Position;
end;


procedure TForm1.TrackBar2Change(Sender: TObject);
begin
  Label2.Caption:= IntToStr(TrackBar2.Position);
  Label3.Caption:= IntToStr(TrackBar1.Position*
                            TrackBar2.Position);
  if CheckBox1.Checked then
     TrackBar1.Position:=TrackBar2.Position
end;


procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  TrackBar2.Position:=TrackBar1.Position;
end;

end.
```

Misal. Loydun başsındıran məsələsi.

Biz bu misalda bir sıra başsındıran və şahmat məsələləri müəllifi kimi tanınan amerikalı riyaziyyatçı Samuel Loydun ilk baxışda çox sadə görünən belə bir məsələsini həll edəcəyik: Loydun təklif etdiyi **25**, **27**, **3**, **12**, **6**, **15**, **9**, **30**, **21**, **19** ədədlərindən elə ədədlər seçin ki, onların cəmi **50** olsun.



Şəkil 12.12. Loydun başsındıran məsələsi

Bu məsələni həll etmək üçün bizə əsasən iki siyahı lazım olacaqdır. Birinci siyahı bayraqlar siyahısı olacaqdır ki, bu siyahıdan bayraq qoymaqla axtarılan ədədlər seçiləcək və seçilmiş ədədlər ikinci adi siyahıda yerləşdiriləcəkdir. Seçilmiş ədədlərin cəmi `Label` yazısı üzərində əks olunacaqdır. Yəqin özünüz başa düşdünüz ki, bu siyahılardan biri **`Additional`** səhifəsindən **`CheckListBox`**, digəri isə **`Standart`** səhifəsindən `ListBox` komponentlərinə uyğun gəlir. Beləliklə, bu komponentləri şəkil 12.12 – də göstərilən qaydada forma üzərində yerləşdirin.

`CheckListBox` komponentini seçərək Obyektlər inspektorunda `Items` xassəsi qarşısında mausun düyməsini iki dəfə basmaqla, *String List Editor* (*Sətirlər siyahısı redaktoru*) pəncərəsini açıb, Loydun təklif etdiyi ədədləri daxil edin. Ədədlərin hər birinin bir sətirdə yerləşməsi üçün hər dəfə *Enter* klavişini basın. *Ok* düyməsini basdıqdan sonra, `CheckListBox` komponentini dartaraq onun ölçüsünü elə tənzimləyin ki, bütün ədədlər görünsün. `Label1`

komponenti üçün `Caption` – Cəm 0, `AutoSize` – *False*, `Aligment` – `taCenter` qiymətləri təyin edin.

Bütün proqramlaşdırma işləri yalnız `CheckListBox` komponentinə aid olacaqdır. Bu siyahıda bayraq qoyulduqda **`OnClickCheck`** hadisəsi baş verir. Bu hadisə üçün prosedur yaradaq:

```
unit Unit1;

interface

uses
   Windows,Messages,SysUtils,Classes,Graphics,
   Controls,Forms,Dialogs,StdCtrls,CheckLst;

type
   TForm1 = class(TForm)
      CheckListBox1: TCheckListBox;
      ListBox1: TListBox;
      Label1: TLabel;
      procedure CheckListBox1ClickCheck(
                        Sender: TObject);

   private
     { Private declarations }

   public
     { Public declarations }

   end;

var
   Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.CheckListBox1ClickCheck(
                          Sender: TObject);
Var
   i,s:Integer;
begin
  ListBox1.Clear;
  for i:=0 to CheckListBox1.Items.Count-1 do
  if CheckListBox1.Checked[i] then
  ListBox1.Items.Add(CheckListBox1.Items[i]);
  s:=0;
  for i:=0 to ListBox1.Items.Count-1 do
  s:=s+StrToInt(ListBox1.Items[i]);
```

```
    Label1.Caption:='Cəm '+IntToStr(s);
    if s=50 then
    Label1.Caption:='QƏLƏBƏ '+#13#10+IntToStr(s);
    if s>=50 then
    begin
      CheckListBox1.Enabled:=False;
      ListBox1.Enabled:=False;
    end;
  end;

  end.
```

Burada, birinci `for` operatoru ilə siyahının bütün sətirləri əhatə olunmaqla, bayraq qoyulmuş sətirlər (`Checked`) seçilir (`if`) və sadə siyahıya əlavə edilir (`Add`). İkinci `for` operatoru ilə seçilmiş ədədlər cəmlənir və bu cəm *50*–yə bərabər olduqda oyunçu qalib gəlir və hər iki siyahı qapanır.

12.8.2. Asılı qeyd olunmuş dəyişdirici

Bu dəyişdirici **`RadioButton`** komponenti ilə təsvir olunur, onlara seçim düymələri də deyirlər. Bu komponentlər sərlövhədən ibarət dairə şəklində təsvir olunur. Bayraqlardan fərqli olaraq, bu dəyişdiricilərdən heç vaxt tək–tək istifadə olunmur, onlar həmişə qrup şəklində fəaliyyət göstərir. Qrupda bir dəyişdirici həmişə seçilmiş vəziyyətdə olur (dairə daxilində qalın nöqtə işarəsi qoyulmaqla). Müstəqil dəyişdiricilərdən fərqli olaraq, seçilmiş asılı dəyişdiricinin vəziyyətini mausun düyməsini onun daxilində təkrarən basmaqla dəyişdirmək mümkün deyildir. Bunun üçün mausun düyməsini qrupda olan digər dəyişdiricilərin ixtiyari biri üzərində basmaq lazımdır. `RadioButton` komponenti üçün `OnClick` hadisəsi yalnız dəyişdirici seçildikdə baş verir. Bu komponentlər üzərində mausun düyməsini təkrarən basdıqda `OnClick` hadisəsi baş vermir. Dəyişdiricilərin yerləşdikləri qrupları konteyner adlandırırlar. Konteyner kimi adətən `Form` forması, `Panel` paneli və `GroupBox` qrupu istifadə olunur. Bu konteynerlərdən başqa, Delphi `RadioButton` dəyişdiricilərindən ibarət xüsusi **`RadioGroup`** komponenti təqdim edir. Dəyişdiriciləri əl ilə yerləşdirilən qruplardan fərqli olaraq, bu qrup, dəyişdiricilərin nizamla düzülməsi və qarşılıqlı əlaqələndirilməsi üçün yaradılmışdır. `RadioGroup` qrupunda digər elementlər də, məsələn, `CheckBox` müstəqil dəyişdiricisi və ya `Edit` mətn redaktoru ola bilər.

Dəyişdiricilərin sayı və adları `TStrings` tipli `Items` xassəsi ilə idarə olunur. Bu xassə qrupun ayrı–ayrı dəyişdiricilərinə müraciət etməyə imkan verir. `Items` xassəsi dəyişdiricilərin sərlövhəsi kimi təsvir olunan sətirlərdən ibarətdir. Massivdə sətirlərin nömrəsi sıfırdan başlayır: `Items[0]`, `Items[1]` və s.

Ayrı–ayrı dəyişdiricilərə müdaxilə `Integer` tipli `ItemIndex` xassəsi ilə yerinə yetirilir. Bu xassə qrupda hansı dəyişdiricinin seçildiyini (qoşulduğunu)

müəyyən etmək üçün istifadə oluna bilər. Susmaya görə, onun qiyməti (*–1*)–ə bərabərdir, yəni qrupda heç bir dəyişdirici seçilməmişdir.

Qrupda dəyişdiricilərin neçə sütunda yerləşməsini `Integer` tipli `Columns` xassəsi müəyyənləşdirir. Bu xassə yalnız dəyişdiricilərə aiddir və qrupda olan digər idarəetmə elementlərinə, məsələn, `Edit`, `Label` komponentlərinə aid deyil.

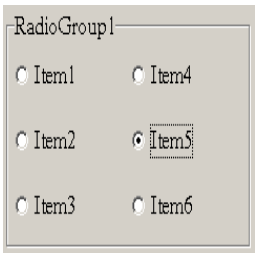
Qrupun sətirləri (dəyişdiricilərin sərlövhələri) üzərində `Add` və `Delete` metodlarını tətbiq etməklə əməliyyatlar aparmaq olar.

12.8.2.1. Asılı qeyd olunmuş dəyişdiricilərə aid misallar

Misal. Proqram yolu ilə dəyişdiricilərin yaradılması.

Forma üzərinə **`Standart`** səhifəsindən yalnız bir komponent – `RadioGroup` komponenti yerləşdirin və forma üçün `OnCreate` prosedurunu yaradın:

```
procedure
TForm1.FormCreate(Sender:TObject);

begin
  RadioGroup1.Items.Clear;
  RadioGroup1.Items.Add('Item1');
  RadioGroup1.Items.Add('Item2');
  RadioGroup1.Items.Add('Item3');
  RadioGroup1.Items.Add('Item4');
  RadioGroup1.Items.Add('Item5');
  RadioGroup1.Items.Add('Item6');
  RadioGroup1.Columns:=2;
  RadioGroup1.ItemIndex:=4;
end;
```



Şəkil.12.13. Dəyişdiricilərin qrupda yerləşdirilməsi

F9 klavişini basdıqdan sonra, forma üzərində iki sütunda *6* dəyişdirici yerləşdiriləcək (şəkil 12.13) və beşinci dəyişdirici seçiləcəkdir.

Misal. `RadioButton`, `Edit` və `ComboBox` komponentlərinin qarşılıqlı təsiri.

Forma üzərinə bir `ComboBox`, bir `Edit`, iki `Button` və üç `RadioButton` komponentləri yerləşdirib, onların sərlövhələrini şəkil 12.14 – də olduğu kimi dəyişdirin.

`Adi üslub` sərlövhəli dəyişdirici üzərində mausun düyməsini basaraq, `OnClick` hadisə emaledici prosedurunu yaradın:

```
procedure TForm1.RadioButton1Click(Sender: TObject);

begin
  ComboBox1.Style:=csDropDown;
end;
```

Bu halda istifadəçi kombinasiyalı siyahıdan elementi seçə və ya daxil edə bilər.

`Qeyd olunmuş` üslub dəyişdiricisi üçün də belə prosedur yaradın:

```
procedure TForm1.RadioButton2Click(Sender: TObject);
begin
  ComboBox1.Style:=csOwnerDrawFixed;
end;
```

Bu halda istifadəçi siyahıdan elementi yalnız seçə bilər (daxil edə bilməz).

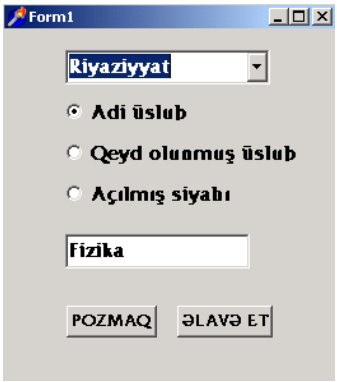
`Açılmış siyahı` sərlövhəli dəyişdirici üçün isə belə prosedur yazın:

```
procedure TForm1.RadioButton3Click(Sender: TObject);
begin
  ComboBox1.Style:=csSimple;
end;
```

Bu halda isə `ComboBox` komponenti `Edit` komponenti kimi görünür və təsir edir (əgər siyahı tam dolmuşsa, onu *PageUp* və *PageDown* klavişləri ilə idarə etmək lazımdır).

İndi isə `Pozmaq` sərlövhəli `Button1` düyməsi üzərində mausun düyməsini iki dəfə basaraq yunitə bu proseduru əlavə edin:

```
procedure TForm1.Button1Click(
           Sender:Object);

begin
  ComboBox1.Clear;
  Edit1.Clear;
end;
```



Şəkil 12.14. İdarəedici elementlərin qarşılıqlı təsiri

Hətta əlavə izaha ehtiyac qalmır.

`Əlavə et` sərlövhəli `Button2` düyməsi üçün `OnClick` hadisə emaledicisini də yunitə əlavə edin:

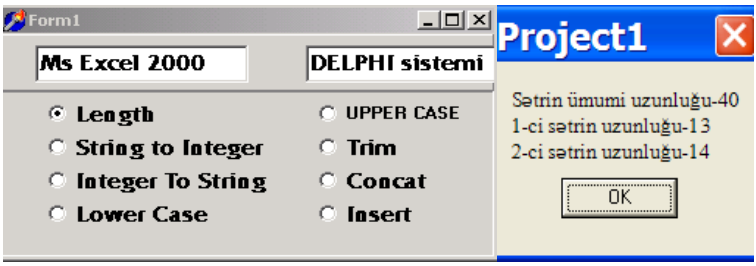
```
procedure TForm1.Button2Click(Sender: TObject);

begin
  if Edit1.Text='' then
  Exit;
  ComboBox1.Items.Add(Edit1.Text);
end;
```

Əgər mətn sahəsi boş olarsa, blokun işi yarımçıq dayandırılır (`Exit`), əks halda bu sahədən daxil edilən mətn kombinasiyalı siyahıya daxil edilir.

F9 klavişini basaraq layihədə bütün bu halların hər birini icra edin.

Misal. Sətirlər üzərində əməliyyatlar.

Forma üzərinə iki `Panel` komponenti yerləşdirin. `Panel1` üzərində iki `Edit` komponenti yerləşdirərək (şəkil 12.15) panelin `Align` xassəsinə `alTop` qiyməti verin. Hər iki mətn sahəsi panellə birlikdə formanın yuxarı hissəsində yerləşəcəkdir. `Panel2` komponenti üzərində isə *8* `RadioButton` düyməsi yerləşdirin. Panelin `Align` xassəsinə `alClient` qiyməti verin. Bu halda panel formanın bütün yerdə qalan hissəsini tutacaqdır. Bu paneli o birindən



Şəkil 12.15. Sətirlər üzərində əməliyyatlar

fərqləndirmək üçün, onun `BevelInner` xassəsinə – `bvNone`, `BevelOuter` xassəsinə – `bvLowered`, `BorderWidth` xassəsinə *5*, `BevelWitdh` xassəsinə isə *0* qiyməti verin (bu xassələrlə az sonra tanış olacağıq). Dəyişdiricilərin sərlövhələrini isə sol və sağ sütunlar üzrə yuxarıdan aşağıya prinsipi ilə dəyişdirin, yəni `RadioButton1` – `Length`, …, `RadioButton8` – `Insert` adlandırın.

Burada biz hər bir proseduru ayrılıqda izah etməyəcəyik. Belə izahatları müvafiq sətirlər qarşısında şərhlərin köməyi ilə verəcəyik. Lakin, bu proqramı başa düşmək üçün kitabın sonundakı *Əlavəni* öyrənmək lazımdır.

Beləliklə, bu məsələnin modulunun tam mətni belə olacaqdır:

```
unit Unit1;

interface

uses
   Windows,Messages,SysUtils,Classes,Graphics,
   Controls,Forms,Dialogs,StdCtrls,ExtCtrls;

type
   TForm1 = class(TForm)
     Panel1: TPanel;
     Edit1: TEdit;
     Edit2: TEdit;
     Panel2: TPanel;
     RadioButton1: TRadioButton;
     RadioButton2: TRadioButton;
     RadioButton3: TRadioButton;
```

```
    RadioButton4: TRadioButton;
    RadioButton5: TRadioButton;
    RadioButton6: TRadioButton;
    RadioButton7: TRadioButton;
    RadioButton8: TRadioButton;
    procedure RadioButton1Click(Sender: TObject);
    procedure RadioButton2Click(Sender: TObject);
    procedure RadioButton3Click(Sender: TObject);
    procedure RadioButton4Click(Sender: TObject);
    procedure RadioButton5Click(Sender: TObject);
    procedure RadioButton6Click(Sender: TObject);
    procedure RadioButton7Click(Sender: TObject);
    procedure RadioButton8Click(Sender: TObject);

  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.RadioButton1Click(Sender: TObject);
Var
  x,y:Integer;
  a,b:String;

// Edit sahələrində olan mətnlərin uzunluqlarının tapılması
begin
 a:=Edit1.Text;
 b:=Edit2.Text;
 x:=Length(a); // Edit1 sahəsində olan mətnin uzunluğunun tapılması
 y:=Length(b); // Edit2 sahəsində olan mətnin uzunluğunun tapılması
 ShowMessage(' Sətrin ümumi uzunluğu- '+IntTOStr(x+y)+
 #13#10+' 1-ci sətrin uzunluğu- '+IntToStr(x)+
 #13#10+' 2-ci sətrin uzunluğu- '+IntToStr(y));
end;


procedure TForm1.RadioButton2Click(Sender: TObject);
Var
  i:Integer;s:String;
begin
```

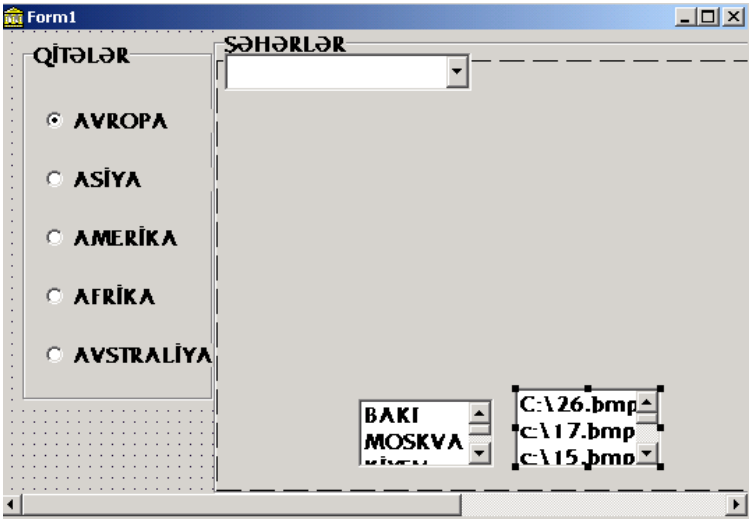
```
  s:='200';           // 200 ədəd deyil, məna kəsb etmir
  i:=StrToInt(s)*2;   // 200 sətri ədədə çevrilərək 2–yə vurulur
  Form1.Left:=i;      // Forma yerini dəyişir
end;


procedure TForm1.RadioButton3Click(Sender: TObject);

begin
  ShowMessage(' Pəncərənin eni- '+IntToStr(Form1.Width)+
  #13#10+' Pəncərənin hündürlüyü- '+
  IntToStr(Form1.Height)); {IntToStr funksiyası tam ədədi
                            sətrə çevirir}
end;


procedure TForm1.RadioButton4Click(Sender: TObject);

begin
  // Redaktorlardan mətni böyük hərflərlə daxil edin
  Edit1.Text:= AnsiLowerCase(Edit1.Text);
  Edit2.Text:= AnsiLowerCase(Edit2.Text);
  // Redaktorlarda mətn kiçik hərfli mətnlə əvəz olunacaq
end;


procedure TForm1.RadioButton5Click(Sender: TObject);
begin
  // Redaktorlardan mətni kiçik hərflərlə daxil edin
  Edit1.Text:= AnsiUpperCase(Edit1.Text);
  Edit2.Text:= AnsiUpperCase(Edit2.Text);
  // Redaktorlarda mətn böyük hərfli mətnlə əvəz olunacaq
end;


procedure TForm1.RadioButton6Click(Sender: TObject);
begin
  // Redaktorlardan mətnin əvvəli və sonuna probellər daxil edin
  Edit1.Text:= Trim(Edit1.Text);
  Edit2.Text:= Trim(Edit2.Text);
  // Mətnin əvvəli və sonundan probellər pozulacaq
end;


procedure TForm1.RadioButton7Click(Sender: TObject);
begin
  ShowMessage(Concat(Edit1.Text,Edit2.Text));
```

```
  ShowMessage(Concat(Edit1.Text+Edit2.Text));
  // Birinci halda Concat funksiyası ilə,
  // 2–ci halda isə toplama ilə mətnlər birləşdirilir
end;


procedure TForm1.RadioButton8Click(Sender: TObject);
Var
   s,y:String;
begin
  // Edit1–dən "Mən oxuyuram"
  // Edit2–dən " 2–ci kursda" mətnləri daxil edin
  s:=Edit1.Text;
  y:=Edit2.Text;
  Insert(y,s,4);
  ShowMessage(s); //"Mən 2–ci kursda oxuyuram" alınacaqdır
end;

end.
```

Misal. Elektron albomun yaradılması.

Bu misalda biz interaktiv elektron albomun yaradılması ilə məşğul olacağıq. İstifadəçi istədiyi şəkli özü seçəcəkdir. Proqramın pəncərəsi iki hissədən ibarət olacaqdır: sol tərəfdə albomun bölmələrindən ibarət dəyişdiricilər, sağ tərəfdə isə şəhərlərin adlarından ibarət açılan siyahı yerləşəcəkdir. Hansı dəyişdiricinin qoşulmasından asılı olaraq siyahının məzmunu dəyişəcəkdir. Siyahıdan aşağıda, seçilmiş şəhərə uyğun fotoşəkil təsvir olunacaqdır. Əgər Sizin kompüterinizdə şəhərlərin şəkilləri yoxdursa, istənilən şəkil və ümumiyyətlə, digər materiallar istifadə edə bilərsiniz. Bu prinsipial əhəmiyyət kəsb etmir.

Dəyişdiriciləri üzərində yerləşdirmək üçün, formanın sol tərəfində **Standart** səhifəsindən, GroupBox haşiyəsi yerləşdirin (şəkil 12.16). Onun sərlövhəsini Qitələr adlandırın. Haşiyə üzərinə RadioButton dəyişdiricisi yerləşdirin. Daha *4* belə dəyişdirici lazımdır. Onları mübadilə buferindən yerləşdirək. Bunun üçün RadioButton1 dəyişdiricisini seçərək *Ctrl+C* klavişlərini, sonra isə dörd dəfə *Ctrl+V* klavişlərini basın. Bu dəyişdiriciləri bir–birinin altında səliqə ilə yerləşdirək. Bu məqsədlə *Shift* klavişini basılı saxlayaraq, hər bir dəyişdirici üzərində mausun klavişini basmaqla bütün dəyişdiriciləri seçin. Bütün dəyişdiricilər seçildikdən sonra, *Edit/Align* (*Düzəliş/Düzləndirmə*) əmrini icra edin. Açılan *Aligment* (*Düzləndirmə*) dialoq pəncərəsində *Left Sides* (*Sol tərəf*) və *SpaceEqually* (*Bərabər məsafələr*) rejimlərini qoşun və *Ok* düyməsini basın. Növbə ilə hər bir dəyişdiricini seçərək onların sərlövhələrini şəklə uyğun olaraq dəyişdirin. RadioButton1 düyməsi üçün Checked xassəsinə *True* qiyməti verin. Bununla da proqram yükləndikdə Avropa dəyişdiricisi qoşulmuş vəziyyətdə olacaqdır.



Şəkil 12.16. Elektron fotoalbom məsələsinin layihəsi

Formada ikinci haşiyəni (`GroupBox`) yerləşdirib, sərlövhəsini `Şəhərlər` adlandırın. Haşiyə üzərində `ComboBox` komponenti yerləşdirin. Onun `Style` (üslub) xassəsinə `csDropDownList` (açılan siyahı) qiyməti seçin. **`Additional`** səhifəsindən **`Image`** komponentini seçərək onu `ComboBox` komponentindən aşağıda yerləşdirin. Bu yeni komponent təsvirləri əks etdirmək üçün tətbiq olunur. Onun əsas xassəsi `Picture` – hazır təsviri `Image` komponenti üzərində yerləşdirməyə imkan verir.

Qitə və şəhərlərin adlarını yadda saxlamaq üçün formanın istənilən hissəsinə `ListBox1` və `ListBox2` komponentləri yerləşdirin. Onların hər ikisinin `Visible` xassəsinə *False* qiyməti verin ki, proqram hazır olduqda onlar pəncərədə görünməsin. `ListBox1` komponenti üçün `Items` xassəsi qarşısında, üç nöqtə təsvirli düymə üzərində mausun düyməsini iki dəfə basaraq açılan redaktorla şəhərlərin adlarını (hər qitə üçün beş şəhər) və eyni qayda ilə `ListBox2` komponenti üçün şəhərlərin şəkillərinin yerləşdiyi faylların adlarını, məsələn, `c:\Albom\baku.bmp` daxil edin.

Bütün bu konstruksiyalaşdırma işlərindən sonra, məsələni proqramlaşdırmağa başlaya bilərik. Burada ən çətin məsələ dəyişdirici üçün `OnClick` hadisəsi baş verdikdə (dəyişdiricinin daxilində mausun düyməsini və ya *Enter* və *Probel* klavişlərini basdıqda) prosedura ötürülən `Sender` parametrinin hansı obyektə aid olduğunu müəyyən etməkdir. Digər tərəfdən, açılan siyahını lazım olan şəhərlərin adları ilə doldurmaq, siyahıda birinci bəndi seçmək və nəhayət lazım olan şəkli seçmək həll olunacaq əsas məsələlərdəndir.

Əlbəttə, prosedura ötürülən `Sender` obyekti dəyişdiricidir (`RadioButton`). Belə dəyişdiricilər beş ədəd olduğu üçün məhz hansı dəyişdiricidən söhbət getdiyini müəyyənləşdirmək lazımdır. Bunun üçün yenə də bütün komponentlərdə mövcud olan, tam müstəqil `Tag` xassəsi köməyimizə

gəlir. `Tag` xassəsindən istifadə etmək üçün şəhərləri nömrələyək. Birinci dəyişdiriciyə *0*–dan *4*–ə, ikinci dəyişdiriciyə *5*–dən *9*–a və s., sonuncu dəyişdiriciyə isə *20*–dən *24*–ə qədər ədədlər uyğun gəlir. `Tag` xassəsinə birinci qiymətləri, yəni birinci dəyişdirici üçün *0*, ikinci dəyişdirici üçün *5*, üçüncü dəyişdirici üçün *10*, dördüncü dəyişdirici üçün *15*, beşinci dəyişdirici üçün *20* ədədləri daxil edin.

Dəyişdirici üçün `OnClick` prosedurunu yaradaq. Biz ilk növbədə siyahını təmizləməliyik:

```
ComboBox1.Clear;
```

Bundan sonra siyahı tərtib edilməlidir, yəni gözə görünməyən `ListBox1` komponentindən beş şəhərin adı `ComboBox1` siyahısına yazılmalıdır. Əlavə olunan birinci bəndin nömrəsi `Sender` obyektinin `Tag` xassəsi ilə müəyyən edilir. Onda

```
ComboBox1.Tag:=(Sender as TRadioButton).Tag;
```

yazmalıyıq (sadəcə olaraq `Sender.Tag` yazmaq olmaz).

İndi isə hər bir dəyişdirici üçün *5* şəhər adını `ComboBox1` komponentinə köçürmək lazımdır. Bu əməliyyat dövr daxilində belə yazılmalıdır:

```
for i:=0 to 4 do
ComboBox1.Items.Add(ListBox1.Items[ComboBox1.Tag+i]);
```

Burada, `i` dəyişdiricinin nömrəsini göstərir və onu prosedurun əvvəlində tam tipli elan etmək lazımdır.

İndi isə proqram yolu ilə birinci dəyişdiricini seçərək (`ItemIndex`) `ListBox2` siyahısından şəkli yükləmək (`LoadFromFile`) lazımdır:

```
ComboBox1.ItemIndex:=0;
Image1.Picture.LoadFromFile(ListBox2.Items
                            [ComboBox1.Tag]);
```

Belə prosedur hər bir dəyişdirici üçün yaradılmalıdır. Modulda beş eyni məzmunlu prosedur yazmamaq üçün, növbə ilə hər bir dəyişdiricini seçərək Obyektlər inspektorunda `OnClick` hadisəsi üçün `RadioButton1Click` prosedurunu təyin edin.

Növbəti mərhələ açılan siyahıdan şəhəri seçdikdə həmin şəhərə uyğun şəklin yüklənməsidir. Bunun üçün biz `OnChange` prosedurunu yaradıb oraya cəmi bir sətir yazmalıyıq:

```
procedure TForm1.ComboBox1Change(Sender: TObject);
begin
  Image1.Picture.LoadFromFile(ListBox2.Items
  [ComboBox1.Tag+ComboBox1.ItemIndex]);
end;
```

F9 klavişini basın. Siz hazır forma üzərində heç bir şəkil görməyəcəksiniz. Lakin, dəyişdiricilərin istənilən birini qoşduqda şəkil peyda olacaqdır. Bu ondan irəli gəlir ki, biz proqramın başlanğıc təyinatını göstərməmişik. Proqramın başlanğıc vəziyyətində isə birinci dəyişdirici seçildikdə nə baş verirsə, elə o da olmalıdır. Bu isə o deməkdir ki, biz yeni bir prosedur yaradaraq obyekt parametri kimi birinci dəyişdiricini ötürməklə dəyişdiricilərin emalı prosedurunu (`RadioButton1Click`) çağırmalıyıq. Yəni, formanın boş sahəsində mausun düyməsini basaraq Obyektlər inspektorunda `OnCreate` hadisəsini aktivləşdirmək lazımdır:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  RadioButton1Click(RadioButton1);
end;
```

Bu məsələnin bir variantda həlli şəkil 12.17 – də göstərilmişdir.

Şəkil 12.17. İnteraktiv fotoalbom

Beləliklə, elektron albom məsələsinin modulunun tam mətni belə olacaqdır:

```
unit Unit1;

interface

uses
     Windows,Messages,SysUtils,Classes,Graphics,
     Controls,Forms,Dialogs,StdCtrls,ExtCtrls;

type
```

```
    TForm1 = class(TForm)
      GroupBox1: TGroupBox;
      RadioButton1: TRadioButton;
      RadioButton2: TRadioButton;
      RadioButton3: TRadioButton;
      RadioButton4: TRadioButton;
      RadioButton5: TRadioButton;
      GroupBox2: TGroupBox;
      ComboBox1: TComboBox;
      Image1: TImage;
      ListBox1: TListBox;
      ListBox2: TListBox;
      procedure FormCreate(Sender: TObject);
      procedure ComboBox1Change(Sender: TObject);
      procedure RadioButton1Click(Sender: TObject);

    private
      { Private declarations }

    public
      { Public declarations }

    end;

var
    Form1: TForm1;

implementation

{$R *.DFM}


procedure TForm1.FormCreate(Sender: TObject);
begin
  RadioButton1Click(RadioButton1);
end;


procedure TForm1.ComboBox1Change(Sender: TObject);
begin
  Image1.Picture.LoadFromFile(ListBox2.Items
  [ComboBox1.Tag+ComboBox1.ItemIndex]);
end;


procedure TForm1.RadioButton1Click(Sender: TObject);

Var
    i:Integer;
```

```
begin
  ComboBox1.Clear;
  ComboBox1.Tag:=(Sender as TRadioButton).Tag;
  for i:=0 to 4 do
  ComboBox1.Items.Add(ListBox1.Items[ComboBox1.Tag+i]);
  ComboBox1.ItemIndex:=0;
  Image1.Picture.LoadFromFile(
         ListBox2.Items[ComboBox1.Tag]);
end;
end.
```

12.9. Dialoqlarla iş

Ms Windows sistemi və onun əlavələri ilə işlədikdə biz dialoqlarla demək olar ki, hər addımda rastlaşırıq. Bu dialoqlar ən müxtəlif xarakterlidir. Delphi–də dialoqlar iki üsulla yerinə yetirilir:

- *xüsusi prosedur və funksiyalar vasitəsilə*;
- *dialoq komponentləri ilə.*

12.9.1. Dialoq prosedur və funksiyaları

Delphi–də bir neçə xüsusi prosedur və funksiyalar mövcuddur ki, onlar ümumi təyinatlı sadə dialoqları ekranda əks etdirmək üçün nəzərdə tutulmuşdur. Bu prosedur və funksiyaların bəziləri ilə qısa tanış olaq.

Xüsusi prosedur və funksiyalar iki qrupa bölünür:

- *məlumatı pəncərəyə çıxarmaq üçün*;
- *məlumatı pəncərədən daxil etmək üçün.*

`ShowMessage`, `MessageDlg` və `MessageDlgPos` prosedur və funksiyaları birinci qrupa, `InputBox` və `InputQuery` funksiyaları isə ikinci qrupa aiddir.

`ShowMessage (const` *Msg* `: String); proseduru` – icra olunduqda ekranda məlumat dialoq pəncərəsi peyda olur ki, onun sərlövhəsi icra olunan əlavə faylının adından, pəncərənin özü isə *Msg* məlumat sətri və *Ok* düyməsindən ibarət olur. Biz bu proseduru həll etdiyimiz məsələlərdə dəfələrlə tətbiq etmişik.

`MessageDlg (const` *Msg* `: String;` *AType* `: TMsgDlgType;`
AButtons **`: TMsgDlgButtons;`** ***HelpCtx*** **`: LongInt) : Word;`**

funksiyası – ekranın mərkəzində məlumat pəncərəsi təsvir edir. Burada, *Msg* – ekrana çıxarılan məlumatdan ibarət mətndir. *AType* parametrindən asılı olaraq məlumat pəncərəsi müxtəlif növ olur, məlumatla yanaşı pəncərədə şəkil də təsvir edilir. *AType* parametri aşağıdakı qiymətləri ala bilər:

`mtWarning` –*pəncərə sarı rəngli üçbucaqlı daxilində qara rəngli nida işarəsindən və* `Warning` *sərlövhəsindən ibarət olur*;

`mtError` –*pəncərə qırmızı rəngli dairə daxilində ağ rəngli xaç işarəsi və* `Error` *sərlövhəsindən ibarət olur*;

`mtInformation` –*pəncərə ağ rəngli dairə daxilində göy rəngli* **i** *hərfindən və* `Information` *sərlövhəsindən ibarət olur*;

`mtConfirmation`–*pəncərə ağ rəngli dairə daxilində göy rəngli* **?** *işarəsindən və* `Confirmation` *sərlövhəsindən ibarət olur*;

`mtCustom` –*pəncərədə şəkil olmur, sərlövhədə isə icra olunan əlavə faylının adı təsvir olunur*.

AButton parametri pəncərədə təsvir olunan düymələri əks etdirir və aşağıdakı qiymətlər kombinasiyasını ala bilər: `mbYes`, `mbNo`, `mbOk`, `mbCancel`, `mbHelp`, `mbAbort`, `mbRetry`, `mbIgnore`, `mbAll`. Düymələrin sərlövhəsi bu qiymətlərə uyğun olaraq `Yes`, `Ok`, `Cancel` və s. olur. Bu düymələrdən hər hansı birini (`mbHelp`–dən başqa) basdıqda məlumat pəncərəsi bağlanır.

HelpCtx parametri istifadəçi ***F1*** klavişini basdıqda ekrana çıxan kontekst məlumatı müəyyən edir və onun qiyməti adətən sıfra bərabər olur.

MessageDlgPos (const *Msg* **: String;** *AType* **: TMsgDlgType;**
AButtons **:TMsgDlgButtons;** *HelpCtx* **: LongInt;**
x, y **: Integer) : Word;**

funksiyası – göründüyü kimi, `MessageDlg` funksiyasından yalnız *x* və *y* parametrləri ilə fərqlənir və bu, ekranda məlumat pəncərəsinin vəziyyətini idarə edir.

InputBox(const *ACaption, APrompt, ADefault* **: String) : string;**
funksiyası – mətndən ibarət sətri *daxil etmək* üçün dialoq pəncərəsini ekranda təsvir edir. Bu pəncərədə sərlövhəli mətn sahəsi, `Ok` və `Cancel` düymələri mövcud olur. Burada *ACaption* parametri pəncərənin sərlövhəsini, *APrompt* parametri mətn sahəsinin sərlövhəsini, *ADefault* parametri isə mətn sahəsinə çıxarılan sətri bildirir; əgər istifadəçi `Cancel` düyməsini və ya *Esc* klavişini basarsa, funksiyanın nəticəsi bu sətirdən ibarət olur. Məsələn, əgər proqramda

```
InputBox('İstifadəçi', 'Soyadı', 'Abbasov');
```

yazılarsa, pəncərənin sərlövhəsində `İstifadəçi`, mətn sahəsinin sərlövhəsində `Soyadı`, mətn sahəsində isə `Abbasov` sözləri təsvir olunacaqdır.

InputQuery (const *ACaption, APrompt* **: String;**
var *Value* **: String) : Boolean;**

funksiyası – `InputBox` funksiyasından onunla fərqlənir ki, üçüncü parametrin (susmaya görə sətrin) yerində *Value* dəyişəni istifadə olunur. Bu parametr istifadəçi `Ok` düyməsini basdıqda daxil edilən sətirdən ibarət olur. İstifadəçi `Ok` düyməsini basdıqda, funksiyanın nəticəsi *True*, `Cancel` düyməsini və ya *Esc* klavişini basdıqda isə *False* olur. Məsələn, əgər proqramda

```
Soyad:='Abbasov';
InputQuery('İstifadəçi', ' Soyadı ', Soyad);
```

yazılarsa, `InputBox` funksiyasının ekrana çıxardığı məlumat pəncərəsi ilə eyni olan sorğu ekrana çıxarılır.

12.9.2. Dialoq komponentləri

Delphi–nin Komponentlər palitrasının **`Dialogs`** səhifəsində yerləşən komponentlər dialoqları həyata keçirməyə imkan verir. Bu dialoqlar Windows sistemində adətən faylları açmaq, saxlamaq, çap etmək və s. kimi əməliyyatları yerinə yetirmək üçün istifadə olunduğundan onlara *standart dialoqlar* deyilir.

Komponentlər palitrasının **`Dialogs`** səhifəsində standart dialoqları yerinə yetirən aşağıdakı komponentlər yerləşir:

`OpenDialog` *–açılacaq faylın seçilməsi*;
`SaveDialog` *–yadda saxlanılacaq faylın seçilməsi*;
`OpenPictureDialog` *–açılacaq qrafik faylın seçilməsi*;
`SavePictureDialog` *–yadda saxlanılacaq qrafik faylın seçilməsi*;
`FontDialog` *–şrift parametrlərinin təyini*;
`ColorDialog` *–rəngin seçilməsi*;
`PrintDialog` *–printerdə çapetmə*;
`PrinterSetupDialog` *–printerin seçilməsi və onun parametrlərinin təyini*;
`FindDialog` *–axtarılacaq mətn sətrinin daxil edilməsi*;
`ReplaceDialog` *–axtarılacaq və əvəz olunacaq mətn sətrinin daxil edilməsi*.

Standart dialoq komponentləri qeyri–vizual komponentlərdir, belə ki, layihələndirmə zamanı onları forma üzərində yerləşdirdikdə müvafiq nişanlarla təsvir olunur, layihə yerinə yetirildikdən sonra isə onlar forma üzərində görünmür. Formada yerləşdirdikdən sonra, bu komponentlərin xassələrinə qiymətlər müəyyənləşdirilir və onlar hər hansı bir hadisə ilə əlaqələndirilirlər. Belə hadisə kimi adətən ya menyuların bəndlərinin seçilməsi, ya da düymələrin basılması hadisələri istifadə edilir.

İstənilən standart dialoq **`Execute`** metodu ilə çağrılır. Bu metod–funksiyanın nəticəsi məntiqi qiymət olur: `Ok` düyməsini basdıqda, funksiyanın qiyməti *True*, imtina düyməsini basdıqda isə *False* olur. Dialoq bağlandıqdan

sonra, o öz xassələri vasitəsilə seçilmiş və ya təyin olunmuş qiymətləri proqrama qaytarır. Məsələn, əgər proqramda **`OpenDialog1.FileName`** və ya **`ColorDialog1.Color`** yazılmışdırsa, onda istifadəçinin seçdiyi fayl və ya rəng yüklənəcəkdir.

12.9.2.1. Faylların açılması və yadda saxlanılması

Yuxarıda qeyd etdiyimiz kimi, bu əməliyyatlar uyğun olaraq **`OpenDialog`** və **`SaveDialog`** komponentləri ilə həyata keçirilir. Bu komponentlərin əsas xassələri aşağıdakılardır:

`FileName` –*faylın adını və ona tam yolu göstərir*;
`Title` –*dialoq pəncərəsinin sərlövhəsini müəyyən edir*; əgər bu xassəyə qiymət verilməzsə, susmaya görə pəncərənin sərlövhəsi `Open` (və ya `Save`) `File` olur;
`InitialDir` –*dialoq pəncərəsi açıldıqda təsvir olunan qovluğu müəyyən edir*; əgər bu xassəyə qiymət verilməzsə, pəncərədə cari qovluq təsvir olunur;
`DefaultExt` –*əgər istifadəçi faylın tipini göstərməzsə, avtomatik olaraq fayla onun tipi mənimsədilir*;
`Filter` –*faylların adlarının örtüyünü* (∗.∗,∗.*doc* və s.) *müəyyən edir*. Bu xassəyə susmaya görə qiymət verilməmişdir ki, bu da bütün tip faylların təsvir edilməsi deməkdir.
`FilterIndex` –`Filter` *xassəsində göstərilmiş örtüklərdən hansının istifadə olunduğunu bildirir*; susmaya görə onun qiyməti *1*–dir, yəni birinci örtük istifadə edilir.
`Options` –*pəncərənin xarici görünüşünü və funksional imkanlarını idarə etmək üçün istifadə edilir*.

`Options` xassəsinin iyirmiyə qədər parametrləri vardır və hər bir parametrin qarşısında bayraq qoymaqla onu qoşmaq olar. Bu parametrlərdən bir neçə ən vaciblərinə baxaq:

`ofAllowMultiSelect`–*eyni vaxtda siyahıdan bir neçə fayl seçmək olar*;
`ofCreatePrompt` –*fayl mövcud olmadıqda onun yaradılması üçün sorğu verilir*;
`ofNoLongNames` –*faylların adları qısa formada (ad üçün 8 simvol, tip üçün 3 simvol) təsvir edilir*.

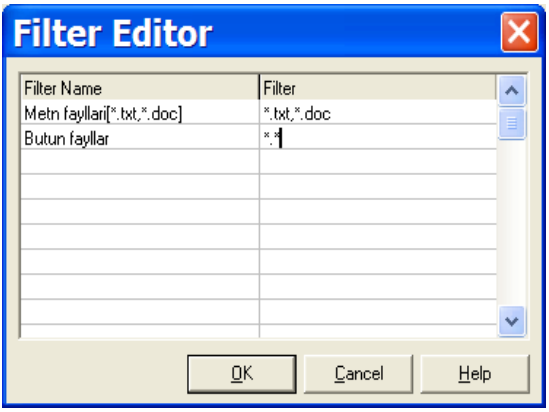
Standart dialoqların hansı fayllarla işləməsi filtrlə (`Filter`) müəyyənləşdirilir. Filtr bir–birindən "|" işarəsi ilə ayrılan qiymətlərdən ibarətdir. Hər bir qiymət təsvir və örtükdən ibarət olur. Təsvir örtüyü izah edən adi mətndir (məsələn, "*mətn faylları*"). Örtük isə faylın adı və tipindən ibarət şablondur (məsələn, ∗.∗, ∗.*txt* və s.). Əgər bu təsvir üçün bir neçə örtük

göstərilərsə, onda onların arasında **;** işarəsi qoyulur. Filtri proqram yolu və ya xüsusi redaktorla müəyyən etmək olar. Məsələn, proqramla filtr belə müəyyənləşdirilə bilər:

```
OpenDialog1.Filter:='Mətn faylları
    |*.TXT;*.DOC;*.Wri;|Bütün fayllar|*.*';
```

Burada, filtr iki örtükdən ibarət olur: mətn faylları üçün və bütün fayllar üçün.

Filtr adətən layihələndirmə zamanı tərtib edilir. Bunun üçün forma üzərində komponenti seçərək Obyektlər inspektorunda `Filter` xassəsi qarşısında mausun düyməsini iki dəfə basmaq lazımdır. Bu zaman ekranda *Filter Editor* (*Filtr redaktoru*) adlı redaktorun pəncərəsi təsvir ediləcəkdir (şəkil 12.18). Bu redaktor `Filter Name` (*Filtrin adı*) və `Filter` (*Filtr*) sütunlarından ibarətdir. Birinci sütunda filtrin təsviri, ikinci sütunda isə uyğun örtük göstərilir.



Şəkil 12.18. Filtr redaktoru

`OpenPictureDialog` və `SavePictureDialog` komponentləri qrafik faylları açmaq və yadda saxlamaq üçün istifadə edilir. Bu komponentlər `OpenDialog` və `SaveDialog` komponentlərindən pəncərənin görünüşü və `Filter` xassəsində müəyyən edilən qiymətlərlə fərqlənir. `Filter` xassəsində susmaya görə aşağıdakı tip qrafik faylların təsviri müəyyən edilmişdir: **.jpg, *.bmp, *.ico, *.emt və *.wmf.*

Bütün bu dialoq komponentləri **`Execute`** metodu ilə çağrılır.

12.9.2.2. Şriftin parametrlərinin seçilməsi

Şriftin adının, ölçülərinin, tərzinin və s. seçilməsi üçün Delphi **`FontDialog`** komponenti təklif edir. Bu komponentin əsas xassələri bunlardır:

`Font` –*şriftin parametrlərini təyin edir*. Şriftin parametrləri bu xassənin `Name` (*ad*), `Style` (*tərz*), `Size` (*ölçü*), `Color` (*rəng*) və s. kimi alt xassələri ilə idarə olunur.

`MaxFontSize` –*şriftin maksimal ölçüsünü müəyyən edir*;
`MinFontSize` –*şriftin minimal ölçüsünü müəyyən edir*;
`Device` –*şriftin quraşdırıldığı qurğunun tipini müəyyən edir*.

`Device` parametri öz növbəsində aşağıdakı üç qiymətdən birini ala bilər:

`fdScreen` –*ekrana çıxarma*;
`fdPrinter` –*printerə çıxarma*;
`fdBoth` –*həm ekrana, həm də printerə çıxarma*.

`Options` –*dialoqun ayrı–ayrı parametrlərini sazlamaq üçün istifadə olunur*.

`Options` xassəsinin özünün bir çox parametrləri mövcuddur.

Misal.

```
if FontDialog1.Execute then
   Label1.Font:=FontDialog1.Font;
```

Bu kodla dialoq pəncərəsindən istifadəçinin seçdiyi şrift yazı üçün tətbiq olunur.

12.9.2.3. Rənglərin seçilməsi

Rənglərlə işləmək üçün Delphi–də **`ColorDialog`** komponenti nəzərdə tutulmuşdur. Bu komponentin əsas xassələri bunlardır:

`Color` –*seçilmiş və ya təyin edilmiş rəngi müəyyən edir*;
`Options` –*dialoqun ayrı–ayrı parametrlərinin sazlanması*.

`Options` xassəsinin özünün aşağıdakı parametrləri vardır:

`cdFullOpen` –*əlavə rəngseçmə panelinin təsviri*;
`cdPreventFullOpen` –"*Определить цвет*" (*Rəngin təyini*) *düyməsinin qoşulmaması*;
`cdShowHelp` –`Help` (*Kömək*) *düyməsinin təsvir edilməsi*;
`cdSolidColor` –*seçilmiş rəng əvəzinə yaxın bütöv rəngin verilməsi*;
`cdAnyColor` –*bütöv olmayan rəngin seçilməsi*.

Misal. Şrift və rənglərin seçilməsi.

Forma üzərinə üç `RadioButton`, bir `CheckBox`, bir `Button`, bir `ColorDialog`, bir `FontDialog` və bir `Label` komponentləri yerləşdirərək onların sərlövhələrini şəkil 12.19 – da olduğu kimi dəyişdirin. Formanın sərlövhəsini pozun. `Label1` komponentinin `AutoSize` xassəsinə *True* qiyməti verin. Həll edəcəyimiz məsələ ondan ibarətdir ki, `RadioButton` dəyişdiriciləri qoşulduqda formanın sərlövhəsi dəyişəcək, formanın rəngi və

Label1 yazısında mətn istifadəçinin seçdiyi rəng və şriftlə təsvir olunacaqdır. CheckBox1 dəyişdiricisini qoşduqda yalnız formanın rəngi dəyişəcək, Button1 düyməsini basdıqda isə formanın əvvəlki vəziyyəti bərpa olunacaqdır.

RadioButton1 (Riyaziyyat) dəyişdiricisi üçün OnClick prosedurunu yaradaq:

```
procedure TForm1.RadioButton1Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
    Form1.Color:=ColorDialog1.Color
  else ShowMessage('Rəng seçilmədi!');
  if FontDialog1.Execute then
  begin
    Form1.Caption:= RadioButton1.Caption;
    Label1.Font.Name:= FontDialog1.Font.Name;
    Label1.Font.Size:= FontDialog1.Font.Size;
    Label1.Font.Color:= FontDialog1.Font.Color;
    Label1.Caption:= 'Mövzu:'+#13#10+
    'PIFAQOR TEOREMI';
  end
  else
  begin
    ShowMessage('Şrift
         seçilmədi!');
    Label1.Caption:=
    'Mövzu:'+#13#10+
    'PIFAQOR TEOREMI';
    Exit;
  end;
end;
```

Şəkil 12.19. Şrift və rənglərin seçilməsi

Riyaziyyat dəyişdiricisi seçildikdə, əgər istifadəçi rəng seçərsə, onda formanın rəngi dəyişdirilir, sərlövhədə Riyaziyyat mətni yazılır. İstifadəçi şrift seçdikdə isə (adı, tərzi, ölçüsü, rəngi və s.) Mövzu: Pifaqor teoremi mətni iki sətirdə çap edilir. İstifadəçi rəngi və şrifti seçmədikdə bu barədə məlumat verilir, hər iki mətn adi şriftlə yazılır.

Analoji əməliyyatları İnformatika və Fizika dəyişdiriciləri üçün də yazaq:

```
procedure TForm1.RadioButton2Click(Sender: TObject);
begin
  if ColorDialog1.Execute then
     Form1.Color:=ColorDialog1.Color
```

```
  else ShowMessage(' Rəng seçilmədi! ');
  if FontDialog1.Execute then
  begin
    Form1.Caption:= RadioButton2.Caption;
    Label1.Font.Name:= FontDialog1.Font.Name;
    Label1.Font.Size:= FontDialog1.Font.Size;
    Label1.Font.Color:= FontDialog1.Font.Color;
    Label1.Caption:= 'Mövzu:'+#13#10+'DELPHI SISTEMI';
  end
  else
  begin
    ShowMessage(' Şrift seçilmədi! ');
    Label1.Caption:= ' Mövzu: '+#13#10+
                     ' DELPHI SİSTEMİ ';
    Exit;
  end;
end;


procedure TForm1.RadioButton3Click(Sender: TObject);
begin

  if ColorDialog1.Execute then
  Form1.Color:= ColorDialog1.Color
  else ShowMessage(' Rəng seçilmədi! ');
  if FontDialog1.Execute then

  begin
    Form1.Caption:= RadioButton3.Caption;
    Label1.Font.Name:= FontDialog1.Font.Name;
    Label1.Font.Size:= FontDialog1.Font.Size;
    Label1.Font.Color:= FontDialog1.Font.Color;
    Label1.Caption:= 'Mövzu:'+#13#10+'NYUTON QANUNLARI';
    End

  else
  begin
    ShowMessage('Şrift seçilmədi!');
    Label1.Caption:= 'Mövzu:'+#13#10+'NYUTON QANUNLARI';
    Exit;
  end;
end;
```

`CheckBox1` komponenti üzərində mausun düyməsini iki dəfə basaraq modula bu proseduru əlavə edin:

```
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  if not CheckBox1.Checked then
  begin
```

```
    Form1.Color:=clBtnFace;
    Exit
  end;
  if ColorDialog1.Execute then
  Form1.Color:=ColorDialog1.Color;

end;
```

Nəhayət, `Button` düyməsi üçün prosedur yaradaq:

```
procedure TForm1.Button1Click(Sender: TObject);

begin

  CheckBox1.Checked:=False;
  RadioButton1.Checked:=False;
  RadioButton2.Checked:=False;
  RadioButton3.Checked:=False;
  Form1.Color:=clBtnFace;
  Form1.Caption:='DƏRS';
  Label1.Caption:='Movzu:';

end;
```

12.9.2.4. Printer və çap parametrlərinin seçilməsi

Çapetmə ilə əlaqədar dialoqlar **`PrintDialog`** və **`PrinterSetupDialog`** komponentləri ilə yerinə yetirilir. Bu komponentlər uyğun olaraq çapetmə və printerin parametrlərini idarə etmək üçündür.

`PrintDialog` komponentinin əsas xassələri aşağıdakılardır:

`FromPage` –*başlanğıc səhifənin nömrəsi*;
`ToPage` –*sonuncu səhifənin nömrəsi*;
`Copies` –*çap olunacaq nüsxələrin sayı*;
`PrintRange` –*çapetmə diapazonu*.

`PrintRange` xassəsi öz növbəsində aşağıdakı qiymətləri ala bilər:

`prAllPages` –*bütün səhifələrin çap edilməsi*;
`prSelection` –*seçilmiş fraqmentin çap edilməsi*;
`prPageNums` –*göstərilən … səhifədən … səhifəyədək çapetmə*;
`Options` –*dialoqun ayrı–ayrı parametrlərini sazlamaq*.

`PrinterSetupDialog` komponenti kağızın ölçüsü, verilmə üsulu və yönü kimi xarakteristikaları müəyyən etməyə imkan verir. Bu komponent üçün bütün sazlama işləri Windows sistemi tərəfindən avtomatik olaraq yerinə yetirilir.

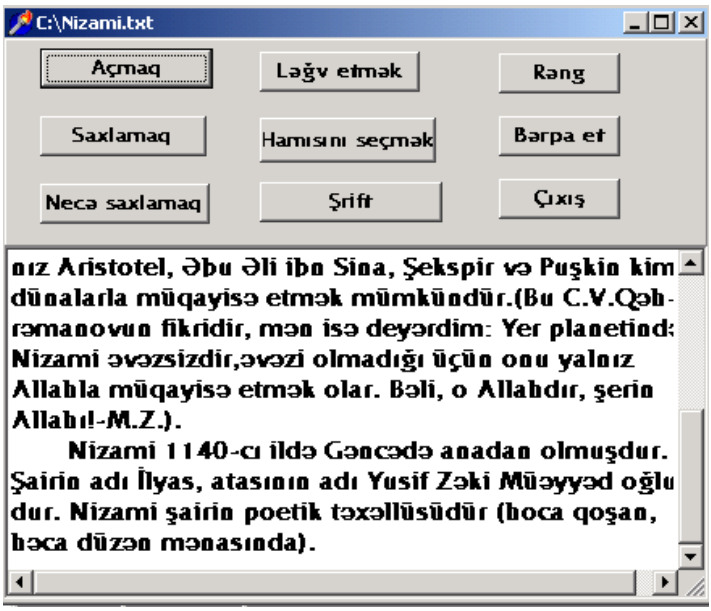
12.10. Mətn redaktorunun yaradılması

Əvvəlcədən şərtləşək ki, bu tam funksiyalı mətn redaktoru olmayacaqdır. Bu misalı həll etməkdən bizim əsas məqsədimiz bir neçə əməliyyatın yerinə yetirilmə nümunəsində mətn redaktorlarının yaradılma prinsipini izah etməkdir.

Forma üzərinə `Panel1` komponenti, onun üzərinə isə 9 ədəd `Button` düyməsi yerləşdirin. Paneli formanın yuxarı hissəsində yerləşdirərək düymələrin sərlövhələrini şəkil 12.20 – də göstərildiyi kimi dəyişdirin.

Formada `Memo1` komponenti yerləşdirərək onun `Align` xassəsinə `alClient`, `ScrollBars` xassəsinə `ssBoth` qiymətləri verin, `Lines` xassəsindən `Memo1` sətrini pozun. Bundan başqa, forma üzərində daha dörd komponent–`OpenDialog`, `SaveDialog`, `FontDialog` və `ColorDialog` komponentləri yerləşdirin.

Sonrakı izahatları isə modulun mətnində şərhlər vasitəsilə verəcəyik.



Şəkil 12.20. Mətn redaktoru

```
unit Unit1;

interface

uses
   Windows,Messages,SysUtils,Classes,Graphics,
   Controls,Forms,Dialogs,StdCtrls,ExtCtrls;

type
  TForm1 = class(TForm)
    Panel1: TPanel;
```

```
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Button5: TButton;
    Button6: TButton;
    Button7: TButton;
    Button8: TButton;
    Button9: TButton;
    Memo1: TMemo;
    OpenDialog1: TOpenDialog;
    SaveDialog1: TSaveDialog;
    FontDialog1: TFontDialog;
    ColorDialog1: TColorDialog;
    procedure Button1Click(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button9Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button7Click(Sender: TObject);
    procedure Button8Click(Sender: TObject);
    procedure Button6Click(Sender: TObject);
    procedure FormClose(Sender: TObject;
              var Action: TCloseAction);

  private
    { Private declarations }

  public
    { Public declarations }

  end;

var
    Form1: TForm1;
    FormColorYad,MemoColorYad: LongInt;
    FAYL: String;

implementation

{$R *.DFM}

procedure TForm1.FormActivate(Sender: TObject);
begin
  OpenDialog1.Title:= ' Mətn faylları ';
  OpenDialog1.Filter:= ' Mətn faylları[*.TXT,*.DOC]
              |*.TXT;*.DOC|Bütün fayllar[*.*]|*.*';
```

```
  OpenDialog1.DefaultExt:='TXT';

  SaveDialog1.Title:='Mətn faylları';{ Dialoq pəncərəsinin
                                       sərlövhəsi müəyyən edilir}

  SaveDialog1.Filter:=' Mətn faylları*.TXT;*.DOC
               |*.TXT;*.DOC|Bütün fayllar*.*|*.*';
  SaveDialog1.DefaultExt:='TXT';
  FormColorYad:=Form1.Color; // Formanın rəngi yadda saxlanır
  MemoColorYad:=Memo1.Color; // Redaktorun rəngi yadda saxlanır
  Memo1.Lines.Clear;        // Redaktor təmizlənir
end;


procedure TForm1.Button1Click(Sender: TObject);
begin
  Memo1.Lines.Clear;
  if OpenDialog1.Execute then { 'Mətn faylları' dialoq
                                pəncərəsi ekrana çıxarılır}
  begin
    FAYL:=OpenDialog1.FileName;//Açılan faylın adı yadda saxlanır
    Form1.Memo1.Lines.LoadFromFile(FAYL); { Açılan fayl
                                            redaktora yüklənir}
    Form1.Caption:=FAYL;     {Açılan faylın adı
                              formanın sərlövhəsinə yazılır}
  end;
end;


procedure TForm1.Button2Click(Sender: TObject);
begin
  Memo1.Lines.SaveToFile(FAYL); // Fayl yadda saxlanır
end;


procedure TForm1.Button9Click(Sender: TObject);
Var
   Rez:TModalResult;

begin
  if Memo1.Modified then
  begin
    Rez:= MessageDlg('Dəyişiklik
    yadda saxlanmayıb! '+#13#10+
    ' Yadda saxlayaq? ',mtConfirmation,[mbOK,mbNo],0);
    if Rez=mrNo then Close; { No düyməsini basdıqda
```

```
                                    dəyişiklik yadda saxlanmır}
   if Rez=mrOK then  { Ok düyməsini basdıqda
                       dəyişiklik yadda saxlanır}
   begin
     Memo1.Lines.SaveToFile(FAYL);
     Close;
   end;
  end
   else Close;
end;


procedure TForm1.Button3Click(Sender: TObject);
begin
  if SaveDialog1.Execute then
  begin
    // Fayl başqa adla yadda saxlanmır
    SaveDialog1.FilterIndex:=2;
    Memo1.Lines.SaveToFile(SaveDialog1.FileName);
  end;
if Memo1.Modified then Memo1.Modified:=False;
end;


procedure TForm1.Button5Click(Sender: TObject);
begin
  // Bütün mətn seçilir
  Memo1.HideSelection:=False;
  Memo1.SelectAll;
end;


procedure TForm1.Button4Click(Sender: TObject);
begin
  // Mətnə dəyişikliklər ləğv edilir
  SendMessage(Memo1.Handle,EM_UNDO,0,0);
end;


procedure TForm1.Button7Click(Sender: TObject);
begin

  // Mətnin rəngi dəyişdirilir
  if ColorDialog1.Execute then
  Memo1.Color:=ColorDialog1.Color;

end;
```

```
procedure TForm1.Button8Click(Sender: TObject);
begin
  // Mətnin rəngi bərpa edilir
  Memo1.Color:=MemoColorYad;
  Form1.Color:=FormColorYad;
end;


procedure TForm1.Button6Click(Sender: TObject);
begin
  // Şrift seçilir
  if FontDialog1.Execute then
  Memo1.FOnt:=FontDialog1.Font;
end;
procedure TForm1.FormClose(Sender: TObject;
                 var Action: TCloseAction);
begin
  if Memo1.Modified then
  if MessageDlg(' Fayl dəyişmişdir! '+#13#10+
  ' Çıxırsınızmı? ',mtConfirmation,
  [mbYes,mbNo],0)=mrYes then
  begin
    if Memo1.Modified then
    begin
      Memo1.Lines.SaveToFile(FAYL);
      Action:=caFree;
    end;
  end
  else Action:=caNone;
end;
end.
```

`OnClose` proseduru ilə `Memo1` komponentində dəyişiklik edilməsi yoxlanır və sorğu aparılır. Əgər istifadəçi proqramdan çıxmazsa, onda `Action` xassəsinə `caNone` qiyməti verilir və forma bağlanmır, əks halda `Action` xassəsinə `caFree` qiyməti verilir və forma və onunla birlikdə əlavə bağlanır.

12.11. Menyularla iş

Menyu Windows sistemində və onun əlavələrində ən vacib elementdir və demək olar ki, elə bir pəncərə yoxdur ki, orada menyu sətri olmasın. Menyu müəyyən funksional əlamətlərə görə birləşdirilmiş bəndlər yığımından ibarətdir və hər bir bənd müəyyən əmri icra edir. Windows sistemindən bilirik ki, menyular əsas menyu və kontekst (peyda olan) menyulardan ibarətdir. *Əsas menyu* menyular sətri kimi pəncərədə həmişə təsvir olunur və bütövlükdə əlavənin bütün funksiyalarını idarə edir. *Kontekst menyu* isə obyekt üzərində

mausun sağ düyməsini basdıqda peyda olur və həmin obyektə aid müəyyən əmrləri icra etmək üçün istifadə edilir.

Delphi–də əsas menyu **MainMenu**, kontekst menyu isə **PopupMenu** komponentləri ilə yaradılır. Bu komponentlər **Standart** səhifəsində yerləşir. Hər iki menyu `TMenuItem` tiplidir. `TMenuItem` sinfi əsas və kontekst menyuların bəndlərini təsvir etmək üçün istifadə olunur. Bu menyuların əsas ümumi xassələri aşağıdakılardır:

Caption xassəsi – `String` tipli `Caption` xassəsi menyunun sərlövhəsindən ibarət sətirdir. Əgər sərlövhədə mətn əvəzinə "–" işarəsi yazılarsa, onda uyğun menyu bəndinin yerində ayrıcı qırıq xətt çəkiləcəkdir.

Bitmap xassəsi – `TBitmap` tipli `Bitmap` xassəsi menyu bəndinin sərlövhəsinin sol tərəfində piktoqramın təsvir edilməsini müəyyənləşdirir, susmaya görə bu xassənin qiyməti `Nil` olur, yəni piktoqram yoxdur.

Enabled xassəsi – `Boolean` tipli `Enabled` xassəsi menyu bəndinin aktivliyini bildirir, əgər onun qiyməti *False* olarsa, onda menyu bəndi aktiv olmur və sərlövhəsi solğun rəngli olur. Bu o deməkdir ki, həmin menyu maus və ya klaviatura ilə icra oluna bilməz. Susmaya görə `Enabled` xassəsinə *True* qiyməti verilmişdir, yəni o aktivdir.

Visible xassəsi – `Boolean` tipli `Visible` xassəsi ekranda menyu bəndinin görünməsini müəyyən edir. Susmaya görə ona *True* qiyməti verilmişdir və menyu bəndi ekranda təsvir olunur.

ShortCut xassəsi – `TShortCut` tipli `ShortCut` xassəsi klavişlər kombinasiyasını müəyyən edir, yəni menyu bəndinin yerinə yetirdiyi funksiyanı müəyyən klavişləri basmaqla da icra etmək mümkün olur. Klavişlər kombinasiyası `Caption` xassəsi ilə də müəyyənləşdirilə bilər **(&** simvolunun köməyi ilə). Bunların fərqi ondadır ki, klavişlər kombinasiyası `Caption` xassəsi ilə müəyyənləşdirildikdə, sərlövhədə simvol altdan xətt çəkilməklə nəzərə çarpdırıldığı halda, `ShortCut` xassəsində klavişlər kombinasiyası menyu bəndinin sağ tərəfində təsvir olunur. Bu xassəyə qiymət vermək üçün Obyektlər inspektorundan istifadə etmək daha əlverişlidir. Klavişlər kombinasiyasını proqramla müəyyən etdikdə isə

ShortCut (*Key* **: Word;** `Shift` **: TShiftState) : TSortCut;**

funksiyasından istifadə etmək lazımdır. Burada `Shift` parametri idarəedici klavişi, *Key* isə hərf–rəqəm klavişini göstərir. Məsələn, *Ctrl+A* klavişlər kombinasiyasını təyin etmək üçün bu funksiya belə yazılmalıdır:

```
mnuSelectAll.ShortCut:= ShortCut(Word('A'),[ssCtrl]);
```

Break xassəsi – `TMenuBreak` tipli `Break` xassəsi menyunun *sütunlara bölünməsini* təyin edir. Bu xassə aşağıdakı qiymətlərdən birini ala bilər:

`mbNone` –*menyu sütunlara bölünmür (susmaya görə)*;
`mbBreak` –*cari bənddən başlayaraq menyu yeni sütun əmələ gətirir*;
`mbBreakBar`–*cari bənddən başlayaraq menyu xətlə ayrılmış yeni sütun əmələ gətirir*.

Checked xassəsi – `Boolean` tipli `Checked` xassəsi menyu bəndinin seçildiyini bildirir. Əgər bu xassəyə *True* qiyməti verilərsə, onda menyu bəndinin sərlövhəsində xüsusi qeydetmə nişanı əmələ gəlir. Susmaya görə `Checked` xassəsinə *False* qiyməti verilmişdir, ona görə də menyu bəndi seçilmir.

RadioItem xassəsi – `Boolean` tipli `RadioItem` xassəsi menyu bəndinin sərlövhəsində əmələ gələn qeydetmə nişanının görünüşünü müəyyən edir. Susmaya görə bu xassəyə *False* qiyməti verilmişdir və qeydetmə nişanı ✓ işarəsindən ibarətdir; *True* qiyməti verildikdə isə belə nişan kimi qalın nöqtə işarəsi təsvir olunur.

Items xassəsi – `TMenuItems` tipli `Items` xassəsi menyu bəndlərindən ibarət massivdir. Bu xassə ilə menyunun ayrı–ayrı bəndlərinə `Items[0]`, `Items[1]` və s. kimi müraciət etmək olar.

Count xassəsi – `Integer` tipli `Count` xassəsi menyuda bəndlərin sayını bildirir. Əgər menyuda bənd yoxdursa, həmin menyu üçün `Count` xassəsi sıfra bərabər olur.

Bu ümumi xassələrdən başqa, `PopupMenu` kontekst menyu komponentinin aşağıdakı xassələri vardır:

AutoPopup xassəsi – `Boolean` tipli `AutoPopup` xassəsi obyektin üzərində mausun sağ düyməsini basdıqda kontekst menyunun ekranda peyda olmasını müəyyən edir. Bu xassəyə susmaya görə *True* qiyməti verildiyindən mausun sağ düyməsini basdıqda kontekst menyu peyda olur. `AutoPopup` xassəsinə *False* qiyməti verdikdə isə kontekst menyu peyda olmayacaqdır.

Aligment xassəsi – `TPopupAligment` tipli `Aligment` xassəsi kontekst menyunun mausun göstəricisinin hansı tərəfində əmələ gəlməsini müəyyən edir. Bu xassənin aldığı aşağıdakı qiymətlərə uyğun olaraq mausun göstəricisi

`paLeft` –*menyunun sol yuxarı kənarını (susmaya görə)*,
`paCenter` –*üfqi vəziyyətə görə menyunun mərkəzini*,
`paRight` –*menyunun sağ yuxarı kənarını*

müəyyən edir.

Komponentin üzərində mausun sağ düyməsini basdıqda kontekst menyunun əmələ gəlməsi üçün, onun PopupMenu xassəsinə qiymət kimi, tələb olunan kontekst menyunun adı mənimsədilməlidir. Məsələn, Label1 komponentinə aid kontekst menyunun yaradılması üçün proqramda

```
Label1.PopupMenu:=PopupMenu1;
```

yazılmalıdır.

Maus və ya klaviatura ilə menyu bəndini seçdikdə baş verən əsas hadisə OnClick hadisəsidir. Əksər hallarda, əlavələrdə eyni bir əməliyyat həm menyu bəndi, həm kontekst menyu və həm də alətlər panelində yerləşən düymə ilə icra olunur. Çünki, həmin əməliyyat eyni bir prosedur (modul) ilə icra olunur. Bunun üçün *imitasiya prinsipindən* istifadə edilir. Menyu bəndinin seçilməsini *imitasiya etmək* üçün Click metodu çağrılmalıdır. Bu zaman bu prosedurun çağrılması istifadəçinin uyğun menyu bəndini seçməsinə ekvivalent olacaqdır.

Misal. Menyu bəndinin seçilməsini imitasiyası.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  mnuOpen.Click;
end;
```

Burada, Button1 düyməsi basıldıqda, mnuOpen (ad şərtidir) menyu bəndinin icra etdiyi əməliyyat yerinə yetiriləcəkdir.

Layihələndirmə zamanı menyuların yaradılması xüsusi *Menyu konstruktorunda* yerinə yetirilir. Menyuları dinamik olaraq, yəni proqramlaşdırma yolu ilə də yaratmaq mümkündür.

12.11.1. Menyu konstruktoru

Əlavələrin layihələndirilməsi prosesində menyuları yaratmaq və ya onları dəyişdirmək üçün Delphi–də Menyu konstruktorundan (*Menu Designer*) istifadə olunur. Bu konstruktoru çağırmaq üçün forma üzərində MainMenu və ya PopupMenu komponentləri yerləşdirərək kontekst menyudan *Menu Designer…* əmrini icra etmək və ya bu komponentlər üzərində mausun düyməsini iki dəfə basmaq lazımdır. Bu redaktorla yaradılan menyu layihə yerinə yetirildikdən sonra necə görünəcəkdirsə, elə o cür də görünür.

Menyu konstruktoru ilə işlədikdə aşağıdakı kontekst menyulardan istifadə etməklə menyuların yaradılması və dəyişdirilməsi prosesini sürətləndirmək olar:

Insert	–menyu bəndini *əlavə* etmək;
Delete	–menyu bəndini *pozmaq*;
Create Submenu	–*alt menyu* yaratmaq;
Select Menu	–menyunu *seçmək*;
Save Template…	–menyunu *şablon* kimi saxlamaq;

`Insert From Template…` –menyunu *şablondan yükləmək*;
`Delete Template…` –menyu *şablonlarını pozmaq*;
`Insert From Resource…` –menyunu *resurslardan yükləmək*.

Redaktorla işləyərkən, Obyektlər inspektorundan istifadə etməklə menyu bəndlərinin xassələrinə qiymətlər verilir.

Menyular yaradıldıqda *drag–and–drop* texnologiyası ilə menyu bəndlərinin yerini dəyişdirmək olar.

Menyular yaradıldıqdan sonra, onlar üçün prosedurlar yaradıldıqda (bənd üzərində mausun düyməsini basmaqla), prosedurun sərlövhəsində bu bəndin nömrəsi göstərilir, məsələn:

```
Procedure Tform1.N3Click(Sender:TObject);
```

Əgər Obyektlər inspektorunda menyu bəndi üçün `Name` xassəsinə ad verilərsə, onda prosedurun sərlövhəsində həmin ad göstərilir, məsələn:

```
Procedure Tform1.mnuCloseClick(Sender:TObject);
```

Burada, `mnuClose` menyu bəndinin adıdır və tamamilə şərti seçilmiş addır. Lakin, unutmayın ki, bu ad yalnız latın hərflərindən və rəqəmlərdən ibarət ola bilər.

12.12. Menyu sətirlərindən ibarət mətn redaktorunun yaradılması

Dialoqlar bölməsində yaratdığımız, düymələrlə idarə olunan mətn redaktorunu yenidən yaradaq. Bu dəfə düymələrin icra etdiyi əmrləri menyular vasitəsilə icra edək. Eyni zamanda menyu bəndlərini proqram yolu ilə deyil, Menyu konstruktoru ilə yaradaq (şəkil 12.21).

Bu redaktorda menyuları belə qruplaşdıraq:

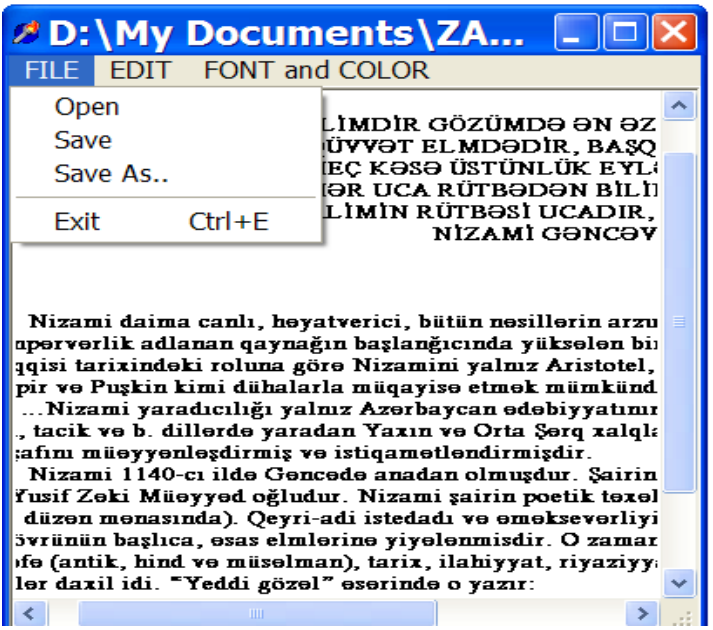
`File` menyusu:
- `Open` –*fayl açmaq*;
- `Save` –*yadda saxlamaq*;
- `Save as…` –*faylı necə yadda saxlamaq*;
- `Exit` –*çıxmaq*.

`Edit` menyusu:
- `Undo` –*ləğv etmək*;
- `SelectAll` –*bütün mətni seçmək*;
- `Reset` –*bərpa etmək*.

`Font and color` menyusu:
- `Font` –*şrift seçmək*;
- `Color` –*rəng seçmək*.

Beləliklə, yaradacağımız redaktorun menyu sətri üç menyudan ibarət olacaqdır. Kontekst menyunu isə aşağıdakı bəndlərdən ibarət tərtib edək:

- `Exit` –*çıxmaq*;
- `Reset` –*bərpa etmək*;
- `Color` –*rəng seçmək*;
- `Font` –*şrift seçmək.*



Şəkil 12.21. Menyulardan ibarət redaktor

Bu dəfə redaktorda heç bir düymə istifadə etməyəcəyik.

Yeni layihə üçün forma üzərinə `Memo`, `MainMenu`, `PopupMenu`, `OpenDialog`, `SaveDialog`, `FontDialog` və `ColorDialog` komponentləri yerləşdirin. `MainMenu` komponentini seçərək, onun üzərində mausun sağ düyməsini basıb, kontekst menyudan *Menu Designer…* konstruktorunu çağırın. Bu konstruktorda bir seçilmiş boş menyu görünəcəkdir. Obyektlər inspektoruna keçərək `Caption` xassəsi qarşısında `File` yazıb *Enter* klavişini basın. Beləliklə, ilk `File` menyusu yaradılacaqdır və Delphi bu menyuya avtomatik olaraq `N1` adı verəcəkdir. Bu menyudan sağ tərəfdə, boş yerdə, mausun düyməsini basıb analoji qayda ilə `Edit` menyusunu və eyni qayda ilə `Font and color` menyusunu yaradın. Yenidən `File` menyusu üzərində mausun düyməsini basın. Bu menyuda yeni bir menyu üçün bənd seçiləcəkdir. Obyektlər inspektorunda `Caption` xassəsinə `Open` mətni daxil edin. Gələcəkdə hansı prosedurun hansı menyu bəndini icra etdiyini başa düşmək üçün menyu bəndlərinə adlar verək (`Name` xassəsi). Bu adları

mnuOpen, mnuSave, ..., mnuColor adlandıraq. Bu menyuların bir neçəsini klavişlər kombinasiyası ilə icra etmək üçün Exit bəndinin ShortCut xassəsinə *Ctrl+E*, Undo bəndinə *Ctrl+U*, SelectAll bəndinə *Ctrl+A* qiymətləri seçin (öz arzunuzla istənilən menyu bəndi üçün klavişlər kombinasiyası təyin edə bilərsiniz). File menyusunda Exit bəndini digər bəndlərdən ayıraq. Bunun üçün Save as... menyu bəndini yaratdıqdan sonra, növbəti təklif olunan bəndin Caption xassəsinə ad deyil, "–" işarəsi yazıb *Enter* klavişini basın.

Kontekst menyunu yaratmaq üçün, PopupMenu komponentini seçərək, analoji əməliyyatları icra edin. Kontekst menyunun bəndlərini isə belə adlandırın (Name xassəsi): mkExit, mkReset, mkColor, mkFont.

Menyu konstruktorunu bağlayın. Mətn redaktorunda menyu sətri yaradılmış olacaqdır.

İndi Memo1 komponentini seçin. Redaktorun müştəri oblastının bütün pəncərəni əhatə etməsi üçün onun Align xassəsinə alClient qiyməti verin. Redaktor bütün pəncərə boyu açılacaqdır. Redaktorda fırlatma zolaqlarının olması üçün onun ScrollBars xassəsinə ssBoth (hər iki zolaq var) qiyməti verin. Redaktorun sərlövhəsini (Memo1) pozun. Bunun üçün Lines xassəsi qarşısındakı üç nöqtə təsvirli düyməni iki dəfə basaraq açılan pəncərədən Memo1 sözünü pozun.

OpenDialog1 komponentini seçib Filter xassəsi qarşısında mausun düyməsini basaraq *Filter Editor* redaktorunu çağırın. Bu redaktorun birinci sütununun birinci sətrinə *Mətn faylları* ∗.*txt*, ∗.*doc*, ikinci sütununun həmin sətrində ∗.*txt*, ∗.*doc* yazın. İkinci sətrin birinci sütununda *Bütün fayllar* ∗.∗, ikinci sütununda isə ∗.∗ yazıb *Ok* düyməsini basın. Obyektlər inspektorunda DefaultExt xassəsinə *'txt'* qiyməti daxil edin. Bütün bu əməliyyatları SaveDialog1 komponenti üçün təkrar edin.

Layihə modulunda bütün prosedurlar menyu bəndləri üzərində, OnActivate (forma aktivləşdikdə) proseduru isə forma üzərində mausun düyməsini bir dəfə basmaqla yaradılacaqdır. Əslində modulun bütün prosedurları (kontekst menyudan başqa) əvvəlki redaktorda düymələr üçün yaradılmış prosedurlardır. Kontekst menyular üçün prosedurlarda isə yeni kodlar yazılmayacaq, sadəcə olaraq əsas menyu bəndlərini imitasiya kodu yazılacaqdır.

Beləliklə, menyularla idarə olunan mətn redaktorunun modulunun tam mətni belə olacaqdır:

```
unit Unit1;

interface

uses
   Windows,Messages,SysUtils,Classes,
   Graphics,Controls,Forms,Dialogs,
   Menus, StdCtrls;
```

```
type
   TForm1 = class(TForm)
      Memo1: TMemo;
      MainMenu1: TMainMenu;
      N1: TMenuItem;
      Y1: TMenuItem;
      N2: TMenuItem;
      mnuOpen: TMenuItem;
      mnuSave: TMenuItem;
      mnuSaveAs: TMenuItem;
      N6: TMenuItem;
      mnuExit: TMenuItem;
      mnuUndo: TMenuItem;
      mnuSelectAll: TMenuItem;
      mnuReset: TMenuItem;
      mnuFont: TMenuItem;
      mnuColor: TMenuItem;
      PopupMenu1: TPopupMenu;
      mkExit: TMenuItem;
      mkReset: TMenuItem;
      mkColor: TMenuItem;
      mkFont: TMenuItem;
      OpenDialog1: TOpenDialog;
      SaveDialog1: TSaveDialog;
      FontDialog1: TFontDialog;
      ColorDialog1: TColorDialog;
      procedure FormActivate(Sender: TObject);
      procedure mnuOpenClick(Sender: TObject);
      procedure mnuSaveClick(Sender: TObject);
      procedure mnuSaveAsClick(Sender: TObject);
      procedure mnuExitClick(Sender: TObject);
      procedure mnuUndoClick(Sender: TObject);
      procedure mnuSelectAllClick(Sender: TObject);
      procedure mnuResetClick(Sender: TObject);
      procedure mnuFontClick(Sender: TObject);
      procedure mnuColorClick(Sender: TObject);
      procedure FormClose(Sender:TObject;
                  var Action:TCloseAction);

   private
     { Private declarations }

   public
     { Public declarations }

   end;

var
   Form1: TForm1;
   FormColorYad,MemoColorYad: LongInt;
```

```
    FAYL: String;

implementation

{$R *.DFM}

procedure TForm1.FormActivate(Sender: TObject);

begin
  OpenDialog1.Title:= ' Mətn faylları ';
  OpenDialog1.Filter:= ' Mətn faylları[*.TXT,*.DOC]
              |*.TXT;*.DOC|Bütün fayllar[*.*]|*.*';
  OpenDialog1.DefaultExt:= 'TXT';
  SaveDialog1.Title:= ' Mətn faylları ';
  SaveDialog1.Filter:= ' Mətn faylları*.TXT;*.DOC|
              *.TXT;*.DOC|Bütün fayllar*.*|*.*';
  SaveDialog1.DefaultExt:='TXT';
  FormColorYad:= Form1.Color;
  MemoColorYad:= Memo1.Color;
  Memo1.Lines.Clear;
  Memo1.PopupMenu:= PopupMenu1;
end;


procedure TForm1.mnuOpenClick(Sender: TObject);
begin
  Memo1.Lines.Clear;
  if OpenDialog1.Execute then
  begin
    FAYL:= OpenDialog1.FileName;
    Form1.Memo1.Lines.LoadFromFile(FAYL);
    Form1.Caption:= FAYL;
  end;
end;


procedure TForm1.mnuSaveClick(Sender: TObject);
begin
  Memo1.Lines.SaveToFile(FAYL);
end;


procedure TForm1.mnuSaveAsClick(Sender: TObject);
begin
  if SaveDialog1.Execute then
  begin
    SaveDialog1.FilterIndex:=2;
    Memo1.Lines.SaveToFile(SaveDialog1.FileName);
  end;
```

```
  if Memo1.Modified then Memo1.Modified:=False;
end;


procedure TForm1.mnuExitClick(Sender: TObject);
Var
   Rez:TModalResult;
begin
  if Memo1.Modified then
  begin
    Rez:=MessageDlg(' Dəyişiklik yadda saxlanmayıb! '
                    +#13#10+' Yadda saxlayaq? ',
                    mtConfirmation,[mbOK,mbNo],0);
    if Rez= mrNo then Close;
    if Rez= mrOK then
    begin
      Memo1.Lines.SaveToFile(FAYL);
      Close;
    end;
  end
   else Close;
end;


procedure TForm1.mnuUndoClick(Sender: TObject);
begin
  SendMessage(Memo1.Handle,EM_UNDO,0,0);
end;


procedure TForm1.mnuSelectAllClick(Sender: TObject);
begin
  MEmo1.HideSelection:=False;
  Memo1.SelectAll;
end;

procedure TForm1.mnuResetClick(Sender: TObject);
begin
  Memo1.Color:= MemoColorYad;
  Form1.Color:= FormColorYad;
end;


procedure TForm1.mnuFontClick(Sender: TObject);
begin
  if FontDialog1.Execute then
  Memo1.Font:=FontDialog1.Font;
end;
```

```
procedure TForm1.mnuColorClick(Sender: TObject);
begin
  if ColorDialog1.Execute then
     Memo1.Color:= ColorDialog1.Color;
end;


procedure TForm1.FormClose(Sender:TObject;
           var Action:TCloseAction);
begin
  if Memo1.Modified then
    if MessageDlg(' Fayl dəyişmişdir! '+#13#10+
                  ' Çıxırsınızmı? ',mtConfirmation,
                  [mbYes,mbNo],0)=mrYes then
     Begin
       if Memo1.Modified then
         begin
         Memo1.Lines.SaveToFile(FAYL);
         Action:=caFree;
         end;
     end
       else Action:=caNone;
end;
end.
```

12.13. İdarəedici elementlərin əlaqələndirilməsi

Yuxarıda qeyd etdik ki, əlavələrdə eyni bir əməliyyat müxtəlif idarəedici elementlərlə yerinə yetirilir. Bir çox idarəedici elementlər proqramın eyni fraqmentini istifadə etməklə eyni əməliyyatların yerinə yetirilməsini imitasiya edirlər. Bu zaman hər hansı bir əməliyyatın yerinə yetirilməsinə icazə verilmirsə, bu əməliyyatı icra edən digər elementlər də qapanmalı, fəaliyyətsiz olmalıdır. Bütün bu əlaqələndirmə və razılaşmaları, əlbəttə, proqramçı özü yerinə yetirməlidir. Yuxarıdakı mətn redaktoru məhz bu prinsiplə layihələndirildi və bu əlaqələri biz özümüz təmin etdik.

Lakin, idarəedici elementlərin qarşılıqlı əlaqələri problemini daha əsaslı şəkildə həll etmək üçün Delphi Komponentlər palitrasının **Standart** səhifəsindən xüsusi **ActionList** komponenti təklif edir. Bu komponent razılaşdırılmış şəkildə əməliyyatlar yerinə yetirən idarəedici elementlərin kodlarını birləşdirir. `ActionList` komponenti `TAction` tipli `Action` *əməliyyatlar obyektlərindən* ibarətdir. Bu obyektlərin əsas xassələri `Caption`, `ImageIndex`, `Enabled`, `Visible`, `Checked`, `Hint`, `ShortCut` xassələridir. Bu xassələr `TMenuItem` tipli menyu bəndlərinin analoji xassələri ilə eynidir. `Action` obyektləri üçün yeni hadisələr `OnExecute` və `OnUpdate` hadisələridir.

`OnExecute` hadisəsi `Action` obyektini seçdikdə baş verir. Bu hadisə emaledicisində kodlar yerləşir ki, obyekt seçildikdə bu kodlar yerinə yetirilir.

`Action` obyekti üçün `OnUpdate` hadisəsi əlavə gözləmə rejimində olduqda baş verir. Bu hadisə emaledicisinin prosedurunda interfeys elementlərinin qapanması və ya qoşulması ilə əlaqədar əməliyyatlar kodlaşdırılır.

İdarəedici elementlərin konkret `Action` obyekti ilə əlaqəsi eyniadlı `Action` xassəsi ilə təmin olunur. Bu xassə `TBasicAction` tiplidir. `Button1` düyməsi `Action1` obyekti ilə bu kodla əlaqələndirilir:

```
Button1.Action:= Action1;
```

Bu zaman `Button1` düyməsi basıldıqda `Action1` obyekti üçün yaradılmış prosedur (`OnExecute` hadisə emaledicisi) yerinə yetiriləcəkdir. Məsələn, `mnuOpen.Action:=ActionOpen;` kodu, `mnuOpen` menyu bəndi ilə alətlər panelindəki düymənin yerinə yetirdiyi əməliyyatları eyni bir prosedurla yerinə yetirməyə imkan verir.

Əlavənin layihələndirilməsi mərhələsində `ActionList` komponentinin `Action` obyektləri ilə işlədikdə xüsusi redaktordan istifadə oluna bilər.

Bu redaktor `ActionList` komponentinin kontekst menyusunun *Action List Editor* əmri ilə ekrana çıxarılır. Bu redaktorla birgə Obyektlər inspektoru da istifadə edilir. Baxmayaraq ki, idarəedici elementlərin və `Action` obyektlərinin xassələri eynidir, bu xassələrin tətbiqində müəyyən xüsusiyyətlər vardır. Belə ki, `Action` obyektinə idarəedici elementi, məsələn, düymə və ya menyu bəndini qoşduqda `Caption` xassəsinin adı kimi `Action` obyektinin adı istifadə olunacaqdır. Bu, `Checked` xassəsinə də aiddir.

12.14. Konteynerlər

Əlavələr yaradıldıqda müəyyən elementlərin qrup şəklində birləşdirilməsi tələb olunur. Buna alətlər panelini misal göstərmək olar. Elementləri qruplaşdırmaqdan əsas məqsəd ondan ibarətdir ki, pəncərənin ölçülərini dəyişdikdə bu elementlər köhnə yerində qalmır, üzərində yerləşdikləri panellə birgə yerlərini dəyişirlər.

Elementlərin qruplarda birləşdirilməsi xüsusi komponentlərlə yerinə yetirilir. Konteyner vizual komponentdir, üzərində digər komponentləri yerləşdirməyə, onları bir qrupda birləşdirməyə imkan verir və onların sahibi olur. Qeyd edək ki, forma özü də konteynerdir və üzərində yerləşən bütün komponentlərin sahibidir.

Delphi universal konteynerlər kimi aşağıdakı komponentləri təklif edir:

`GroupBox` *qrupu*;
`Panel` *paneli*;
`ScrollBox` *fırlatma oblastı*;
`Frame` *freymi (haşiyə).*

12.14.1. Qrup

Bu konteyner **GroupBox** komponenti ilə yaradılır. Onun üzərində əsasən funksional əlaqəli vizual komponentlər yerləşdirilir. Konteyner düzbucaqlı haşiyədən ibarətdir, onun sol yuxarı küncündə konteynerin sərlövhəsi (`Caption`), üzərində isə elementlər yerləşir. Bu komponentin tətbiqi ilə biz məsələ həll etmişik (Elektron fotoalbom məsələsi).

12.14.2. Panel

Bu konteyner üzərində istənilən idarəedici elementlər yerləşdirilə bilər. Panel adətən alətlər paneli və vəziyyətlər sətri yaratmaq üçün tətbiq edilir. Panel – Komponentlər palitrasının **Standart** səhifəsində yerləşən **Panel** komponenti ilə yaradılır.

Panel daxili və xarici faskalardan ibarət olur. *Daxili faska* paneli haşiyəyə alır, *xarici faska* isə daxili faskanın ətrafında təsvir olunur. Hər bir faskanın eni `TBewelWidth` tipli `BewelWidth` xassəsi ilə müəyyən olunur. Bu xassəyə `1..MaxInt` aralığında istənilən qiymət vermək olar. Susmaya görə, onun qiyməti *1*–ə bərabərdir.

Daxili və xarici faskaların görünüşünü `BevelInner` və `BevelOuter` xassələri (`TPanelBevel` tipli) müəyyən edir. Bu xassələrin hər biri aşağıdakı qiymətləri ala bilər:

`bvNone` –*faska yoxdur*;
`bvLowered` –*faska çökmüş vəziyyətdədir*;
`bvRaised` –*faska azca qaldırılmış vəziyyətdədir*;
`bvSpace` –*faskanın təsiri məlum deyil*.

Faskalar arasında məsafə də ola bilər. Bu məsafənin eni `TBorderWidth` tipli `BorderWidth` xassəsi ilə müəyyən olunur. Susmaya görə onun qiyməti sıfra bərabərdir, yəni faskalar arasında məsafə yoxdur.

Panelin sərlövhəsinin vəziyyəti `TAligment` tipli `Aligment` xassəsi ilə müəyyən edilir. Bu xassə aşağıdakı qiymətləri ala bilər:

`taLeftJustif` –*sol tərəfə görə düzləndirmə*;
`taCenter` –*mərkəzə görə düzləndirmə (susmaya görə)*;
`taRightJustify` –*sağ tərəfə görə düzləndirmə*.

Əgər panelin sərlövhəsi lazım deyilsə, onda `Caption` xassəsinin qarşısında boş sətir olmalıdır.

Panel komponentinin tətbiqi ilə bir neçə misal həll etmişik. Bu xassələrin təsirini əyani müşahidə etmək üçün forma üzərində yalnız `Panel` komponenti yerləşdirib, bu xassələrə müxtəlif qiymətlər verməklə, onun müxtəlif vəziyyətlərini müşahidə edin.

12.14.3. Fırlatma oblastı

Fırlatma oblastı **ScrollBox** komponenti ilə təsvir olunur. Komponent daxilində olan elementlər tam görünmürsə, onda avtomatik olaraq ya üfqi, ya şaquli və ya hər iki fırlatma zolaqları əmələ gəlir. Fırlatma oblastı böyüdüldükdə fırlatma zolaqları avtomatik olaraq yox olur.

Fırlatma oblastının bütün müştəri oblastını əhatə etməsi üçün `Align` xassəsinə `alClient` qiyməti vermək lazımdır.

Fırlatma zolağının avtomatik peyda olması üçün `Boolean` tipli `AutoScroll` xassəsinə *True* qiyməti vermək lazımdır. Fırlatma sahəsi haşiyədə ola bilər. Bunun üçün `BorderStyle` xassəsindən istifadə olunur və bu xassə aşağıdakı qiymətləri ala bilər:

`bsNone` –*haşiyə yoxdur*;
`bsSingle` –*haşiyə var (susmaya görə).*

12.14.4. Freymlər

Freym də digər komponentlər üçün konteyner olmaqla, forma kimi yaradılır, lakin, ondan fərqli olaraq freym özü başqa konteynerdə, məsələn, forma və ya panel üzərində yerləşdirilə bilər. Bu konteynerlə işləmək üçün Delphi **Frame** komponenti təklif edir.

Freymlə iş iki mərhələdən ibarətdir:

– *Freymin yaradılması və konstruksiya edilməsi*;
– *Freymin yaradılması və paneldə yerləşdirilməsi.*

Freym yaratmaq üçün *File/New Frame* (*Fayl/Yeni Freym*) əmrini icra etmək lazımdır. Digər elementlər freym üzərində forma üzərində yerləşdirildiyi kimi yerləşdirilir.

Freymi forma üzərində yerləşdirmək üçün komponentlər palitrasından `Frame` komponentini seçərək forma üzərində, lazım olan yerdə, yerləşdirmək lazımdır. Bu zaman *Select Frame to insert* (*Yerləşdirmək üçün freymi seçin*) dialoq pəncərəsi açılacaqdır ki, bu pəncərədən həmin freymin adını seçib *Ok* düyməsini basmaq lazımdır. Freym forma üzərində yerləşdikdən sonra, avtomatik olaraq, modulun `Uses` bölməsinə freymə istinad əlavə olunur.

On üçüncü fəsil

ƏLAVƏLƏRDƏ FORMALARIN YERİ

Forma ən əsas vizual komponentdir və istənilən əlavənin yaradılmasında mərkəzi yer tutur. Forma və pəncərə eyni elementlərdir.

Forma `TForm` tipli **`Form`** komponenti ilə təsvir olunur və əlavənin layihələndirilməsi məhz formadan başlayır. Bu məqsədlə Delphi yükləndikdə o, proqramçıya avtomatik olaraq **`Form1`** boş forması təqdim edir. Biz indiyədək yaratdığımız ən sadə və nisbətən mürəkkəb layihələrin hamısını forma üzərində konstruksiyalaşdırmışıq.

Bununla bərabər, nə qədər təəccüblü görünsə də, formasız əlavələr də yaratmaq mümkündür. Belə layihələr əyani olmur, müəyyən dərəcədə ənənəvi proqramlaşdırmaya çevrilir. Lakin, eyni zamanda, formasız əlavələrin üstünlüyü ondan ibarətdir ki, belə proqramlar yaddaşda müqayisə olunacaq dərəcədə az yer tutur.

Formasız layihələrin əksinə olaraq Delphi–də nəinki bir formadan ibarət, hətta çoxformalı əlavələr də yaratmaq mümkündür.

Biz bu bölmədə formasız əlavələrin yaradılması qaydasını öyrənəcək, formanın xarakteristikaları ilə daha yaxından tanış olacaq və çoxpəncərəli layihələrin yaradılması prinsipini öyrənərək formaların qarşılıqlı təsirlərini nəzərdən keçirəcəyik.

13.1. Formasız əlavələr

Elə proqramlar vardır ki, onlar ən mürəkkəb funksiyaları yerinə yetirir və bu zaman istifadəçi ilə dialoqa, demək olar ki, zərurət yaranmır. Belə proqramları pəncərəli proqramlar kimi yaratmağa ehtiyac qalmır, pəncərələrin

olmaması hesabına isə bu proqramlar yaddaşdan qənaətlə istifadə etməyə imkan verir və daha sürətlə icra olunur. Bu proqramlara *utilitləri* misal göstərmək olar.

Pəncərəsiz proqramlar çox asanlıqla yaradılır. Bu məqsədlə *Project/Remove from Project…* (*Layihə/Layihədən çıxarmaq…*) əmirni icra etmək və ya alətlər panelində *Remove file from Project* düyməsini basmaq lazımdır. Açılan dialoq pəncərəsində `Unit1` və `Form1` seçərək *OK* düyməsini basmaqla onları pozun. Baxmayaraq ki, forma və yunit pozulmuşdur, alətlər panelindəki *Run* düyməsi aktivdir. Yəni, pəncərənin pozulması hələ proqramın pozulması demək deyildir. Sadəcə olaraq bu proqram ekranda görünmür və onu ekranda təsvir etmək üçün *Project / View Source* əmrini icra etmək lazımdır. Bu əmrin icrasından sonra, ekranda görünən proqramın mətni belə olacaqdır:

```
Program Project1;

Uses
     forms;

{$R*.RES}

Begin
  Application.Initialize;

  Application.run;
end.
```

Proqramı nəzərdən keçirdikdə görürük ki, bu kitabın onuncu fəslində göstərilmiş formaya malik layihə faylından onun fərqi ondadır ki, `Uses` bölməsində `Unit1.pas` moduluna istinad və proqramın icra olunan hissəsində `Form1` formasının yaradılması ilə əlaqədar

```
Application.CreateForm(TForm1, Form1);
```

operatoru yoxdur. ***Ctrl+F9*** klavişlərini basaraq proqramı kompilyasiya edin. Əgər kompilyasiya olunmuş proqramın yaddaş həcmini yoxlasanız, görəcəksiniz ki, formasız proqram təxminən *170 Kb* yer tutur. Bu proqramın `Uses` bölməsini və operatorlarını da pozaraq aşağıdakı proqramı yadda saxlayın:

```
Program Project1;

{$R*.RES}

begin

end.
```

Bu boş proqramı kompilyasiya etdikdən sonra, o, təxminən *15 Kb* yer tutacaqdır. Əgər xatırlayırsınızsa, heç bir əməliyyat yerinə yetirməyən boş formadan ibarət proqram təxminən *275 Kb* həcmə malik idi. Bu müqayisəni ona görə gətiririk ki, əslində formasız proqramı belə kiçik həcmə malik proqramdan

yaratmaq olar. Məsələn, sonuncu mətndən ibarət proqrama müəyyən bir funksiyanı yerinə yetirməyi həvalə edək. Bunun üçün proqramın kodunu belə dəyişdirin:

```
program Project1;

Uses
     Windows;

{$R *.RES}

Begin
  if CreateDirectory('c:\NoForm',nil)=True
  then
  MessageBox(0,'Qovluq yaradıldı!','Projecr1',
              MB_OK+MB_ICONINFORMATION);
end.
```

Bu proqramda `Windows` modulu, *Windows API*–dən isə `CreateDirektory` və `MessageBox` funksiyaları istifadə olunmuşdur. Proqramın funksiyası yalnız `NoForm` adlı qovluq yaratmaqdan ibarətdir (bu zaman diskdə `NoForm` adlı qovluq mövcud olmamalıdır). Qovluğu `CreateDirectory` funksiyası yaradır. Bu proqramın da yaddaşda tutduğu yeri müqayisə etsəniz, görərsiniz ki, yaddaşda ciddi fərq olmamışdır.

13.2. Formanın xarakteristikaları

Tipik forma haşiyələnmiş düzbucaqlı pəncərədən ibarətdir. Əksər pəncərələr sərlövhə sətrindən və bu sətirdə yerləşən pəncərəni bükmə, ekran boyu açma və bağlama düymələrindən ibarət olur. Pəncərələrdə adətən menyu sətri və vəziyyətlər sətri yerləşir. Zərurət yarandıqda, pəncərədə fırlatma zolaqları əmələ gəlir. Formanın yerdə qalan hissəsi müştəri sahəsi adlanır. Müştəri sahəsi istifadəçinin sərəncamında olur və bu sahədən o, mətn, qrafik materiallar və s. daxil edir və ya orada idarəedici elementlər yerləşdirir.

Bütün vizual komponentlər kimi, forma da, xassə, metod və hadisələrə malikdir. Daha ümumi olan xassə, metod və hadisələrlə komponentləri öyrəndikdə tanış olduq. Bu xassə və metodların əksəriyyəti formaya aiddir (`Caption`, `Color`, `Enabled`, `OnKeyPress` və s.). Bütün vizual komponentlər üçün ümumi olan xassə, metod və hadisələrdən başqa, forma üçün səciyyəvi olan xassə, metod və hadisələr mövcuddur.

Layihəyə yeni forma əlavə etdikdə Delphi avtomatik olaraq sinfin bir nüsxəsini (`Form1`, `Form2` və s.) yaradır. Formanın nüsxəsi `Create` metodu ilə yaradılır. Məsələn, yeni forma yaratmaq üçün prosedura

```
Form2:= TForm2.Create(Application);
```

kodu yazmaq lazımdır. Forma yaradıldıqda və istifadə olunduqda aşağıdakı ardıcıllıqla `TNotifyEvent` tipli hadisələr əmələ gəlir: `OnCreate`, `OnShow`,

OnResize, OnActivate və OnPaint. Bu hadisələrdən yalnız OnCreate hadisəsi forma yaradılarkən bir dəfə baş verir, qalan hadisələr isə hər dəfə forma təsvir olunduqda, aktivləşdirildikdə və s. baş verir. OnCreate hadisə emaledicisinə adətən formanın xassələrinə və digər parametrlərə başlanğıc qiymətlər vermək və forma yaradıldıqda bir dəfə baş verən əməliyyatları təsvir etmək üçün kodlar yazılır (mətn redaktoru yaratdıqda biz bu emaledici vasitəsilə – filtri və digər məsələlərdə isə başlanğıc qiymətləri müəyyən etmişdik).

Hər dəfə forma üzərində mausun düyməsini basdıqda o, fokus alır, aktivləşir və OnActivate hadisəsi baş verir. Forma fokusu itirdikdə OnDeActivate hadisəsi baş verir.

Formanı bağlamaq üçün Close metodundan istifadə olunur. Formanı bağladıqda layihənin işi tamamilə dayandırılır. Close proseduru formanı məhv etmir, sadəcə olaraq bağlayır, yəni bağlanmış formanı yenidən ekrana çıxarmaq olar. Bunun üçün **Show** və ya **ShowModal** metodları tətbiq edilməlidir.

Formanı bağladıqda və məhv etdikdə ardıcıl olaraq aşağıdakı hadisələr baş verir: OnCloseQuery, OnClose, OnDeActivate, OnHide və OnDestroy.

TCloseEvent tipli OnClose hadisəsi adətən birbaşa forma bağlanmazdan əvvəl baş verir. Bunun üçün hadisə emaledicisinə TCloseAction tipli Action dəyişəni ötürülür ki, bu dəyişən aşağıdakı qiymətləri ala bilər:

caNone	*–formanı bağlamaq olmaz*;
caHide	*–forma gizlədilir*;
caFree	*–forma məhv edilir və onunla əlaqədar olan yaddaş təmizlənir*;
caMinimize	*–formanın pəncərəsi bükülür.*

Close metodu ilə formanı bağladıqda Action dəyişəni, susmaya görə, caHide qiyməti, formanı məhv etdikdə isə (məsələn, Destroy metodu ilə) caFree qiyməti alır.

Misal. Formanı bağlamaq proseduru.

```
Procedure TForm1.FormClose(Sender:TObject;
                  var Action:TCloseAction);
begin
  If Memo1.Modified then
     Action:= caNone else Action:= caHide;
end;
```

Forma bağlanarkən Memo1 mətn redaktorunun məzmununda dəyişiklik edilməsi yoxlanır: əgər mətn dəyişmişdirsə, forma bağlanmır.

Hər dəfə formanın ölçülərini dəyişdirdikdə TNotifyEvent tipli OnResize hadisəsi baş verir.

Formanın tərzi `TFormStyle` tipli `FormStyle` xassəsi ilə müəyyən edilir. Bu xassə aşağıdakı qiymətləri ala bilər:

`fsNormal` –*əksər pəncərələrdə istifadə olunan standart tərz*;
`fsMDIChild` –*çoxsənədli əlavələrdə törəmə forma*;
`fsMDIForm` –*çoxsənədli əlavələrdə valideyn forma*;
`fsStayOnTop` –*bütün pəncərələrdən üstdə duran forma*.

Hər bir forma haşiyəyə malik olur. Haşiyə formanın sərhədlərini məhdudlaşdırır. Haşiyənin görkəmi və vəziyyətini `TFormBorderStyle` tipli `BorderStyle` xassəsi müəyyən edir. Bu xassə aşağıdakı qiymətləri ala bilər:

`bsDialog` –*dialoq forması*;
`bsSingle` –*dəyişməz ölçülü forma*;
`bsNone` –*formanın haşiyəsi və sərlövhəsi yoxdur və forma ölçülərini dəyişə bilməz*;
`bsSizeable` –*ölçülərini dəyişən adi forma (susmaya görə), sərlövhə sətri var və bütün düymələrə malik ola bilər*;
`bsToolWindow` –*alətlər paneli forması (bu halda forma alətlər paneli kimi istifadə edilir)*;
`bsSizeToolWin` –*ölçülərini dəyişən alətlər paneli forması*.

Misal. Formanın ölçülərinin dəyişdirilməsi.

```
Prosedure Tform1.Button1Click(Sender:TObject);
begin
  Form1.Width:= Form1.Width+50;
  Form1.Height:= Form1.Height+30;
end;
```

`Button1` düyməsini basdıqda `BorderStyle` xassəsinin qiyməti hətta `bsNone`, `bsDialog` və `bsSingle` olsa da, formanın eni 50, hündürlüyü isə 30 piksel artacaqdır.

Formanın sərlövhə sətrində yerləşən 4 düymənin təsviri `TBorderIcon` tipli `BorderIcons` xassəsi ilə müəyyən edilir. Bu xassə aşağıdakı qiymətləri ala bilər:

`biSystemMenu` –*sərlövhədə sistem menyusu düyməsi var*;
`biMinimize` –*sərlövhədə pəncərəni bükmə düyməsi var*;
`biMaximize` –*sərlövhədə pəncərəni bütün ekran boyu açma düyməsi var;*
`biHelp` –*sərlövhədə sual işarəsi təsvirli məlumat düyməsi var*.

Pəncərənin müştəri sahəsinin eni və hündürlüyü, uyğun olaraq, `Integer` tipli `ClientWidth` və `ClientHeight` xassələri ilə müəyyən edilir.

Misal. Pəncərənin müştəri sahəsinin ölçülərinin təyini.

```
Procedure TForm1.FormCreate(Sender:TObject);
```

```
begin
  Form1.Caption:= 'müştəri sahəsinin eni'+
   IntToStr(Form1.ClientWidth )+' uzunluğu'+
   IntToStr(Form1.ClientHeight);
end;
```

Formanın ölçülərini dəyişdirdikdə sərlövhədə onun müştəri sahəsinin ölçüləri təsvir olunacaqdır.

Formanın sərlövhə sətrinin sol tərəfində piktoqram (sistem menyusu düyməsi) yerləşir. Bu piktoqram Obyektlər inspektorunda və ya proqramda `TIcon` tipli `Icon` xassəsinə qiymət vermək yolu ilə dəyişdirilə bilər.

Ekranda formanın yeri və ölçüsü `TPosition` tipli `Position` xassəsi ilə müəyyən edilir. Bu xassə aşağıdakı qiymətləri ala bilər:

`poDesigned` –*forma layihələndirmə zamanı yerləşdirildiyi mövqedə və ölçülərində qalır*. Formanın vəziyyəti və ölçüləri `Left`, `Top`, `Width` və `Height` xassələri ilə müəyyənləşdirilir;

`poScreenCenter` –*forma ekranın mərkəzində yerləşir, onun eni və hündürlüyü* (`Width` və `Height`) *dəyişmir*;

`poDefault` –*Windows pəncərənin başlanğıc mövqeyi və ölçülərini avtomatik müəyyən edir, proqramçı bu parametrlərə nəzarət edə bilmir*;

`poDefaultPosOnly` –*Windows formanın başlanğıc vəziyyətini müəyyən edir, onun ölçüləri dəyişmir*;

`poDefaultSizeOnly`–*Windows formanın hündürlüyü və eninin başlanğıc qiymətlərini müəyyən edir, layihələndirmə zamanı isə onu göstərilən mövqedə yerləşdirir*.

Formanın təsviri `TWindowState` tipli `WindowState` xassəsi ilə müəyyən edilir və bu xassə aşağıdakı qiymətlərdən birini ala bilər:

`wsNormal` –*adi vəziyyət (susmaya görə)*;
`wsMinimized` –*bükülmüş vəziyyət*;
`wsMaximized` –*ekran boyu açılmış vəziyyət*.

Forma konteyner olduğu üçün, onun üzərində digər idarəedici elementlər yerləşir. Bu elementlərdən hansının fokus almasını `TWinControl` tipli `ActiveControl` xassəsi ilə və ya `SetFocus` metodu ilə müəyyən etmək olar, məsələn:

```
Form1.ActiveControl:= Edit1;
```

və ya

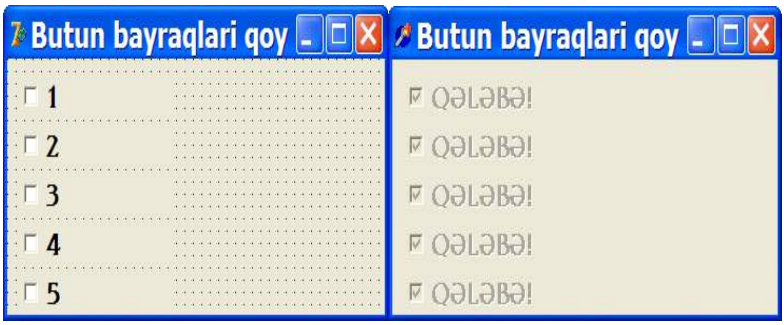
```
Edit1.SetFocus;
```

Bütün konteynerlər kimi, formanın da `Controls` xassəsi vardır ki, bu xassə konteynerdə yerləşən idarəetmə elementlərindən ibarət massivdir.

Misal. Bayraqlardan ibarət əyləncəli məsələ.

Proqram pəncərəsində beş bayraq vardır. Bütün bayraqlar atılmışdır. Məsələ bu beş bayrağın hamısını qoymaqdan ibarətdir. Lakin, bu bayraqları qoymaq o qədər də asan deyildir, çünki bir bayraq qoyulduqda digər iki bayrağın vəziyyəti dəyişir: qoyulmuş bayraqlar atılır, atılmış bayraqlar qoyulur. Oyunçu yalnız atılmış bayraqlar üzərində mausun düyməsini basa bilər. Formanın layihəsi və hazır proqramın pəncərəsi şəkil 13.1 – də göstərilmişdir.

Forma üzərində bir `CheckBox` komponenti yerləşdirin. Bayrağın sayını beşə çatdırmaq üçün mübadilə buferindən istifadə edək. *Ctrl+C* klavişlərini basdıqdan sonra, 4 dəfə *Ctrl+V* klavişlərini basın. Hər bir bayrağın sərlövhəsini (`Caption`) **1, 2, 3, 4, 5** adlandırın. Bayraqları forma üzərində düzləndirin.



Şəkil 13.1. Bayraqlardan ibarət əyləncəli məsələ

Məntiqi tipli `s` qlobal dəyişəni götürək və adi halda ona *False* (heç bir bayraq qoyulmamışdır) qiyməti verək, bayraq qoyulma hadisəsi emal olunduqda isə ona *True* qiyməti verəcəyik. Yunitin `Var` bölməsində

```
s:Boolean;
```

kodu və forma üçün `OnCreate` proseduru yaradaraq oraya cəmi bir sətir –

```
s:=False;
```

yazın. Bayraq qoyulduqda `OnClick` hadisəsi baş verir. Ona görə də `CheckBox1Click` prosedurunu yaradaq. Əsas çətinlik bayrağın öz vəziyyətini dəyişməsini müəyyən etməkdən ibarətdir. Bayraqları isə massivin elementləri kimi qəbul edək. Bayraqlar forma üzərində yerləşir, ona görə də forma onlar üçün konteynerdir. Konteynerlərin isə `Controls` (idarəetmə elementləri) xassəsi mövcuddur. Beləliklə, bayraqları biz `Controls[i]` (`i=0÷4`) kimi seçə bilərik. Ona görə də biz bayraqlara `Form1.Controls[i] as TCheckBox` kimi müraciət edəcəyik. Çünki, biz bir `Sender` obyekti ilə yox, obyektlər massivi `Controls[i]` ilə işlədiyimizə görə, onların xassələrinə müraciət etmək üçün, bu obyektlərin tipini `as` operatoru ilə aşkar müəyyən etməliyik. Məhz hansı obyekt üzərində

mausun düyməsinin basılmasını `Sender` obyekti müəyyən edir. Ona görə də hansı bayrağın dəyişdirildiyini dövr daxilində müəyyən edə bilərik:

```
for Index:=0 to 4 do
if Sender= Controls[Index] then Break;
```

`Break` operatoru dövrü dayandırır və bu zaman sərbəst seçilmiş `Index` dəyişəninin qiyməti dəyişdirilmiş bayrağa uyğun dəyişdiricinin nömrəsini müəyyən edir. İndi isə aydınlaşdırmaq lazımdır ki, bu dəyişdiricidə bayraq qoyulmuş və ya atılmışdır:

```
if not(Controls[Index] as TcheckBox).Checked then
```

Əgər bayraq indi atılmışdırsa, deməli, mausun düyməsini basana qədər o var idi. Bu halda bayrağın atılmasını ləğv etməliyik, yəni

```
(Controls[Index] as TcheckBox).Checked:=True
else
begin
```

yazmalıyıq.

Bundan sonra, digər bayraqların vəziyyətinin dəyişdirilməsini proqramlaşdırmaq lazımdır. Yəni, iki növbəti bayraq dəyişdirilməlidir. Onların nömrəsini müəyyən edək, bunun üçün növbəti bayrağı seçək:

```
num:=Index+1;
```

`Index=4` olduqda (sonuncu bayraq) biz massivin indeksinin dəyişmə diapazonunu aşacağıq. Bunun qarşısını almaq üçün sıfrıncı bayrağın vəziyyətini dəyişək:

```
if Index=4 then num:=0;
```

İndi isə bayrağın vəziyyətini əks vəziyyətə çevirə bilərik:

```
(Controls[num] as TcheckBox).Checked:=
not(Controls[num] as TcheckBox).Checked;
```

Eyni qayda ilə daha bir bayrağın vəziyyətini dəyişə bilərik:

```
num:=Index+1;
if Index=3 then num:=0;
Controls[num] as TcheckBox).Checked
not (Controls[num] as TcheckBox).Checked;
```

İndi isə oyunçunun qalib gəlmək mərhələsi proqramlaşdırılmalıdır.

Beləliklə, bu əyləncəli məsələnin proqramının tam mətni belə olacaqdır:

```
unit Unit1;

interface

uses
   Windows,Messages,SysUtils,Classes,Graphics,
   Controls,Forms,Dialogs,StdCtrls;

type
```

```
    TForm1 = class(TForm)
      CheckBox1: TCheckBox;
      CheckBox2: TCheckBox;
      CheckBox3: TCheckBox;
      CheckBox4: TCheckBox;
      CheckBox5: TCheckBox;
      procedure FormActivate(Sender: TObject);
      procedure CheckBox1Click(Sender: TObject);

    private
      { Private declarations }

    public
      { Public declarations }

    end;

var
    Form1: TForm1;
    s:Boolean;

implementation

{$R *.DFM}

procedure TForm1.FormActivate(Sender: TObject);
begin
  s:=False;
end;


procedure TForm1.CheckBox1Click(Sender: TObject);

Var
    Index,i,num: integer;
    e: Boolean;

begin
  if s then Exit;
  s:= True;
  for Index:=0 to 4 do
  if Sender= Controls[Index] then Break;
  if not(Controls[Index] as TcheckBox).Checked
  then (Controls[Index] as TcheckBox).Checked:=True
  else
    begin
      num:= Index+1;
      if Index= 4 then num:=0;
      (Controls[num] as TcheckBox).Checked:=
      not(Controls[num] as TcheckBox).Checked;
      num:= num+1;
      if Index=3 then num:=0;
      (Controls[num] as TcheckBox).Checked:=
      not (Controls[num] as TcheckBox).Checked;
```

```
    end;
  e:= True;
  for i:=0 to 4 do
  e:= e and(Controls[i] as TcheckBox).Checked;
  if e then
  for i:=0 to 4 do
    begin
      (Controls[i] as TcheckBox).Caption:='QƏLƏBƏ!';
      (Controls[i] as TcheckBox).Enabled:=False;
    end;
  s:=False;
end;

end.
```

Layihəni yerinə yetirməzdən əvvəl, `CheckBox1Click` prosedurunu bütün bayraqlar üçün təyin etməyi unutmayın. Bunun üçün növbə ilə, hər bir dəyişdiricini seçərək, `OnClick` hadisəsi qarşısında mausun düyməsini basmaqla, `CheckBox1Click` prosedurunu göstərin.

13.3. Çoxformalı əlavələr

Hər bir əlavə üçün *bir neçə forma* ola bilər. Bu formalardan biri *əsas forma* olaraq proqram yüklәndikdə təsvir olunur. Əsas forma (pəncərə) bağlandıqda bütün pəncərələr bağlanır və əlavənin işi dayandırılır. Layihə yaradılmağa başladıqda Delphi, avtomatik olaraq, birinci formanı əsas forma edir və onu `Form1` adlandırır. Layihə faylında bu forma birinci yaradılır, məsələn:

```
Application.Initialize;
Application.CreateForm(TForm1, Form1);
Application.CreateForm(TForm2, Form2);
Application.Run;
```

`Form1` forması birinci olaraq yaradıldığı üçün, o, əsas forma olur. Proqramla istənilən formanı əsas etmək olar, məsələn, aşağıdakı kod `Form2` formasını əsas edir:

```
Application.Initialize;
Application.CreateForm(TForm2, Form2);
Application.CreateForm(TForm1, Form1);
Application.Run;
```

Hansı formanın əsas olduğunu Delphi–nin pəncərəsindən müəyyən etmək daha asandır. Bunun üçün *Project/Options…* (*Layihə/Parametrlər…*) əmrini icra etməklə parametrlər pəncərəsini ekranda təsvir etmək lazımdır. Əsas forma `Form` səhifəsinin *Main Form* açılan siyahısından seçilir. Bundan sonra, Delphi, avtomatik olaraq, layihə faylına müvafiq dəyişikliklər edir.

Formalar modal və qeyri–modal formalara bölünür. *Qeyri–modal* forma özünü bağlamadan başqa formaya keçməyə imkan verir. *Modal* forma isə başqa formaya keçməzdən əvvəl, hökmən özünün bağlanmasını tələb edir.

Üzərində hər hansı məlumatı təsvir edən və ya istifadəçidən hər hansı məlumatı daxil etməyi tələb edən forma *dialoq* forması adlanır. Dialoqlar da öz növbəsində modal və qeyri–modal olur. Windows sistemində əsasən iki növ əlavə olur:

- *Birsənədli* və ya *SDI* (*Single Document Interface – birsənədli interfeys*) əlavələr;
- *Çoxsənədli* və ya *MDI* (*Multiple Document Interface – çoxsənədli interfeys*) əlavələr.

Cari anda birsənədli əlavə yalnız bir sənədlə (obyektlə) işləyə bilər. *Birsənədli* əlavəyə *Notepad* mətn redaktoru və *Paint* qrafik redaktoru misal ola bilər. Birsənədli əlavə çoxlu pəncərələrdən ibarət ola bilər, lakin, o, yalnız bir sənədi emal edə bilər. Birsənədli əlavəyə ən xarakterik misal elə Delphi inteqrallaşdırılmış mühiti ola bilər ki, burada ən azı dörd pəncərə həmişə ekranda təsvir olunur. Bu əlavələrin üstün cəhəti ondan ibarətdir ki, bu proqramlar yaddaşda az yer tutur, daha böyük sürətlə icra olunur. Bununla yanaşı, *SDI*–əlavələrdə heç bir pəncərə vizual olaraq özündə başqa pəncərəni əks etdirmir, ona görə də hansı formanın əsas olduğu bilinmir. Bundan başqa, bu proqramlar daha mürəkkəb koda malik olur.

Çoxsənədli əlavələrdə isə əsas pəncərənin hüdudlarında yerləşən törəmə pəncərələr olur. Bu tip əlavələr adətən mürəkkəb mətn və qrafik prosedurların yaradılması üçün istifadə olunur. Çoxsənədli əlavələrin üstünlüyü onların əyaniliyində və proqramların daha asan dərk olunmasındadır. Lakin, *MDI* əlavələr böyük yaddaş həcmi və böyük sürətə malik kompüterlər tələb edir. Ümumiyyətlə, bu və ya digər əlavələrin seçilməsi, həll ediləcək məsələnin xüsusiyyətindən və proqramçının özünün bu məsələyə yanaşmasından asılıdır.

Çoxsənədli forma bir əsas formadan və onun daxilində yerləşən bir neçə törəmə pəncərələrdən ibarət olur. Əsas və törəmə pəncərələrin görkəmi `FormStyle` xassəsi ilə təyin olunur. Yalnız əsas forma üçün `FormStyle` xassəsinə `fsMDIForm` qiyməti, törəmə pəncərələr üçün isə `fsMDIChild` qiyməti verilməlidir. Bu qiymətlər Obyektlər inspektorundan və ya proqramdan təyin edilə bilər.

Əsas formada adətən yazı və düymə kimi idarəedici elementlər olmur. Əgər əsas formada belə element, məsələn, `Button` düyməsi olarsa, o, pəncərənin üzərini örtən törəmə pəncərədən görünəcəkdir. Əsas pəncərə adətən menyu sətri, alətlər paneli və vəziyyətlər sətrindən ibarət olur. Qalan hissə isə müştəri sahəsi üçün ayrılır.

Törəmə pəncərələrdə isə menyu sətri, alətlər paneli və vəziyyətlər sətri olmur. Əgər törəmə pəncərədə `MainMenu` komponenti yerləşdirilərsə, forma yaradıldıqda həmin menyu əsas formanın menyusu ilə birləşəcəkdir. Belə

birləşmə avtomatik olaraq yerinə yetirildiyi üçün, hər iki menyu üçün `AutoMerge` xassəsinə *False* qiyməti verilməlidir.

Layihənin yaradılması mərhələsində avtomatik olaraq əsas və bir törəmə pəncərə formaları sinfi təsvir olunur. Layihə yerinə yetirildikdən sonra, susmaya görə, hər iki pəncərə ekranda təsvir olunur. Yerdə qalan törəmə pəncərələr (ikinci, üçüncü və s.) dinamik olaraq proqram yolu ilə yaradılmalıdır. Əgər birinci törəmə pəncərənin ekranda avtomatik olaraq təsvir olunması tələb edilmirsə, onda layihə faylından müvafiq operatoru pozmaq lazımdır. Bunu iki yolla yerinə yetirmək olar: layihə faylından əl ilə sətri pozmaq və ya daha asan olmaq üçün, layihə parametrləri pəncərəsində (*Project/Options… – Layihə/Parametrlər…* əmri ilə çağrılır) törəmə pəncərəni avtomatik yaradılan formalar siyahısından çıxarıb, müraciət olunacaq formalar siyahısında yerləşdirmək lazımdır.

Proqram yolu ilə formanın nüsxələri `Create` metodu ilə yaradılır, məsələn,

```
Form3:= TForm3.Create(Application);
```

Qeyd edək ki, baxmayaraq ki, törəmə forma əsas formanın hüdudlarında yerləşir, onun sahibi əsas forma yox, yaradılan əlavə olur. Ona görə də `Create` metodunda parametr kimi `Application` qlobal obyekti istifadə edilir. Proqramın işi başa çatdıqda bağlanmamış törəmə forma əlavənin digər formaları ilə birlikdə məhv edilir. Unutmayın ki, `Create` metodu ilə yaradılan forma `Free` metodu ilə məhv edilməlidir (`Form3.Free;`).

Layihə yerinə yetirildikdə pəncərə ekranda görünə və ya görünməyə bilər. Pəncərələrin yaradılması və pozulması, görünməsi və gizlədilməsi və habelə onların qarşılıqlı əlaqələri proqramçı tərəfindən idarə olunur. Bu məqsədlə müvafiq xassə, metod və hadisələr istifadə edilir.

Formanı idarə etmək üçün `Application` (*əlavə* üçün) və `Screen` (*ekran* üçün) qlobal obyektləri də istifadə olunur.

Formanın *görünməsi* `Visible` xassəsi ilə müəyyən edilir. Yalnız əsas forma üçün bu xassənin qiyməti *True*, digər formalar üçün isə *False* olur. Formanı yaratdıqdan sonra, onun görünməsi üçün, proqramçı kod vasitəsilə `Visible` xassəsinin qiymətini dəyişməlidir.

Misal. Formanın ekranda təsvir edilməsi.

```
Procedure TForm1.Button1Click(Sender:Tobject);

Begin

  Form3. Visible:= True;
  // gizlətdikdə isə Form 3.Visible:= False; yazılmalıdır

end;
```

Formanı ekranda təsvir etməyin və ya gizlətməyin digər üsulu `Show` və `Hide` metodlarının tətbiqindən ibarətdir. `Show` metodu ilə forma qeyri–modal rejimdə təsvir olunur.

Misal. Formanın ekranda təsvir edilməsi.

```
Procedure TForm1.Button1Click(Sender:TObject);
Begin
  Form3.Show;   // gizlətdikdə isə Form3.Hide; yazılmalıdır

end;
```

Formaları modal rejimdə təsvir etmək üçün isə `ShowModal` metodu tətbiq olunur (`Form3.ShowModal`). `Show` metodundan fərqli olaraq, `ShowModal` funksiyası `ModalResult` xassəsinin qiymətini müəyyən edir. Bu xassənin ala biləcəyi qiymətlərlə düymələri öyrəndikdə tanış olduq.

Çoxsənədli əlavələrin törəmə formaları yaradıldıqdan dərhal sonra ekranda təsvir olunur.

Yeni forma yaradıldıqdan sonra, onun adı – `Form3` sonuncu yaradılmış formanın nüsxəsini göstərir. Bütün törəmə pəncərələrə müraciət etmək üçün əlavənin əsas formasının `TForm` tipli `MDIChildren[I:integer]` xassəsini tətbiq etmək lazımdır. Bu xassə çoxsənədli əlavənin törəmə formalarından ibarət massivdir. Törəmə pəncərələrin nömrələnməsi sıfırdan başlayır. Üstdə duran pəncərənin sıra nömrəsi sıfır olur. Çoxsənədli əlavələrdə pəncərələrin sayını tam tipli `MDIChldCount` xassəsi təyin edir. Aktiv törəmə pəncərəyə müraciət etmək üçün `TForm` tipli `ActiveMDIChild` xassəsindən istifadə edilir. Bu xassə hansı törəmə pəncərənin fokus almasını müəyyən edir. Əgər əlavə çoxsənədli deyilsə, bu xassənin tətbiqi səhvə gətirəcəkdir.

`MDIChildren`, `MDIChildCount` və `ActiveMDIChild` xassələri proqram icra olunanda qiymət alır, onlara qiymət vermək olmaz.

Misal. Törəmə pəncərələrə müraciət.

Bir neçə törəmə pəncərə yaradaraq (*Project/Options…*) aşağıdakı proseduru icra edin.

```
Procedure TForm1.Button1Click(Sender:TObject);
Var n:integer;
Begin

  for n:=0 to Form1.MDIChildCount-1 do
  Form1.MDIChldren[n].Caption:=
        'Pəncərə N  ' +IntToStr(n);

end;
```

Törəmə formaların sərlövhələrində onların sıra nömrəsi təsvir olunacaqdır. Bunu `Form1` əsas formasının `MDIChldren` xassəsi yerinə yetirir.

Misal. Aktiv törəmə pəncərənin bağlanması.

```
Procedure TForm1.Button1Click(Sender:TObject);
Begin
  If Form1.ActiveMDIChild<>nil then
  Form1.ActiveMDIChild.Close;
end;
```

Pəncərə bağlanmazdan əvvəl onun mövcud olması yoxlanır.

Əsas forma bağlandıqda, onunla bərabər, bütün törəmə pəncərələr bağlanır və əlavənin işi dayandırılır. Törəmə pəncərəni isə onun sərlövhəsindəki bağlamaq düyməsini, ***Ctrl+F4*** klavişlərini basmaqla və ya törəmə forma üçün `Close` metodunu çağırmaqla bağlamaq mümkün olmur. Susmaya görə, `Action` parametr–əlaməti `caMinimize` qiyməti aldığı üçün, törəmə pəncərəni sadalanan bu üsullarla bağlamağa cəhd etdikdə, o bağlanmır, bükülür. Pəncərəni adi qaydada bağlamaq üçün törəmə pəncərə üçün `OnClose` hadisə emaledicisi yaratmaq lazımdır:

```
Procedure TForm3.FormClose(Sender:TObject;
              Var Action:TClose Action);
begin
  Action:= caFree;
end;
```

Burada, `TForm3` törəmə pəncərə sinfidir.

Redaktor kimi istifadə olunan törəmə pəncərəni `OnClose` metodu ilə bağladıqda redaktorda mətnin yadda saxlanmasını yoxlamaq lazımdır. Bu halda `OnClose` hadisə emaledicisi belə ola bilər:

```
Procedure TForm3.FormClose(Sender:TObject;
              Var Action:TClose Action);
Begin
  if Memo1.Modified then
  if MessageDlg('fayl yadda saxlanılmamışdır!'+
           #13#10+'Çıxmaq istəyirsinizmi?'+
           mnInformation,[mbYes,mbNo],0)=mrYes
  then
  Action:= caFree;
  else Action:= caNone;
end;
```

Törəmə pəncərəni `Hide` metodu ilə bağlamaq olmaz, bu halda müstəsna hal (səhv) baş verəcəkdir.

Törəmə pəncərələri ekranda *kaskad* və ya *mozaika* variantında yerləşdirmək üçün, uyğun olaraq, `Cascade` və `Title` metodları tətbiq olunur.

Mozaika variantında pəncərələrin yerləşdirilməsi `TTileMode` tipli `TileMode` xassəsi ilə idarə olunur. Bu xassə iki qiymət ala bilər:

`tbHorizontal` – *törəmə pəncərələr valideyn formanın bütün müştəri sahəsinin eni boyu yerləşir (susmaya görə)*;

`tbVertical` – *törəmə pəncərələr valideyn formanın bütün müştəri sahəsinin hündürlüyü boyu yerləşir*.

13.3.1. Çoxsənədli əlavələrə aid misallar

Misal. Cəbri tənliklərin təqribi həlli.

Çoxformalı əlavələri tətbiq etməklə $f(x)=0$ tənliyinin təqribi üsulla həllini proqramlaşdıraq. Əlavə üç formadan ibarət olacaqdır: əsas forma menyu sətrindən ibarət olacaq, digər iki törəmə formalarda isə uyğun olaraq *Nyuton* və *iterasiya üsulları* ilə tənliyin həllinin nəticələri təsvir olunacaqdır. Xatırladaq ki, *Nyuton* üsulunun rekurrent tənliyi

$$x_{n+1} = x_n - \frac{f(x_n)}{f^1(x_n)},$$

iterasiya üsulunun rekurrent tənliyi isə

$$x_{n+1} = f(x_n)$$

şəklindədir. Burada, *n* – iterasiyanın nömrəsidir. Proqramda konkret olaraq $x^4-3x^2-4x-1=0$ tənliyi həll edilmiş, başlanğıc yaxınlaşma kimi $x_0=1$, dəqiqlik isə *Eps = 0.0004* qəbul edilmişdir.

Obyektlər inspektorunda `Form1` formasının `Caption` xassəsinə `Ədədi üsullar`, `Name` xassəsinə `fmUsul`, `FormStyle` xassəsinə isə `fsMDIForm` qiymətləri təyin edin. Forma üzərində yeganə komponent – `MainMenu` yerləşdirin. Menyu sətri üç menyudan və onların daxilində bəndlərdən (əmrlərdən) ibarət olacaqdır. Bu menyular aşağıdakılardır:

`Fayl (mnuFile)`
- `Bağlamaq (mnuClose)`
- `Hamısını bağlamaq (mnuCloseAll)`
- `Çıxış (mnuExit)`

`Metod (mnuUsul)`
- `Nyuton (mnuNyuton)`
- `İterasiya (mnuIterasiya)`

`Pəncərə (mnuWindow)`
- `Kaskad (mnuCascade)`
- `Mozaika (mniTile)`.

Bu menyular isə belə yaradılır. MainMenu komponenti üzərində mausun düyməsini iki dəfə basmaqla və ya mausun sağ düyməsini basaraq, *Menu Designer...* əmrini icra etməklə *Menyu konstruktorunu* çağırın. Obyektlər inspektorunda Caption xassəsi qarşısında – Fayl, Name xassəsi qarşısında isə mnuFile yazıb *Enter* klavişini basın. Bununla da ilk menyu yaranacaqdır. Növbəti menyu və ya menyu bəndlərini yaratmaq üçün lazım olan yerlərdə mausun düyməsini basaraq, Obyektlər inspektorunda Caption və Name xassələrinə müvafiq adlar verin (menyuların yuxarıdakı strukturunda menyunun adı Caption xassəsinə, mötərizə daxilində yazılmış adlar isə Name xassəsinə aiddir). Fayl menyusunda Çıxış bəndini digərlərindən fərqləndirmək üçün Hamısını bağlamaq menyu bəndindən sonrakı menyu üçün Caption xassəsinə "–" işarəsi daxil edərək *Enter* klavişini basın. Bu bənd düz xətdən ibarət olacaqdır. Analoji qayda ilə bütün menyuları və menyu bəndlərini yaradıb, *Menyu konstruktorunu* bağlayın.

İkinci formanı yaradaq. Bunun üçün fayl menyusundan *File/New/Form* əmrini icra etmək lazımdır. Delphi Sizə Form2 adlı yeni boş forma təklif edəcəkdir. Bu formanın Caption xassəsinə – Nyuton üsulu, Name xassəsinə – fmNyuton, FormStyle xassəsinə isə fsMDIChild qiymətləri təyin edin. Forma üzərinə ListBox komponenti yerləşdirib onun sərlövhəsini pozun. Bu komponent bizə hər bir iterasiyanın nəticələrini əks etdirmək üçün lazımdır. ListBox komponentinin yanında Label komponenti yerləşdirərək, onun sərlövhəsini Tənliyin kökü: adlandırın. Bu komponent bizə tənliyin həllini təsvir etmək üçün lazım olacaqdır.

Yuxarıdakı qayda ilə Form3 forması yaradaraq, onun sərlövhəsini İterasiya üsulu adlandırın, Name xassəsinə – fmIter, FormStyle xassəsinə isə fsMDIChild qiymətləri təyin edin. Bu forma üzərində ListBox və Label komponentləri yerləşdirərək müvafiq əməliyyatları təkrar edin.

File/Save As... əmrini icra etməklə layihəni aşağıdakı adlarla yadda saxlayın: Unit1 – uUsul, Unit2 – uNyuton, Unit3 – uIter və Project1 – MDIUsul.

Hər bir menyu bəndinin yerinə yetirəcəyi funksiyalara uyğun prosedurlar yaratmaq üçün həmin bəndi seçmək kifayətdir: bu halda yunit ön plana keçərək müvafiq prosedur yaradılacaq və yazılacaq kodların mövqeyi kursorla göstəriləcəkdir.

Nyuton və İterasiya bəndlərinə uyğun prosedurlarda *Nyuton* və *iterasiya* üsullarının proqramları tərtib edilmişdir, bu ənənəvi Pascal proqramından başqa bir şey deyildir, hətta xüsusi izahata ehtiyac qalmır. Lakin, uUsul modulunun Implemntation bölməsinə hökmən

```
Uses uNyuton, uIter;
```

kodu əlavə olunmalıdır. Çünki, nəticələr əsas formada yox, törəmə formalarda təsvir olunacaqdır.

`Bağlamaq` menyusu üçün tərtib olunmuş `mnuCloseClick` proseduru aktiv pəncərəni, `mnuCloseAll` proseduru (`Hamısını bağlamaq` bəndi) isə bütün törəmə pəncərələri bağlayır.

`Çıxış` menyusu üçün tərtib olunmuş `mnuExitClick` proseduru isə əlavəni bağlayır.

Törəmə formalar bağlandıqda onların məhv edilməsi üçün (yaddaşda yer tutmaması üçün), `uNyuton` və `uIter` modullarında `FormClose` proseduru yaradılaraq, `Action` parametrinə `caFree` qiyməti mənimsədilmişdir.

Törəmə pəncərələr əlavəyə məxsus olduğu üçün, onlar ekranda həmişə təsvir olunur. Onları kaskad və ya mozaika rejimlərində yerləşdirmək üçün müvafiq prosedurlar yaradılmışdır.

Yaradılmış bütün formalar layihə faylına (`MDIUsul`) avtomatik olaraq əlavə edildiyi üçün, əvvəlki layihələrdə olduğu kimi, burada da bu fayla heç bir kod yazmırıq.

Əlavənin bütün modullarının mətni aşağıda göstərilmişdir.

```
program MDIUsul;

uses
   Forms,
   uUsul in 'uUsul.pas' {fmUsul},
   uNyuton in 'uNyuton.pas' {fmNyuton},
   uIter in 'uIter.pas' {fmIter};
{$R *.RES}

begin
  Application.Initialize;
  Application.CreateForm(TfmUsul, fmUsul);
  Application.CreateForm(TfmNyuton, fmNyuton);
  Application.CreateForm(TfmIter, fmIter);
  Application.Run;
end.


unit uUsul;

interface

uses
     Windows,Messages,SysUtils,Classes,Graphics,
     Controls,Forms,Dialogs,Menus;
type
  TfmUsul = class(TForm)
    MainMenu1: TMainMenu;
    N1: TMenuItem;
    mnuClose: TMenuItem;
    mnuCloseAll: TMenuItem;
    mnuExit: TMenuItem;
    N5: TMenuItem;
    mnuNyuton: TMenuItem;
```

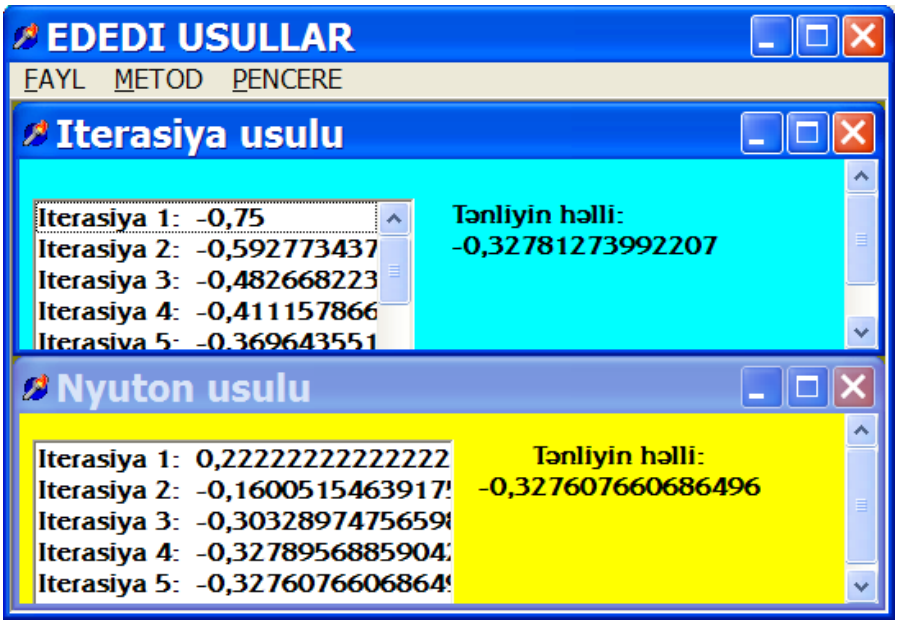
```
    mnuIterasiya: TMenuItem;
    N6: TMenuItem;
    N7: TMenuItem;
    mnuCascade: TMenuItem;
    mnuTitle: TMenuItem;
    procedure mnuNyutonClick(Sender: TObject);
    procedure mnuIterasiyaClick(Sender: TObject);
    procedure mnuCloseClick(Sender: TObject);
    procedure mnuCloseAllClick(Sender: TObject);
    procedure mnuExitClick(Sender: TObject);
    procedure mnuCascadeClick(Sender: TObject);
    procedure mnuTitleClick(Sender: TObject);

  private
    { Private declarations }

  public
    { Public declarations }
  end;

var
   fmUsul: TfmUsul;

implementation

uses uNyuton, uIter;

{$R *.DFM}

procedure TfmUsul.mnuNyutonClick(Sender: TObject);
Const x0=1; Eps=0.0004;
Var
   x,x1:Real;
   k:integer;
begin
  fmNyuton.ListBox1.Clear;
  x1:=x0;
  k:=0;
  Repeat
  x:=x1; k:=k+1;
  x1:=x-(sqr(x)*sqr(x)-3*sqr(x)-4*x-1)/
                          (x*x*x-6*x-4);
  fmNyuton.ListBox1.Items.Add('İterasiya '+
            IntToStr(k)+':'+FloatToStr(x1));
  Until abs(x-x1)<=Eps;
  fmNyuton.Label1.Caption:='Tənliyin həlli:'
                     +#13#10+FloatToStr(x1);
end;


procedure TfmUsul.mnuIterasiyaClick(Sender: TObject);
Const x0=1; Eps=0.0004;
Var
```

```
   x,x1:Real;
   k:Integer;
begin
  fmIter.ListBox1.Clear;
  x1:=x0;
  k:=0;
  Repeat
  x:= x1; k:= k+1;
  x1:= (sqr(x)*sqr(x)-3*sqr(x)-1)/4;
  fmIter.ListBox1.Items.Add('İterasiya '+
         IntToStr(k)+':  '+FloatToStr(x1));
  Until abs(x-x1)<= Eps;
  fmIter.Label1.Caption:= 'Tənliyin həlli:'+
                      #13#10+FloatToStr(x1);
end;


procedure TfmUsul.mnuCloseClick(Sender: TObject);
begin
  if fmUsul.ActiveMDIChild <> nil then
  fmUsul.ActiveMDIChild.Close;
end;


procedure TfmUsul.mnuCloseAllClick(Sender: TObject);
Var
   n:Integer;
begin
  for n:=1 to fmUsul.MDIChildCount-1 do
  if fmUsul.ActiveMDIChild<>nil then
     fmUsul.MDIChildren[n].Close;
end;


procedure TfmUsul.mnuExitClick(Sender: TObject);
begin
  Close;
end;


procedure TfmUsul.mnuCascadeClick(Sender: TObject);
begin
  fmUsul.Cascade;
end;


procedure TfmUsul.mnuTitleClick(Sender: TObject);
begin
  fmUsul.Tile;
end;
end.
```

```
unit uIter;

interface

uses
   Windows,Messages,SysUtils,Classes,Graphics,
   Controls,Forms,Dialogs,StdCtrls;

type
 TfmIter = class(TForm)
   ListBox1: TListBox;
   Label1: TLabel;
   procedure FormClose(Sender: TObject;
             var Action: TCloseAction);

 private
   { Private declarations }

 public
   { Public declarations }

 end;

var
   fmIter: TfmIter;

implementation

{$R *.DFM}


procedure TfmIter.FormClose(Sender: TObject;
            var Action: TCloseAction);
begin
 Action:=caFree;
end;

end.


unit uNyuton;

interface

uses
   Windows,Messages,SysUtils,Classes,Graphics,
   Controls,Forms,Dialogs,StdCtrls;

type
 TfmNyuton = class(TForm)
   ListBox1: TListBox;
   Label1: TLabel;
   procedure FormClose(Sender: TObject;
            var Action: TCloseAction);
```

```
  private
    { Private declarations }

  public
    { Public declarations }

  end;

var
   fmNyuton: TfmNyuton;

implementation


{$R *.DFM}

procedure TfmNyuton.FormClose(Sender: TObject;
          var Action: TCloseAction);
begin
  Action:=caFree;
end;
end.
```

F9 klavişini basaraq layihəni icra edin. Əgər proqramda səhvlər olmazsa, Siz, şəkil 13.2 – də göstərilmiş əlavəni alacaqsınız.



Şəkil 13.2. Ədədi üsullardan ibarət çoxformalı əlavə

Misal. Mətn və qrafik fayllara baxış.

Bu layihə də üç formadan ibarət olacaqdır: əsas forma yenə menyulardan ibarət olduğu halda, törəmə formaların birində mətnləri, digərində isə şəkilləri təsvir edəcəyik. Şəkillərin təsvir ediləcəyi formada kontekst menyu da yaradacağıq ki, bu menyunun əmrləri icra olunduqda dahi Azərbaycan şairi Nizami Gəncəvinin və dahi bəstəkarımız Üzeyir Hacıbəyovun abidələri əks olunacaqdır.

Əlavəyə daxil olan formaları belə müəyyənləşdirin:

```
Form1:
   Caption    –Fayllara baxış;
   Name       –fmFile;
   FormStyle  –fsMDIForm;

Form2:
   Caption    –Mətn;
   Name       –fmText;
   FormStyle  –fsMDIChild;

Form3:
   Caption    –Şəkil;
   Name       –fmPict;
   FormStyle  –fsMDIChild.
```

`Fayllara baxış` adlandırdığımız əsas forma üzərində üç komponent – `MainMenu`, `OpenDialog` və `OpenPictureDialog` komponentləri yerləşdirin. Əlavə hazır olduqda bu komponentlərin heç biri görünməyəcəkdir. Bu formada menyuların strukturu belə olacaqdır:

```
Fayl (mnuFile)
  • Mətn açmaq (mnuOpenText)
  • Qrafik açmaq (mnu Graphic)
     • Bütün şəkillər (mnuOpenPicture)
     • Nizami (mnuOpenNizami)
     • Üzeyir (mnuOpenUzeyir)
  • Çıxış (mnuExit1)

Bağlamaq (mnuClose)
  • Kaskad (mnuCascade)
  • Mozaika (mnuTile)
  • Çıxış (mnuExit2)
```

Bu menyuların necə yaradıldığını Siz artıq bilirsiniz. Lakin, `Fayl` menyusunda `Qrafik açmaq` menyu bəndi alt menyulardan ibarətdir. Bu bəndə daxil olan menyuları yaratmaq üçün bəndin özünü yaratdıqdan sonra, ya

***Ctrl*+→** klavişlərini basmaq, ya da bu bənd üzərində mausun sağ düyməsini basaraq, kontekst menyudan *Create Submenu* əmrini icra etmək lazımdır.

`OpenDialog` və `OpenPictureDialog` komponentlərinin xassələrini belə müəyyənləşdirin:

`OpenDialoq:`

`Title` – *Mətnlərin seçilməsi* – mətn fayllarını açmaq üçün ekranda təsvir olunan dialoq pəncərəsinin sərlövhəsi;

`Filter` – *Mətn faylları*[*.*txt, *.doc*]| *.*txt, *.doc* – həmin dialoq pəncərəsində yalnız *.*txt* və *.*doc* tipli mətn faylları təsvir olunacaqdır;

`Options` xassəsinin `ofFileMustExist` alt xassəsinə *True* qiyməti verin; bu, fayl tapılmadıqda baş verəcək anlaşılmazlığın qarşısını alacaq.

`OpenPictureDialog:`

`Title` –*Şəkillərin seçilməsi*;
`Filter` –*Rastr təsvirlər*[*.*bmp*]| *.*bmp*;
`DefaultExt` –*.*bmp* – susmaya görə seçilən fayllar;
`Options.ofFileMustExist`–*True.*

`Mətn` adlı forma üzərində bir `Memo1` komponenti yerləşdirməklə, onun aşağıdakı xassələrini təyin edin:

`Memo1` –sözünü pozun;
`Align` –`alClient`;
`ScrollBar` –`ssBoth` (hər iki fırlatma zolağının qoyulması).

`Şəkil` adlı üçüncü forma üzərində `PopupMenu` komponenti, `Panel` komponenti, onun üzərində isə `Label` və `Image` komponentləri yerləşdirin. Bu komponentlərin aşağıdakı xassələrini təyin edin:

`Panel1:`

`Caption` –sərlövhəni pozun;
`BevelInner` –`bvNone` – daxili faska yoxdur;
`BevelOuter` –`bvLowered` – xarici faska yoxdur;
`BevelWidth` –*2* – faskanın eni *2* piksel müəyyən edilir;
`Align` –`alClient`.

`Image1:`

`Align` –`alRight`;
`AutoSize` –*True* –`Image` komponenti öz ölçüsünü şəklin ölçüsünə müvafiq olaraq dəyişdirir, şəklin keyfiyyəti pisləşmir.
`Stretch` –*False.*

Label1:

Caption –Abidə;
WordWrap –*True* – mətn bir neçə sətirdə yerləşdirilir;
Transparent –*True* – mətn şəffaf olmaqla şəklin üstünü örtmür.

Kontekst menyunu yaratmaq üçün, PopupMenu komponenti üzərində mausun düyməsini iki dəfə basaraq, menyu konstruktorunu açın. Sonrakı əməliyyatlar adi menyu yaradıldıqda icra olunan əməliyyatlarla eyni olacaqdır.

Kontekst menyuları belə adlandırın:

Nizami (mnukOpenNizami);
Üzeyir (mnukOpenUzeyir).

Layihəni yadda saxladıqda proqramlara aşağıdakı adları təyin edin: Unit1 – uFile, Unit2 – uText, Unit3 – uPicture, Project1 – Pfile.

Bu proqramların tam mətnləri aşağıda göstərilmişdir.

```
program PFile;

uses
   Forms,
   uFile in 'uFile.pas' {fmFile},
   uText in 'uText.pas' {fmText},
   uPicture in 'uPicture.pas' {fmPict};

{$R *.RES}

begin
  Application.Initialize;
  Application.CreateForm(TfmFile, fmFile);
  Application.CreateForm(TfmText, fmText);
  Application.CreateForm(TfmPict, fmPict);
  Application.Run;
end.


unit uFile;

interface

uses
     Windows,Messages,SysUtils,Classes,Graphics,
     Controls,Forms,Dialogs,ExtDlgs, Menus;

type
  TfmFile = class(TForm)
    MainMenu1: TMainMenu;
    mnuFile: TMenuItem;
    mnuClose: TMenuItem;
    mnuOpenText: TMenuItem;
    mnuGraphic: TMenuItem;
```

```
    mnuOpenPicture: TMenuItem;
    mnuOpenNizami: TMenuItem;
    mnuOpenUzeyir: TMenuItem;
    mnuExit1: TMenuItem;
    mnuExit2: TMenuItem;
    mnuCascade: TMenuItem;
    mnuTile: TMenuItem;
    OpenDialog1: TOpenDialog;
    OpenPictureDialog1: TOpenPictureDialog;
    procedure mnuOpenNizamiClick(Sender: TObject);
    procedure mnuOpenUzeyirClick(Sender: TObject);
    procedure mnuExit1Click(Sender: TObject);
    procedure mnuOpenPictureClick(Sender: TObject);
    procedure mnuCascadeClick(Sender: TObject);
    procedure mnuTileClick(Sender: TObject);
    procedure mnuOpenTextClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);

  private
    { Private declarations }

  public
    { Public declarations }

  end;

var
   fmFile: TfmFile;

implementation

uses uPicture, uText;

{$R *.DFM}


procedure TfmFile.mnuOpenNizamiClick(Sender: TObject);
begin
  fmPict.Caption:= 'Nizami';
  fmPict.Image1.Picture.LoadFromFile('C:\17.bmp');
  fmPict.Label1.Caption:= 'Dahi Azərbaycan
                    şairi Nizami Gəncəvi';
end;


procedure TfmFile.mnuOpenUzeyirClick(Sender: TObject);
begin
  fmPict.Caption:= 'Uzeyir';
  fmPict.Image1.Picture.LoadFromFile('C:\21.bmp');
  fmPict.Label1.Caption:= 'Dahi Azərbaycan
                    bəstəkarı Üzeyir Hacıbəyov';
end;
```

```
procedure TfmFile.mnuExit1Click(Sender: TObject);
begin
  Close;
end;


procedure TfmFile.mnuOpenPictureClick(Sender: TObject);
begin
if OpenPictureDialog1.Execute then
  begin
    fmPict.Caption:= OpenPictureDialog1.FileName;
    fmPict.Image1.Picture.LoadFromFile(
    OpenPictureDialog1.FileName);
    fmPict.Label1.Caption:='';
  end;
end;


procedure TfmFile.mnuCascadeClick(Sender: TObject);
begin
  fmFile.Cascade;
end;


procedure TfmFile.mnuTileClick(Sender: TObject);
begin
  fmFile.Tile ;
end;


procedure TfmFile.mnuOpenTextClick(Sender: TObject);
begin
  if OpenDialog1.Execute then
  begin
    fmText.Caption:= OpenDialog1.FileName;
    fmText.Memo1.Lines.LoadfromFile(
                  OpenDialog1.FileName);
  end;
end;


procedure TfmFile.FormCreate(Sender: TObject);
begin
  OpenDialog1.Filter:='Mətn faylları[*.txt;*.doc]|
            *.txt;*.doc|Bütün fayllar[*.*]|*.*';
            fmFile.TileMode:=tbVertical;
end;

end.
```

```
unit uPicture;

interface

uses
   Windows,Messages,SysUtils,Classes,Graphics,Controls,
   Forms,Dialogs,Menus, ExtCtrls, StdCtrls;

type
 TfmPict = class(TForm)
   Panel1: TPanel;
   Image1: TImage;
   PopupMenu1: TPopupMenu;
   mnukOpenNizami: TMenuItem;
   mnukOpenUzeyir: TMenuItem;
   Label1: TLabel;
   procedure mnukOpenNizamiClick(Sender: TObject);
   procedure mnukOpenUzeyirClick(Sender: TObject);
   procedure FormClose(Sender: TObject;
             var Action: TCloseAction);
   procedure Image1MouseDown(Sender: TObject;
             Button:TMouseButton;
             Shift: TShiftState;X, Y: Integer);
   procedure FormCreate(Sender: TObject);

 private
   { Private declarations }

 public
   { Public declarations }

 end;

var
   fmPict: TfmPict;

implementation

{$R *.DFM}

procedure TfmPict.mnukOpenNizamiClick(Sender: TObject);
begin
 Caption:= 'Nizami';
 Image1.Picture.LoadFromFile('C:\17.bmp');
 Label1.Caption:=
       'Dahi Azərbaycan şairi Nizami Gəncəvi';
end;


procedure TfmPict.mnukOpenUzeyirClick(Sender: TObject);
begin
 Caption:= 'Uzeyir';
 Image1.Picture.LoadFromFile('C:\21.bmp');
```

```
  Label1.Caption:=
        'Dahi Azərbaycan bəstəkarı Üzeyir Hacıbəyov';
end;


procedure TfmPict.FormClose(Sender:TObject;
                  var Action:TCloseAction);
begin
  Action:=caFree;
end;


procedure TfmPict.Image1MouseDown(Sender: TObject;
           Button: TMouseButton;
           Shift: TShiftState; X, Y: Integer);
Var
   p:TPoint;
begin
  if Button=mbRight then
  begin
    p.x:= x;
    p.y:= y;
    p:= ClientToScreen(p);
    PopupMenu1.Popup(p.x, p.y);
  end;
end;


procedure TfmPict.FormCreate(Sender: TObject);
begin
  Panel1.PopupMenu:=PopupMenu1;
end;
end.


unit uText;

interface

uses
   Windows,Messages,SysUtils,Classes,
   Graphics,Controls,Forms,Dialogs,StdCtrls;

Type

  TfmText = class(TForm)
    Memo1: TMemo;

  private
     { Private declarations }

  public
```

```
    { Public declarations }

  end;

var
   fmText: TfmText;

implementation

{$R *.DFM}

end.
```

Layihədə yaradılmış prosedurların əksəriyyəti əvvəlki misalların fraqmentləri olduğu üçün, onların əlavə izahına ehtiyac qalmır. Kontekst menyulara uyğun prosedurlar isə ilk dəfə yaradıldığı üçün, onları izah edək. `Nizami` sərlövhəli kontekst menyuya uyğun prosedur yaratmaq üçün, Obyektlər inspektorunun birinci sətrində yerləşən açılan siyahıdan `mnukOpenNizami` seçib, *Events* səhifəsində `OnMauseDown` hadisəsi qarşısında mausun düyməsini iki dəfə basmaq lazımdır. Bu hadisə mausun ixtiyari düyməsini basdıqda baş verdiyi üçün, kontekst menyu da mausun istənilən düyməsini basdıqda baş verəcəkdir. Kontekst menyunun mausun sağ düyməsini basdıqda peyda olması üçün modula

```
if Button=mbRight then begin
```

sətri, yəni "*sağ düymə basıldıqda*" kodu əlavə edilmişdir. Kontekst menyunun digər vacib xüsusiyyəti də vardır: menyu *ekranın* sol yuxarı küncündən hesablanan ekran koordinatında peyda olur. Lokal koordinatlar isə *formanın* sol yuxarı küncündən hesablanır. Ona görə də lokal koordinat ekran koordinatına çevrilməlidir. Xoşbəxtlikdən, Delphi burada da öz köməyini bizdən əsirgəmir, belə ki, bu çevirmə üçün o, `ClientToScreen` metodu təklif edir. Bu metoda verilənlər `TPoint` strukturu ilə ötürülür ki, onun `X` və `Y` xassələri vardır. Ona görə də biz `TPoint` tipli `P` dəyişəni yaradaraq ona lokal koordinatlar verməklə, ekran koordinatlarını alacaq və kontekst menyunu açacağıq.

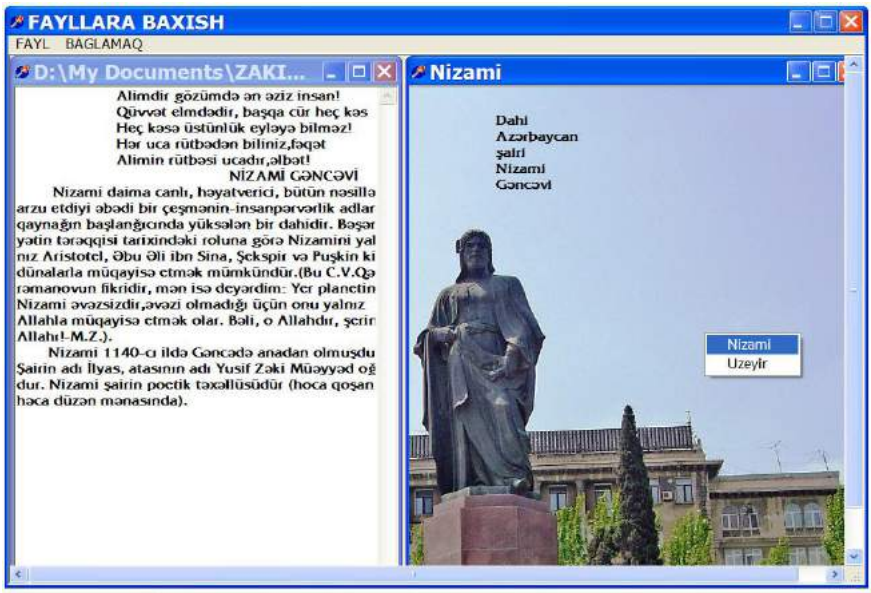
Kontekst menyunun hansı komponentə aid olması isə başlanğıc qiymətlərin təyin olunduğu `OnCraete` (forma yaradıldıqda) prosedurunda göstərilmişdir.

Onu da qeyd edək ki, `Çıxış` (`mnuExit1`) prosedurunu `mnuExit2` menyusuna təyin etmək lazımdır.

Pəncərələrin mozaika rejimində müştəri oblastının hündürlüyü boyu yerləşdirilməsi üçün, `OnCreate` prosedurunda `TileMode` metodu çağrılaraq, ona `tbVertical` qiyməti verilmişdir.

F9 klavişini basmaqla layihəni işə buraxın. Əgər proqramda heç bir səhv olmazsa, `Fayl` menyusundan `Mətn açmaq` əmrini icra etdikdə mətn fayllarından ibarət dialoq pəncərəsi, `Qrafik açmaq/Bütün şəkillər` əmrini icra etdikdə isə şəkil fayllarından ibarət dialoq pəncərəsi ekranda təsvir olunacaq, lazımi faylları seçdikdən sonra, onlar müvafiq `Mətn` və `Şəkil`

pəncərələrində yerləşəcəkdir. `Fayl/Qrafik açmaq/Nizami` əmrini icra etdikdə isə `Şəkil` pəncərəsində Nizami Gəncəvinin abidəsi və şəklin üzərində "*Dahi Azərbaycan şairi Nizami Gəncəvi"* sözləri təsvir olunacaqdır (şəkil 13.3). Şəkillər üzərində mausun sağ düyməsini basdıqda `Nizami` və `Üzeyir`



Şəkil 13.3. Mətn və şəkil pəncərələrindən ibarət əlavə

əmrlərindən ibarət kontekst menyu açılacaq və `Üzeyir` əmri seçildikdə Üzeyir Hacıbəyovun abidəsi və onun üzərində müvafiq yazı təsvir olunacaqdır.

Misal. Modal dialoq pəncərələrindən ibarət layihə.

Belə bir məsələnin həllinə zərurət törəmə pəncərə ilə modal dialoq pəncərələrinin fərqini izah etməkdən irəli gəlir. Əvvəlki iki layihədə gördük ki, törəmə pəncərələr ekranda həmişə təsvir olunur və onları bağlamadan istənilən pəncərəyə keçmək mümkündür. *Modal dialoq pəncərələri* isə onları çağırana qədər ekranda təsvir olunmur, təsvir olunduqdan sonra isə onu bağlamadan digər pəncərələrlə işləməyə icazə vermir. Modal dialoq formasının `BorderStyle` xassəsinə, sadəcə olaraq, `bsDialog` qiyməti verməklə o, *dialoq pəncərəsi olmur*, pəncərə o vaxt *modal olur* ki, o `ShowModal` metodu ilə çağrılır. Törəmə pəncərələr kimi modal pəncərələr də həm əvvəlcədən, həm də proqram yolu ilə (`TForm.Create(Application)` və ya `TForm.Create(Self)` metodları ilə) yaradıla bilər.

`Form1` formasının `Name` xassəsinə `MainForm` qiyməti daxil edin. Bu forma üzərində `Panel` komponenti yerləşdirib onun `Align` xassəsinə `alBottom` qiyməti verin: panel formanın aşağı hissəsində bütün müştəri sahəsinin enini tutacaqdır. `Panel1` adını pozun, sonra onun üzərinə iki `Button` düyməsi yerləşdirin. Düymələrin sərlövhəsini `Açmaq` və `Müəllif`

adlandırın. Forma üzərinə `Image` komponenti yerləşdirib, onun `Align` xassəsinə – `alClient`, `AutoSize` xassəsinə – *True*, `Stretch` xassəsinə – *False*, `Center` xassəsinə isə *True* qiymətləri verin.

File/New/Form əmrini icra edərək yeni boş `Form2` forması yaradın. Bu formanın `Name` xassəsinə `About` qiyməti daxil edin, `BorderStyle` xassəsi üçün `bsDialog` qiyməti, `Position` xassəsi üçün `poScreenCenter` (*ekranın mərkəzində*) qiyməti seçərək formanın enini – `Width`–*330*, hündürlüyünü isə `Height`–*130* piksel müəyyənləşdirin. Bu formanın üzərində iki `Label` komponenti yerləşdirib, `Label1` komponentini formanın yuxarı hissəsində, `Label2` komponentini isə formanın aşağı hissəsində düzləndirin. Hər iki komponentin sərlövhəsini pozun.

File/New/Form (*Fayl/Yeni forma*) əmrini icra etməklə üçüncü boş forma yaradıb, onun `Name` xassəsinə – `Settings`, `BorderStyle` xassəsinə – `bsDialog`, `Width` və `Height` xassələrinə isə uyğun olaraq *300* və *170* qiymətləri daxil edin. Bu forma üzərinə `Bevel` (faska) komponenti yerləşdirib, onun üzərində mausun sağ düyməsini basıb, *Send To Back* əmrini icra etməklə, onu aşağı çökdürün. Faskanı böyüdərək onun üzərində `ComboBox` siyahı, `CheckBox` dəyişdirici və iki `Button` düymə komponentləri yerləşdirin. `Button1` düyməsini faska üzərində yox, forma üzərində yerləşdirin. `ComboBox1` komponentinin sərlövhəsini pozun, `CheckBox1` komponentinin sərlövhəsini `Göstər`, `Button2` düyməsinin sərlövhəsini `Xülasə...`, `Button1` düyməsinin sərlövhəsini isə `OK` adlandırın. Bu forma üzərinə daha bir komponent – `OpenPictureDialog` yerləşdirərək onun `Options.ofFileMustExist` xassəsinə *True* qiyməti verin.

Bütün bu konstruksiyalaşdırma işlərindən sonra, layihəni yadda saxlayaraq, proqramlara aşağıdakı adları verin: `Unit1` – `main`, `Unit2` – `Aboutform`, `Unit3` – `Setform`, `Project1` – `many`.

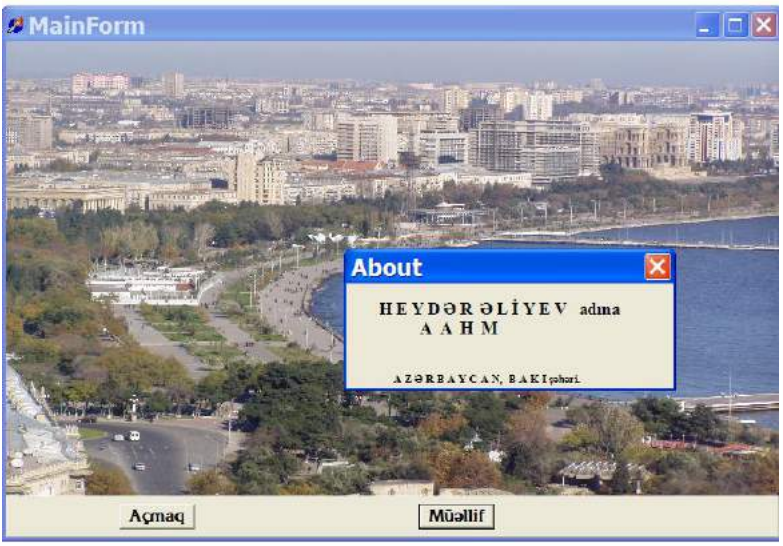
Bundan sonra, *Project* (*Layihə*) menyusundan *Options…* (*Parametrlər…*) əmrini icra etməklə, açılan dialoqun `Application` səhifəsinə keçərək `Title` xassəsinə `Many Window` adı daxil edin. `Title` xassəsi əlavənin sərlövhəsini müəyyən edir və o, layihə faylında kod kimi təsvir olunacaqdır.

Kod redaktorunda `main` səhifəsini seçərək *File/Uses Unit…* (*Fayl/İstifadəçi modulları…*) əmrini icra edin. Açılan dialoq pəncərəsində növbə ilə digər iki formanı seçərək *OK* düyməsini basın. Sonra, `Setform` səhifəsinə keçərək eyni qayda ilə `main` modulunu göstərin. Bu əməliyyatlar bir formadan digər formaları çağırmağa imkan verəcəkdir.

Yaradılacaq layihə aşağıdakı əməliyyatları yerinə yetirəcəkdir. Əsas forma üzərindəki `Açmaq` düyməsini basdıqda `Settings` dialoq pəncərəsi ekranda təsvir olunacaqdır. Bu pəncərənin `ComboBox` siyahısında hər hansı bir qrafik fayla tam yolu daxil edib, `Checked` dəyişdiricisi üzərində mausun düyməsini basdıqda (dəyişdiricidə bayrağın vəziyyətindən asılı olmayaraq) əsas formada şəkil təsvir olunacaqdır (şəkil 13.4). `Settings` pəncərəsi üzərində `Xülasə...` düyməsini basdıqda isə şəkil fayllarından ibarət dialoq pəncərəsindən seçilmiş

faylın ünvanı ComboBox komponentinin siyahısına, birinci element kimi daxil ediləcək, Göstər dəyiş-diricisi üzərində mausun düyməsini basdıqda, əsas formada yeni şəkil təsvir ediləcəkdir. Settings pəncərəsi bağlanmayınca, heç bir forma ilə işləmək mümkün olmayacaqdır. Settings formasının sərlövhəsində yalnız bir düymə – pəncərəni bağlama düyməsi olacaqdır. OK düyməsini basdıqda dialoq pəncərəsi bağlanacaqdır. Əsas formada Müəllif düyməsini basdıqda layihəni yerinə yetirən təşkilatın adı və ünvanından ibarət dialoq pəncərəsi peyda olacaqdır (şəkil 13.5).

Şəkil 13.4. MainForm əsas pəncərəsi və Settings dialoq pəncərəsi



Şəkil 13.5. MainForm əsas pəncərəsi və About dialoq pəncərəsi

Yaradılacaq layihənin layihə faylı və modullarının tam mətni aşağıda göstərilmişdir.

```
program Many;

uses
   Forms,
   main in 'main.pas' {MainForm},
```

```
   Aboutform in 'Aboutform.pas' {About},
   Setform in 'Setform.pas' {Settings};

{$R *.RES}

begin
   Application.Initialize;
   Application.Title := 'Many window';
   Application.CreateForm(TMainForm, MainForm);
   Application.CreateForm(TAbout, About);
   Application.CreateForm(TSettings, Settings);
   Application.Run;
end.


unit main;

interface

uses
     Windows,Messages,SysUtils,Classes,Graphics,
     Controls,Forms,Dialogs,ExtCtrls,StdCtrls;

type
 TMainForm = class(TForm)
   Panel1: TPanel;
   Button1: TButton;
   Button2: TButton;
   Image1: TImage;
   procedure Button1Click(Sender: TObject);
   procedure Button2Click(Sender: TObject);

 private
   { Private declarations }

 public
   { Public declarations }

 end;

var
  MainForm: TMainForm;

implementation

uses Aboutform, Setform;

{$R *.DFM}

procedure TMainForm.Button1Click(Sender: TObject);
begin
 Settings.ShowModal;
end;
```

```
procedure TMainForm.Button2Click(Sender: TObject);

begin

  About.Label1.Font.Name:= 'times latin';
  About.Label1.Font.Style:= [fsBold];
  About.Label1.Font.Size:= 13;
  About.Label1.Caption:=
        'H E Y D Ə R  Ə L İ Y E V adına '+
        #13#10 +'     A  A  H  M      ';

  About.Label2.Font.Name:= 'times latin';
  About.Label2.Font.Style:= [fsBold];
  About.Label2.Font.Size:= 8;
  About.Label2.Caption:=
        ' A Z Ə R B A Y C A N  B A K I şəhəri.';

  About.ShowModal;

end;

end.


unit Aboutform;

interface

uses
   Windows,Messages,SysUtils,Classes,Graphics,
   Controls,Forms,Dialogs,StdCtrls;

Type

  TAbout = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    procedure Button1Click(Sender: TObject);

  private
     { Private declarations }

  Public
     { Public declarations }

  end;

var
   About: TAbout;

implementation

{$R *.DFM}
```

```
procedure TAbout.Button1Click(Sender: TObject);
begin
  Close;
end;
end.

unit Setform;

interface

uses
   Windows,Messages,SysUtils,Classes,Graphics,
   Controls,Forms,Dialogs,ExtDlgs,StdCtrls,ExtCtrls;

Type

  TSettings = class(TForm)
    Bevel1: TBevel;
    ComboBox1: TComboBox;
    CheckBox1: TCheckBox;
    Button1: TButton;
    Button2: TButton;
    OpenPictureDialog1: TOpenPictureDialog;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure CheckBox1Click(Sender: TObject);


  private

     { Private declarations }

  Public

     { Public declarations }

  end;


var
   Settings: TSettings;


implementation

uses main;

{$R *.DFM}
```

```
procedure TSettings.Button1Click(Sender: TObject);
begin

  Settings.Close;

end;


procedure TSettings.Button2Click(Sender: TObject);
begin

  if OpenPictureDialog1.Execute then
  begin
    ComboBox1.Items.Add(OpenPictureDialog1.FileName);
    ComboBox1.ItemIndex:= ComboBox1.Items.Count-1;
  end;

end;


procedure TSettings.CheckBox1Click(Sender: TObject);
begin
  Mainform.Image1.Picture.LoadFromFile(ComboBox1.text);
end;
end.
```

On dördüncü fəsil

MÜSTƏSNA HALLAR

Müstəsna hallar əlavənin yerinə yetirilməsi prosesində baş verən səhvlərlə əlaqədardır. Proqramın yerinə yetirilməsi zamanı sintaksis, məntiqi və dinamik səhvlər baş verə bilər.

Sintaksis səhvlər dilin sintaksisinə əməl edilmədikdə baş verir. Bu səhvlər kompilyator tərəfindən aşkar edilir, istifadəçi tərəfindən aradan qaldırılır. Kompilyator səhvin yerini göstərməklə, səhvin xarakteri haqqında məlumat verir və proqramın işini dayandırır.

Məntiqi səhvlər düzgün olmayan alqoritmin yerinə yetirilməsi zamanı baş verir. Bu səhvlər haqqında kompilyator məlumat vermir və proqramın işini dayandırmır. Bu halda proqram korrekt işləmir və səhv nəticələr verir.

Dinamik səhvlər proqram icra olunduqda baş verir və bu səhvlər operator, prosedur, funksiya və proqram metodlarının, hətta əməliyyat sisteminin səhvləri nəticəsində əmələ gəlir. Dinamik səhvlərə *yerinə yetirmə vaxtının səhvləri* (*Runtime erors*) də deyirlər.

Proqramçı dinamik səhvlərin əmələ gələ biləcəyini əvvəlcədən görməli və onların baş verməməsi üçün tədbir görməlidir. Dinamik səhvləri emal etmək üçün Delphi–yə *müstəsna hal* anlayışı daxil edilmişdir.

Müstəsna hal proqramın yerinə yetirilməsi zamanı onun işinin tədricən və ya tamamilə dayandırılmasına səbəb olan səhvlərdən ibarətdir.

Müstəsna halları emal etmək üçün əlavə bir qlobal emalediciyə və müəyyən müstəsna hallara reaksiya verən bir neçə xüsusi prosedura – emaledicilərə malikdir. Hər bir müstəsna halı onun özünün *lokal emaledicisi* emal edir. Lokal emaledicisi olmayan müstəsna hal əlavənin *qlobal emaledicisi ilə* emal olunur. Qlobal emaledicilərə misal olaraq müxtəlif məsələlərin həllində tətbiq etdiyimiz istifadəçiyə verilən məlumatları (`MessageDlg`, `MesageBox` və s. vasitəsilə) göstərmək olar.

14.1. Müstəsna halların lokal aradan qaldırılması

Lokal emaledicilərin istifadə edilməsi üçün dilin tərkibinə iki **try...finally** və **try...except** konstruksiyaları daxil edilmişdir. Bu konstruksiyaların sintaksisləri eyni, təyinatları isə müxtəlifdir. Bu konstruksiyalardan hansının seçilməsi proqram operatorlarından və səhvlər baş verdikdə yerinə yetirilən əməliyyatlardan asılıdır. `try` operatorundan sonra bir və ya bir neçə operator yazmaq olar. Bir `try` operatorunun daxilində başqa `try` operatoru yerləşə bilər.

`try...finally` konstruksiyası iki bölmədən (`try` və `finally`) ibarətdir və onların ümumi forması belədir:

```
try
    // səhvlər əmələ gətirə biləcək operatorlar
finally
    // hətta səhv baş verdikdə yerinə yetirilməsi vacib olan operatorlar
end;
```

Bu konstruksiya səhvləri aradan qaldırmır, sadəcə olaraq səhv baş verdikdə də operatorları icra olunmağa məcbur edir. `try...finally` konstruksiyası belə işləyir: `try` bölməsində yazılmış operatorlardan birində müstəsna hal olarsa, onda idarəetmə `finally` bölməsində yerləşən birinci operatora verilir və bu bölmədəki bütün operatorlar yerinə yetirilir. Müstəsna hal baş vermədikdə isə operatorlar öz adi qaydası ilə icra olunur. Biz həll etdiyimiz məsələlərin bəzilərində bu konstruksiyanı tətbiq etmişdik. İndi isə bu konstruksiyanı tam dərk etmək üçün çox sadə bir misala baxaq.

Misal. Formanın bağlanması.

Boş forma üçün `OnClose` hadisə emaledicisini aktivləşdirərək yunitə bu kodları yazın:

```
Procedure TFormClose (Sender:TObject;
            var Action:TCloseAction);
begin
  // try
  Action:= caNone;
  // finally
```

```
  // Action:= caFree;
  // end;
end;
```

F9 klavişini basaraq proqramı icra edin. Sizin bütün cəhdlərinizə baxmayaraq, formanı bağlamaq mümkün olmayacaqdır. Çünki, burada yalnız bir operator icra olunur ki, o da pəncərəni bağlamağa icazə vermir (`Action:=caNone;`). Yenidən Delphi–yə qayıtmaq üçün *Run/Program Reset* əmrini icra edin və ya ***Ctrl+F2*** klavişlərini basın. Proqramda olan bütün şərh simvollarını (`//`) götürüb, onu yenidən icra edin. Bu dəfə pəncərə bağlanacaqdır, baxmayaraq ki, `Action` xassəsinə `caNone` (pəncərəni bağlamaq olmaz) qiyməti verilmişdir.

`try...finally` konstruksiyasından fərqli olaraq `try...except` konstruksiyası müstəsna halı aşkar və ləğv etməyə imkan verir. Bu konstruksiyanın ümumi forması belədir:

```
try
   // səhvlər əmələ gətirə biləcək operatorlar
except
   // hətta səhv baş verdikdə yerinə yetirilməsi vacib olan operatorlar
end;
```

`try...except` konstruksiyası belə işləyir: əgər `try` operatorunda müstəsna hal baş verərsə, onda idarəetmə `except` bölməsində yerləşən birinci operatora verilir. Əgər müstəsna hal yaranmırsa, onda `except` bölməsindəki operatorlar yerinə yetirilmir. Müstəsna hal baş verdikdə isə bu operatorlar səhvi aradan qaldırır və proqramın işini bərpa edir.

Misal. Müstəsna halların lokal emal edilməsi.

```
Procedure TForm.Button1Click(Sender:TObject);
Begin
  Try
    Edit1.Text:= IntToStr(StrToInt(Edit1.Text)+5);
  Except
    MesageDlg('Edit–in məzmunu'+Edit1.Text+
    #13#10+'ədəd deyil!',mtError,[mbOK],0);
    Edit1.Text:=' ';
    If Edit1.CanFocus then Edit1.SetFocus;
  end;
end;
```

`Button1` düyməsini basdıqda `Edit1`–dən daxil edilən ədədin üzərinə 5 əlavə edilir. Əgər `Edit1`–dən ədəd deyil, ixtiyari simvol daxil edilərsə, onda səhv baş verəcəkdir. Ona görə də bu operator `try` bölməsinə daxil edilmişdir. Bu səhvin emal edilməsi ondan ibarətdir ki (`except` bölməsi), səhv haqqında proqramçıya məlumat verilir, mətn sahəsindəki informasiya ləğv edilir və yenidən ədəd daxil etmək üçün fokus redaktorun özündə saxlanır. Əgər ədəd düzgün daxil edilərsə və toplama əməliyyatı müvəffəqiyyətlə yerinə yetirilərsə, onda `except` bölməsində yerləşən operatorların heç biri icra olunmayacaqdır.

On beşinci fəsil

QRAFİKLƏRLƏ İŞ

Delphi–də şəkilçəkmə kifayət qədər sadə prosesdir. Layihə yaratdıqda şəkil çəkmək üçün istifadəçinin sərəncamında *xolst* və ya *kanva* (`TCanvas` tipli `Canvas` xassəsi), *qələm* (`TPen` tipli `Pen` xassəsi), *fırça* (`TBrush` tipli `Brush` xassəsi), *şrift* (`TFont` tipli `Font` xassəsi) və bir neçə *sadə həndəsi fiqurlar – xətt, düzbucaqlı, ellips* və s. olur. Bütün komponentlər bu sadalanan xassələrin hamısına malik olmur. Məsələn, forma `Canvas`, `Font`, `Brush` xassələrinə, standart düymə `Font`, `Brush` xassələrinə, həndəsi fiqurlar (`Shape`) isə `Font`, `Pen` və `Brush` xassələrinə malikdir. Bunlardan başqa, təsvirlərlə işləmək üçün, Delphi bu sinifləri təklif edir:

- **`TPicture`** *–təsvirlər üçün konteyner*;
- **`TGraphic`** *–qrafik obyekt – təsvirlər üçün baza sinfi*;
- **`TBitMap`** *–rastr təsvirlər*;
- **`TIcon`** *–piktoqram*;
- **`TMetaFile`** *–metafayl.*

Delphi sistemində komponentlərin səthində şəkilçəkmə əməliyyatı digər əməliyyatlarda olduğu kimi, layihənin yerinə yetirilməsi prosesində proqram yolu ilə və ya əlavə konstruksiya edildikdə təsvirləri birbaşa yaratmaq yolu ilə yerinə yetirilə bilər.

Məlumdur ki, Windows əlavələri qrafik informasiyaları ekran və ya printerə **GDI** (*Graphics Devices Interfase – Qrafik qurğular interfeysi*) funksiyasının köməyi ilə çıxarır. *GDI*–də icra olunan funksiyalar aparat vasitələrindən asılı olmadığı üçün, əlavələr fiziki yox, məntiqi qurğularla işləyir. Məntiqi qurğular isə geniş rənglər palitrasına, yüksək yolverməyə və s. malikdir. İnformasiyanı ekran və ya printerə çıxardıqda drayver fiziki qurğunun məhdud imkanlarını və xüsusiyyətlərini nəzərə alır. Məsələn, əlavədə həndəsi fiqurun təsvirində bütün

16,7 milyon rəng istifadə oluna bilər, lakin bütün qurğular, o cümlədən, şırnaqlı printer bu qədər rəngi təsvir etmək qabiliyyətinə malik deyildir. Ona görə də, belə fiqurun çap edilməsi üçün, avtomatik olaraq, printerin imkan verdiyi rənglər seçilir. Analoji qayda şrift simvollarına da aiddir.

15.1. Qrafik komponentlər

Qrafik komponentlərlə şəkilçəkmə işlərinin yerinə yetirilməsi bir növ şəkilçəkmə redaktorları ilə işi xatırladır. Lakin, şəkilçəkmə redaktorlarından fərqli olaraq, Delphi–nin şəkilçəkmə komponentləri çox azdır və onların imkanları məhduddur. Ona görə də qrafik komponentlərdən istifadə etməklə haşiyə, faska və ya elementar həndəsi fiqurların təsvirlərindən ibarət sadə vizual effektlər yaratmaq mümkün olur.

Qrafik elementlər pəncərəsiz vizual komponentlər olmaqla, `TGraphicControl` sinfindən əmələ gəlmişdir. Ən çox istifadə olunan qrafik komponentlər `Shape` *həndəsi fiquru*, `Bevel` *faskası*, `Image` *qrafik obrazlarıdır*.

15.1.1. `Shape` komponenti

Komponentlər palitrasının **`Additional`** səhifəsində yerləşən **`Shape`** komponenti elementar *həndəsi fiqurları çəkmək* üçün nəzərdə tutulmuşdur. Bu komponentlə təsvir olunan fiqurun görkəmi eyniadlı `TShapeType` tipli `Shape` xassəsi ilə müəyyən edilir. Bu xassəyə verilən qiymətlər uyğun olaraq aşağıdakı həndəsi fiqurları çəkməyə imkan verir:

- **`stCircle`** *–dairə*;
- **`stEllipse`** *–ellips*;
- **`stRectangle`** *–düzbucaqlı*;
- **`stRoundRect`** *–oval künclü düzbucaqlı*;
- **`stRoundSquare`** *–oval künclü kvadrat*;
- **`stSquare`** *–kvadrat.*

Fiqurların rənglənməsi və doldurulması `Pen` və `Brush` xassələri ilə idarə olunur.

`TPen` tipli `Pen` xassəsi xolst üzərində *xətt* və *həndəsi fiqurları çəkmək üçün qələmin atributlarını* idarə edir. Bu xassənin özünün bir sıra xassələri vardır ki, onlardan ən əsasları bunlardır:

- `TColor` tipli `Color` xassəsi *qələmin rəngini* müəyyən edir;
- `TPenStyle` tipli `Style` xassəsi çəkilən xətlərin tərzini bildirir və bu xassə özü aşağıdakı qiymətləri ala bilər:
  - **`psSolid`** *–bütöv xətt (susmaya görə)* ;
  - **`psDash`** *–tire işarələrindən ibarət xətt*;

- **psDot** *–nöqtələrdən ibarət xətt*;
- **psDashDot** *–növbələşən nöqtə və tirelərdən ibarət xətt*;
- **psDashDotDot** *–növbələşən tire – nöqtə – nöqtə kombinasiyasından ibarət xətt*;
- **psClear** *–görünməyən xətt*;
- **psInsideFrame** *–eni 1 pikseldən qalın olan rəngli xətt.*

❖ `Integer` tipli `Width` xassəsi çəkilən xəttin piksellərlə qalınlığını müəyyən edir;

❖ `TPenMode` tipli `Mode` xassəsi xətt çəkdikdə qələmin xolstun pikselləri ilə *qarşılıqlı təsir üsulunu* müəyyən edir. Bu xassə yeni çəkilən xəttin rəngini bildirir və çoxlu qiymətlər alır ki, onlardan bəziləri aşağıdakılardır:

- **pmBlack** *–həmişə qara*;
- **pmWhite** *–həmişə ağ*;
- **pmNop** *–dəyişmir*;
- **pmNot** *–şəkilçəkmə səthində rənglərin inversiyası*;
- **pmCopy** *–*`Color` *xassəsində müəyyən edilmiş rənglə eyni olur (susmaya görə)*;
- **pmNotCopy** *–qələmin rənginin inversiyası*;
- **pmXor** *–qələm və ekranın rəngləri* `xor` *əməliyyatı ilə əlaqəlidir*;
- **pmNotXor** *–*`pmXor` *rejimində alınmış effektin inversiyası.*

`TBrush` tipli `Brush` xassəsi həndəsi fiqurları, məsələn, düzbucaqlı, ellips və s. *fırça ilə rəngləmək* üçün istifadə olunur. Fiqurların doldurulması `Brush` fırçasının xassələri ilə idarə olunur ki, onlardan ən əsasları aşağıdakılardır:

❖ `TColor` tipli `Color` xassəsi – fırçanın *rəngini* müəyyən edir;

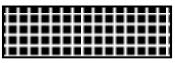
❖ `TBrushStyle` tipli `Style` xassəsi – fırçanın *tərzini* bildirir və bu xassə öz növbəsində aşağıdakı qiymətləri ala bilər:

- **bsSolid** *–tam doldurma*

- **bsClear** *–doldurulmur*
- **bsHorizontal** *–paralel üfqi xətlərlə doldurma*

- **bsVertical** *–paralel şaquli xətlərlə doldurma*

- **bsFDiagonal** *–yuxarıya yönəlmiş paralel diaqonal xətlərlə doldurma*

- **bsBDiagonal** *–aşağıya yönəlmiş paralel diaqonal xətlərlə doldurma* 
- **bsCross** *–düz qəfəslə doldurma* 
- **bsDiagCross** *–maili qəfəslə doldurma* 

❖ `TBitMap` tipli `BitMap` xassəsi fırça tərzi kimi istifadə edilən *rastr təsvirini* müəyyən edir. Bu xassə tətbiq olunduqda `Brush` fırçasının `Color` və `Style` xassələrindən istifadə edilmir.

15.1.2. `Bevel` komponenti

Faska ilə işləmək üçün Delphi **`Bevel`** komponenti təklif edir. *Faska* düzbucaqlı oblastdan, haşiyə və xətlərdən ibarətdir. Onlar müstəvi və həcmi görkəmdə olurlar ki, formanın digər elementlərindən fərqləndirilərək daha yaxşı nəzərə çarpsın. Bu komponent Komponentlər palitrasının **`Additional`** səhifəsində yerləşir. Faska üçün istifadə edilən fiqurlar `Bevel` komponentinin `TBevelShape` tipli `Shape` xassəsi ilə seçilir. `Shape` xassəsi aşağıdakı qiymətləri ala bilər:

bsBox *–düzbucaqlı*;
bsFrame *–haşiyə*;
bsTopLine *–yuxarıdan xətt*;
bsBottemLine *–aşağıdan xətt*;
bsLeftLine *–soldan xətt*;
bsRightLine *–sağdan xətt*;
bsSpacer *–proqram icra olunduqda görünməyən düzbucaqlı oblast.*

`TBevelStyl` tipli `Style` xassəsi faskanın tərzini bildirir və aşağıdakı qiymətləri ala bilər:

bsLowered *–yerləşdiyi səthə nisbətən faska çökmüş vəziyyətdədir*;
bsRaised *–səthə nisbətən faska azca qalxmış vəziyyətdədir.*

15.1.3. `Image` komponenti

`Image` komponenti *qrafik obrazlarla* işləmək üçün nəzərdə tutulmuşdur. Bu komponent adətən forma üzərində yerləşdirilir, görünməyən konteyner rolunu ifa edir və müvafiq formatlı şəkilləri öz üzərində təsvir etdirmək üçündür.

`Image` komponentinin bir neçə xassə və metodları vardır ki, onlardan ən əsası `Picture` xassəsidir. `TPicture` tipli `Picture` xassəsi `Image` komponentinin daxilində yerləşən *təsviri* müəyyən edir. `Image` komponenti

üzərində *.bmp, .jpg* və s. tipli şəkillər, piktoqramlar və metafayllar yerləşdirilə bilər. Bu təsvirlər, uyğun olaraq, `BitMap`, `Icon` və `MetaFile` xassələri ilə müəyyən olunur. `Image` komponentində yerləşdirilən qrafik obrazları isə `TGraphic` tipli `Graphic` xassəsi müəyyən edir. `TGraphic` sinfi rastr təsvirlər, piktoqram və metafayllar üçün mücərrəd əcdad sinfidir.

`Integer` tipli `Height` və `Widh` xassələri `Image` komponentində yerləşdirilmiş təsvirin *hündürlüyü* və *enini* bildirir. Qeyd edək ki, konteynerdə yerləşən təsvirin eni və hündürlüyü konteynerin özünün eni və hündürlüyündən fərqli olur. Bu parametrlər `Boolean` tipli `AutoSize` və `Stretch` xassələri ilə idarə olunur. Əgər `AutoSize` xassəsinə *True* qiyməti verilərsə, onda `Image` komponentinin ölçüləri onun üzərində yerləşən təsvirin ölçülərinə *uyğunlaşdırılır* (eyni olur). Bu xassəyə *False* qiyməti verdikdə isə ölçülər dəyişmir.

`Stretch` xassəsinə *True* qiyməti verildikdə konteyner üzərində yerləşən *təsvirin ölçüləri avtomatik* dəyişərək *konteynerin ölçüləri ilə eyni olur*. Bu xassəyə *False* qiyməti verildikdə isə ölçülər dəyişmir. Əgər şəklin ölçüsü konteynerin ölçülərindən böyükdürsə və bu zaman `AutoSize` və `Stretch` xassələrinə *False* qiyməti verilmişdirsə, onda şəklin bir hissəsi görünməyəcəkdir. Bu halda müvafiq xassələrə *True* qiyməti vermək və ya fırlatma zolaqlarından istifadə etmək lazımdır. Sonuncu halda, adətən, `Image` komponentinin özünü `Panel` konteyneri üzərində yerləşdirirlər. Qeyd edək ki, şəkillərin ölçülərini dəyişdirdikdə onların keyfiyyəti pisləşə bilər.

`Boolean` tipli `Center` xassəsi şəklin `Image` konteyneri üzərində *mərkəzə görə simmetrikliyini* idarə edir. Əgər bu xassəyə *True* qiyməti verilərsə, şəkil mərkəzə nisbətən simmetrik, *False* qiyməti verildikdə isə (susmaya görə) şəkil konteynerdə mərkəzə görə qeyri–simmetrik yerləşdirilir.

Çoxsənədli əlavələr yaratdıqda, biz `Image` komponentini və onun göstərilən xassələrini tətbiq etməklə məsələ həll etmişdik.

`Image` komponentində təsvir yerləşdirmək üçün

```
LoadFromFile(const FileName : String);
```

metodu, konteynerdə yerləşən təsviri yadda saxlamaq üçün isə

```
SaveToFile(const FileName : String);
```

metodu tətbiq edilir.

Şəkilləri Obyektlər inspektoru vasitəsilə də yükləmək olar. Bunun üçün `Picture` xassəsi qarşısındakı üç nöqtə təsvirli düyməni basdıqda açılan *Picture Editor* (*Təsvirlər redaktoru*) dialoq pəncərəsinin *Load* düyməsini basaraq yenidən açılan *Load Picture* dialoq pəncərəsindən faylı seçib, *Open* düyməsini basmaq lazımdır. Qeyd etmək lazımdır ki, şəkli Obyektlər inspektorundan yüklədikdə layihənin *exe*–faylının həcmi böyüyür. Ona görə də faylları proqram yolu ilə yükləmək daha məqsədəuyğundur.

15.1.4. `PaintBox` komponenti

Komponentlər palitrasının **`System`** səhifəsində yerləşən **`PaintBox`** komponenti *şəkilçəkmə oblastı* (*molbert*) yaratmaq üçün təqdim edilir. Bu komponentin əsas xassəsi `TCanvas` tipli `Canvas` xassəsidir. Şəkilçəkmə pəncərəsi əsasən o hallarda tətbiq olunur ki, şəkilçəkmə səthini `Canvas` səthindən kiçik oblastla məhdudlaşdırmaq və `Canvas` xassəsinə malik olmayan komponentləri çəkmək lazımdır. `PaintBox` komponenti görünməyən komponentdir və üzərindəki təsvirləri əks etdirir. `Image` komponentindən fərqli olaraq, `PaintBox` komponenti üzərində şəkil çəkilir və bu oblasta hazır şəkilləri yerləşdirmək olmaz. Bu komponentin `Canvas` xassəsini proqram yolu ilə şəkilçəkmə əməliyyatını izah etdikdə öyrənəcəyik.

15.1.5. `ImageList` komponenti

`ImageList` komponenti *qrafik obrazlar siyahısı* adlanır. Bu siyahıda qrafik təsvirlər saxlanır. Siyahı eyni ölçülü, eyni tipli obrazlardan ibarətdir. Hər bir obraza onun indeksi vasitəsilə müraciət etmək olar. Siyahı özü görünməyən komponentdir və onun tərkibində olan obrazlar da görünmür. Bu obrazlar yalnız hər hansı bir vizual komponent üzərində yerləşdirildikdə görünür. Siyahıdan təsviri seçmək üçün təsvirin nömrəsini bildirən xassədən istifadə edilir. Məsələn, alətlər paneli düymələri (`ToolButton`) üçün bu xassə `ImageIndex` xassəsidir.

Siyahıda yerləşən bütün elementlər *16*16* piksel ölçüdədir. Zərurət yarandıqda təsvirin ölçüsünü dinamik olaraq dəyişdirmək olar. Bu məqsədlə

`CreateSize` **(***AWidth***,** *AHeight* **: `integer`);**

konstruktoru istifadə edilir. *AWidth* və *AHeight* parametrləri təsvirin eni və hündürlüyünü bildirir. Təsvirin ölçülərini Obyektlər inspektorunda `Width` və `Height` xassələrinə qiymətlər verməklə də dəyişmək olar.

15.2. Proqram yolu ilə şəkilçəkmə

Şəkilçəkmə əməliyyatının əsasını `TCanvas` sinfi təşkil edir. Bu sinfin xassə və metodlarının köməyi ilə `Canvas` xassəsinə malik olan görünən komponentlər üzərində şəkil çəkmək mümkündür. `Canvas` xassəsinə malik olan əsas komponentlər `Form` forması, `Label` yazısı və `Image` qrafik obrazlarıdır. Şəkilçəkmə daha çox forma üzərində yerinə yetirilir.

Şəkilçəkmə səthi (xolst, kətan) piksel (**`Pixels`**[*x*,*y*]**:`TColor`;** xassəsi) adlanan kiçik kvadratlardan ibarət şəbəkədir. Hər pikselin iki nömrəsi olur: birinci nömrə pikselin üfqi, ikinci nömrə isə şaquli vəziyyətini müəyyən edir. Xolstun sol yuxarı küncünün koordinatı [*0,0*], yəni `Pixels[0,0]` olur. Piksellərin ümumi sayı üfqi koordinat üzrə – `Width`, şaquli koordinat üzrə isə

`Height` xassələri ilə müəyyən olunur. Hər bir piksel Windows sisteminin yol verdiyi rənglərlə rənglənə bilər. Bunun üçün həmin pikselə müvafiq rəng mənimsədilməlidir, məsələn,

```
Image1.Canvas.Pixels[100,100]:= clRed;
```

kodu [*100,100*] koordinatlı pikseli qırmızı rənglə rəngləyir. Əks əməliyyatla pikselin rəngini də müəyyən etmək olar:

```
Color:= Image1.Canvas.Pixels[100,100];
```

`Canvas` xassəsinə proqram yerinə yetirildiyi zaman müraciət etmək olar, ona görə də onun köməyi ilə çəkilən şəkillər dinamik olur. Bu şəkillər yalnız əlavə işlədikdə mövcud olur, onları yadda saxlamaq və ya çap etmək olar. Proqram yolu ilə tərpənməz və ya animasiyalı (ölçü, forma və vəziyyətini dəyişən) şəkillər yaratmaq olar.

15.3. Şəkilçəkmə səthi

Şəkilçəkmə səthi `TCanvas` *xolst* və ya *kanva* obyektindən ibarətdir.

`TCanvas=class(TPersistent);`

sinfi *GDI* qrafik təsvirlər interfeysini, alətləri (qələm, fırça, şrift) və sadə həndəsi fiqurları çəkmək üçün metodları birləşdirir. Baxmayaraq ki, `TCanvas` komponent deyildir, o xassə kimi bir sıra digər vizual komponentlərə daxildir.

İstənilən şəkilçəkmə səthinə `Tpen`– *qələm*, `Tbrus` – *fırça*, `Tfont`– *şrift* obyektləri daxildir. Qələm və fırça obyektləri – həndəsi fiqurları çəkmək, şrift obyekti isə mətnin atributlarını idarə etmək üçündür. Qeyd edək ki, `Canvas` xassəsinə malik olan komponentlərin özlərinin də `Pen`, `Brush` və `Font` xassələri ola bilər. Məsələn, forma üçün `Form1.Font` şriftləri `Form1.Canvas.Font` şriftləri ilə eyni deyildir.

Qrafik əməliyyatlar yerinə yetirildikdə şəkilçəkmə mövqeyi müəyyən edilməlidir. Bunun üçün

`MoveTo (`*x, y* **`: integer);`**

metodu tətbiq olunur. Bu prosedurun yerinə yetirilməsi nəticəsində qələm *x* və *y* mövqeyinə yerləşdirilir, lakin heç bir şəkil çəkilmir.

Həndəsi fiqurları çəkmək üçün aşağıdakı metodlar istifadə olunur:

- **`ARC (`***x1,y1,x2,y2,x3,y3,x4,y4***`:integer);`** *–qövs çəkmək*;
- **`Ellipse (`***x1,y1,x2,y2***`:integer);`***–içərisi doldurulmuş ellips çəkmək*;
- **`FillRect (const`** *Rect*`: TRect);`*–düzbucaqlı oblastın doldurulması*;
- **`FrameRect (const`** *Rect*`:TRect);`** *–içərisi boş düzbucaqlı çəkmək*;
- **`LineTo (`***x,y***`:integer);`***–cari mövqedən x və y koordinatlı nöqtəyə qədər düz xətt çəkmək*;
- **`Polygon (const`** *Points*`:array of TPoint);`***–içərisi doldurulmuş çoxbucaqlı çəkmək*;

- **PolyLine (const** *Points* **: array of TPoint);** *–içərisi boş çoxbucaqlı çəkmək*;
- **Rectangle (***x1,y1,x2,y2***:integer);** *–içərisi doldurulmuş düzbucaqlı çəkmək*;
- **RounRect(***x1,y1,x2,y2,x3,y3***:integer);***–oval künclü, içərisi doldurulmuş düzbucaqlı çəkmək.*

`Arc` və `Ellipse` metodlarında *x1* və *y1* – qövs və ellipsi əhatə edən düzbucaqlının sol yuxarı küncünün, *x2* və *y2* isə sağ aşağı küncünün koordinatlarını, *x3*, *y3*, *x4* və *y4* isə qövsün başlanğıc və son nöqtələrini bildirir.

`Rectangle` və `RounRect` metodlarında *x1* və *y1* –çəkilən düzbucaqlının sol yuxarı küncünün, *x2* və *y2* isə sağ aşağı küncünün koordinatlarını müəyyən edir.

`Polygon` və `PolyLine` metodlarında *Points* parametri çoxbucaqlının təpə nöqtələrindən ibarət massivdir.

Fiqurların xətt və rəngləri qələm və fırçanın xassələri ilə müəyyən edilir.

İndi isə bu fiqurların yaradılmasına aid bir neçə misala baxaq.

Misal. Bütün formanın səthini əhatə edən ellipsin çəkilməsi.

```
Procedure TForm1.FormPaint(Sender:TObject);
begin
  Canvas.Ellipse(0,0,ClientWidth,ClientHeight);
end;
```

Misal. Qırmızı rənglə doldurulmuş düzbucaqlının çəkilməsi.

```
Procedure TForm.Click(Sender:TObject);
Var
   NewRect: TRect;
begin
  NewRect:= Rect(20, 30, 50, 90);
  Form1.Canvas.Brush.Color:= clRed;
  Form1.Canvas.FillRect(NewRect);
end;
```

Misal. Düzbucaqlı daxilinə mətn çıxarmaq.

```
Var
   TheRect:TRect;
begin
  Canvas.Brush.Color:= clRed;
  TheRect.Top:= 10;
  TheRect.Left:= 10;
  TheRect.Bottom:= 100;
  TheRect.Right:= 100;
  Canvas.TextRect(TheRect, 10, 10, 'Boş düzbucaqlı');
  Canvas.FrameRect(TheRect);
end;
```

Burada, koordinatları (*10,10*) və (*100,100*) olan düzbucaqlı daxilində, `TextRect` metodu ilə, `Boş düzbucaqlı` mətni yazılır.

Misal. Düz xəttin çəkilməsi.

```
Procedure TForm1.FormMouseDown(Sender:TObject;
          Button:TMouse Button;Shift:TShiftState;
          x,y:integer);
begin
  Canvas.MoveTo(0,0);
  Canvas.LineTo(x,y);
end;
```

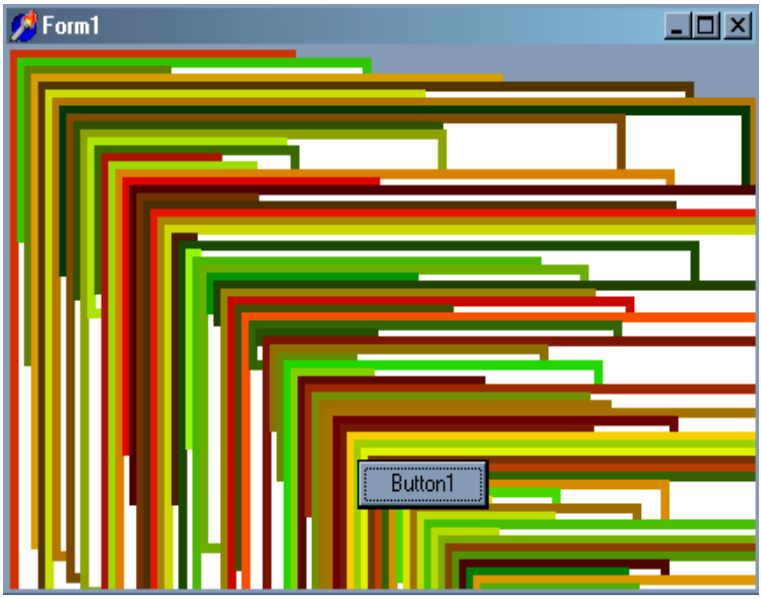
Burada, formanın sol yuxarı küncündən mausun düyməsi basılan nöqtəyədək düz xətt çəkilir.

Misal. Forma üzərində yaşıl rəngli çoxbucaqlının çəkilməsi.

```
Procedure TForm1.Button1Click(Sender:TObject);
begin
  Canvas.Brush.Color:= clTeal;
  Canvas.Polygon([Point(10,10), Point(30,10),
  Point(130,30), Point(240,120)]);
end;
```

Misal. Müxtəlif ölçülü və rəngli düzbucaqlıların çəkilməsi (şəkil 15.1).

```
unit Unit1;

interface

uses
    Windows,Messages,SysUtils,Classes,Graphics,
    Controls,Forms,Dialogs,StdCtrls,ExtCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Timer1: TTimer;
    procedure Button1Click(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);

  private
    { Private declarations }

  public
    { Public declarations }

  end;
```



Şəkil 15.1. Müxtəlif ölçülü və rəngli düzbucaqlılar

```
var
   Form1: TForm1;
   x,y: Integer;

implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender:TObject);
begin
  Timer1.Enabled:= True;
end;


procedure TForm1.FormActivate(Sender: TObject);
begin

  WindowState:= wsMaximized;
  Canvas.Pen.Width:= 5;
  Canvas.Pen.Style:= psDot;
  Timer1.Interval:= 50;
  Timer1.Enabled:= False;
  Randomize;

end;
```

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  X:= x+4;
  y:= y+4;
  Canvas.Pen.Color:= Random(65535);
  Canvas.Rectangle(x, y, x+Random(400), y+Random(400));
  if x>700 then Timer1.Enabled:= False;
end;

end.
```

Misal. Forma üzərində mausla düzbucaqlının çəkilməsi.

Elə proqram tərtib edək ki, biz özümüz formanın ixtiyari hissəsində, ixtiyari ölçülü düzbucaqlı çəkə bilək. Bu məsələnin məntiqi ondan ibarətdir ki, mausun düyməsini basdıqda forma üzərində nöqtənin koordinatlarını yadda saxlamaq və mausu dartdıqda düymənin basılmasını yoxlamaq: əgər düymə basılmışdırsa, onda həmin başlanğıc nöqtədən düzbucaqlı çəkmək.

Yunitin `private` bölməsinə düzbucaqlının başlanğıc koordinatlarını əlavə edin:

```
Private
   { Private declarations }
   SPenMode : TPenMode;
   BashX, BashY : integer;
   Chekmek : boolean;
```

Buradakı `Chekmek` dəyişəni mausun dartılması əlamətidir: əgər onun qiyməti *True* olarsa, deməli, istifadəçi mausun düyməsini basmışdır, mausun düyməsi basılmadıqda isə bu dəyişənin qiyməti *False* olmalıdır. Ona görə də, dərhal, `OnCreate` (forma yaradıldıqda) hadisəsini yaradaraq oraya cəmi bir sətir – `Chekmek:=False;` operatoru yazın.

Mausun düyməsini basma prosedurunu (`OnMouseDown`) yaradaq:

```
procedure TForm1.FormMouseDown(Sender: TObject;
          Button:TMouseButton;Shift:TShiftState;
          X,Y:Integer);
begin
   BashX:= X;
   BashY:= Y;
   Chekmek:= True;
end;
```

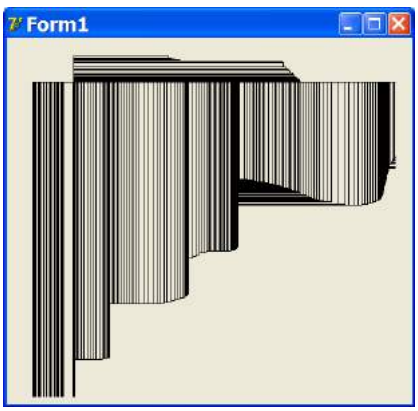
Burada, mausun düyməsi basılan nöqtənin koordinatları `BashX` və `BashY` dəyişənlərində yadda saxlanır və `Chekmek` dəyişəninə *True* qiyməti verilir (düzbucaqlının çəkilməsinə icazə verilir).

İndi isə mausun göstəricisinin yerinin dəyişdirilməsi prosedurunu (`OnMouseMove`) yaradaraq düzbucaqlı çəkmək prosedurunu (`Rectangle`) əlavə edək:

```
procedure TForm1.FormMouseMove(Sender: TObject;
          Shift:TShiftState;X,Y:Integer);
begin
   if Chekmek = False then exit;
   Canvas.Rectangle(BashX, BashY, X, Y);
end;
```

Mausun düyməsini buraxdıqda düzbucaqlının çəkilməsi prosesi dayandırılmalıdır və düzbucaqlı artıq çəkilmiş olmalıdır. Bunun üçün mausun düyməsinin buraxılması proseduru (`OnMouseUp`) yaradılmalı və `Chekmek` dəyişəninə *False* qiyməti verilməlidir:

```
procedure TForm1.FormMouseUp(Sender: TObject;
          Button:TMouseButton;Shift:TShiftState;
          X,Y:Integer);
begin
   Chekmek:= False;
end;
```



Şəkil 15.2. Forma üzərində mausla düzbucaqlılar çəkmək

F9 klavişini basmaqla layihəni yerinə yetirin. Mausun düyməsini basıb dartın. Düzbucaqlı çəkiləcək. Mausun düyməsini basıb yenidən dartdıqda isə ağlasığmaz təsvirlər müşahidə edəcəksiniz (şəkil 15.2). Bu vəziyyətdən çıxmaq üçün köhnə düzbucaqlının koordinatlarını yadda saxlamaq lazımdır. Bunun üçün `private` bölməsinə yeni üç dəyişən əlavə etmək lazımdır: `SPenMode` – şəkilçəkmə rejiminin yadda saxlanması, `SX` və `SY` – köhnə düzbucaqlının son koordinatlarının yadda saxlanması (başlanğıc koordinatlar `BashX` və `BashY` dəyişənlərində saxlanır):

```
private
   { Private declarations }
   SPenMode:TPenMode;
   BashX,BashY,SX,SY:integer;
   Chekmek:Boolean;
```

Növbəti mərhələdə `OnMouseDown` prosedurunu belə dəyişdirək:

```
procedure TForm1.FormMouseDown(Sender: TObject;
          Button:TMouseButton;Shift:TShiftState;
          X,Y:Integer);
begin
   Canvas.Brush.Color:= clWhite;
   SPenMode:= Canvas.Pen.Mode;
```

```
    Canvas.Pen.Mode:= pmNotXor;
    BashX:= X;
    BashY:= Y;
    SX:= X;
    SY:= Y;
    Chekmek:= True;
end;
```

Burada, əvvəlcə xolstun fırçası üçün ağ rəng müəyyənləşdirilmişdir. İkinci sətirdə `SPenMode` dəyişənində cari şəkilçəkmə rejimi yadda saxlanmışdır. Növbəti sətirdə xolstun `Mode` xassəsi üçün `pmNotXor` rejimi seçilmişdir. Bu rejim `Mode` xassəsinin ən çox istifadə edilən rejimidir. `pmNotXor` rejimi ona görə diqqətəlayiqdir ki, təkrarən şəkil çəkdikdə ekran nöqtələrinin rəngi bərpa olunur (yeri gəlmişkən, `Mode` xassəsinin yuxarıda göstərilmiş bütün rejimlərini bu proqramda müşahidə etmək Sizin üçün çox faydalı ola bilər). Kodun növbəti sətirlərində isə düzbucaqlının koordinatları yadda saxlanılmışdır.

`OnMouseMove` prosedurunda da dəyişiklik aparaq:

```
procedure TForm1.FormMouseMove(Sender: TObject;
          Shift:TShiftState;X,Y:Integer);
begin
   if Chekmek=False then exit;
   Canvas.Rectangle(BashX, BashY, SX, SY);
   Canvas.Rectangle(BashX, BashY, X, Y);
   SX:= X;
   SY:= Y;
end;
```

Əgər bu mərhələdə layihəni yerinə yetirsəniz, forma üzərində adi xətlə adi düzbucaqlı çəkiləcəkdir. Lakin, müvafiq dəyişiklikləri `OnMouseUp` prosedurunda da yerinə yetirsək, onda forma üzərində içərisi ağ rəngli düzbucaqlılar çəkiləcəkdir:

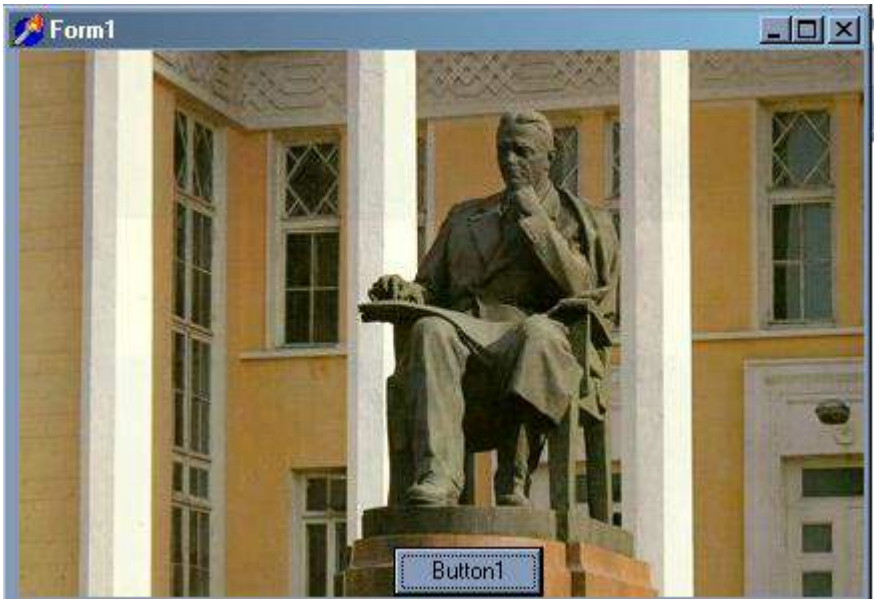
```
procedure TForm1.FormMouseUp(Sender: TObject;
          Button:TMouseButton;Shift:TShiftState;
          X,Y:Integer);
begin
   Chekmek:= False;
   Canvas.Pen.Mode:= SPenMode;
   Canvas.Rectangle(BashX, BashY, X, Y);
end;
```

Qrafik təsvirləri xolst üzərinə çıxarmaq üçün

`Draw` **(***x*,*y* **: `integer`;** *Graphic* **: `TCraphic`);**

prosedurundan istifadə olunur. Burada *x* və *y* təsvirin yerləşəcəyi oblastın sol yuxarı küncünü, *Graphic* parametri isə çıxarılan təsvirin tipini (rastr, metafayl və ya piktoqram) göstərir.

Misal. `Draw` metodunun tətbiqi (şəkil 15.3).



Şəkil 15.3. `Draw` metodu ilə şəklin kanva üzərinə çıxarılması

```
unit Unit1;

interface

uses
    Windows,Messages,SysUtils,Classes,Graphics,
    Controls,Forms,Dialogs,StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    OpenDialog1: TOpenDialog;
    procedure Button1Click(Sender: TObject);

  private
    { Private declarations }

  public
    { Public declarations }

  end;

var
   Form1: TForm1;

implementation
```

```
{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
Var
   Ris: TBitMap;
Begin
  Ris:= TBitMap.Create;
  try
   Ris.LoadFromFile('C:\UZEYIR.bmp');
   Form1.Canvas.Draw((Form1.Width div 2)-
        (Ris.Width div 2), (Form1.Height div 2)
        -(Ris.Height div 2), ris);
  finally
   Ris.Free;
  end;

end;

end.
```

Burada, `UZEYİR.bmp` qrafik faylı `Form1` formasında kanvanın mərkəzində təsvir edilir.

Şəkilçəkmə səthinə mətn çıxarmaq üçün

`TextOut` (*x, y* : **`integer`**; **`const`** *Text* : **`String`**);

metodundan istifadə oluna bilər. Bu metodla sol yuxarı küncü *x* və *y* koordinatları ilə müəyyən olunmuş oblasta *Text* parametri ilə verilən mətn çıxarılır. Bu halda mətnin şriftlərini proqramçı özü müəyyən edir.

Şəkilçəkmə səthinə mətn

`TextRect` (*Rect* : **`TRect`**; *x, y* : **`integer`**; **`const`** *Text* : **`Stringq`**);

metodu ilə də çıxarıla bilər. Bu metod `TextOut` metodundan yalnız *Rect* parametri ilə fərqlənir ki, bu parametr şəkil çıxarma oblastını məhdudlaşdırır. Biz hər iki metodun tətbiqini nümayiş etdirmişik.

`Canvas` xassəsinin ən vacib xassələri `Font`, `Pen` və `Brush` xassələridir. `Font` xassəsinin aldığı qiymətləri bu kitabın müxtəlif bölmələrində, `Pen` və `Brush` xassələrini isə `Shape` komponentini öyrəndikdə izah etmişik.

Şəkilçəkmə səthinin məzmunu dəyişdikdə `TNotifyEvent` tipli **`OnChanging`** və **`OnChange`** hadisələri baş verir. `OnChanging` hadisəsi xolstda dəyişiklikdən əvvəl, `OnChange` hadisəsi isə dəyişiklikdən sonra baş verir.

15.4. Animasiya

Animasiya dedikdə müxtəlif təsvirlərin ekranda forması, ölçüləri və vəziyyətinin dəyişməsi başa düşülür. Animasiyanı yaratmaq üçün müxtəlif alqoritmlər tətbiq olunur. Sadə alqoritmlərdən biri aşağıdakı əməliyyatlar ardıcıllığından ibarətdir:

1. *Obyekti müəyyən rənglə ekranda göstərmək*;
2. *Obyektin rəngini fonun rənginə çevirərək ekrandan onu yox etmək*;
3. *Müəyyən yerdəyişmə ilə obyekti öz rəngində yenidən ekranda təsvir etmək.*

Animasiya alqoritmlərini vizual komponentlərə də tətbiq etmək olar.

15.5. Qrafik təsvirlərə aid misallar

Misal. Həndəsi fiqurların koordinat üzrə hərəkət etdirilməsi.

Biz bu layihədə müxtəlif həndəsi fiqurlar yaradaraq onları fırlatma zolaqları vasitəsilə ekranda hərəkət etdirəcəyik. Fırlatma zolağı kifayət qədər mürəkkəb idarəetmə elementidir. O, fırlatma zolağından, onun üzərində hərəkət edən məkik və ucları ox təsvirli düymələrdən ibarətdir. Düymələri basdıqda ekranda olan obyekt – bir sətir, məkiyin yanında mausun düyməsini basdıqda isə obyekt bir səhifə yerini dəyişir. Məkiyi tutub hərəkət etdirməklə səhifənin müxtəlif hissələrinə cəld keçmək olar.

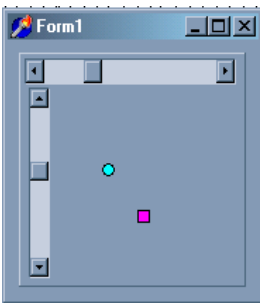
Beləliklə, forma üzərində bir `Panel` komponenti yerləşdirib, onun eni (`Width`) və hündürlüyünü (`Height`) *161* piksel müəyyən edin (tək ədədlə haşiyənin mərkəzini təyin etmək daha asandır). `Panel1` sərlövhəsini pozun. Bu panel üzərində `ScrollBar` (*Fırlatma zolağı*) komponenti yerləşdirin. Bizə şaquli fırlatma zolağı da lazımdır, ona görə də panel üzərinə daha bir `ScrollBar` komponenti endirin və Obyektlər inspektorunda onun `Kind` (*görünüş*) xassəsinə `sbVertical` qiyməti verin: fırlatma zolağı şaquli vəziyyət alacaqdır. Bu fırlatma zolağını formanın sol tərəfində yerləşdirin. Fiqurların panelin hüdudlarından kənara çıxmaması üçün onun sərhədlərindən *5* piksel aralı məsafə müəyyən edin (şəkil 15.4).

Üfqi fırlatma zolağını seçib Obyektlər inspektorunda onun aşağıdakı xassələrinə qiymətlər verin:

`Min` *–məkik sol kənarda yerləşdikdə onun minimal qiyməti – 25* daxil edin;

`Max` *–məkik sağ kənarda yerləşdikdə onun maksimal qiyməti – 150* daxil edin;

`Position` *– məkiyin cari vəziyyəti – 75* daxil edin; bununla də məkik ortada yerləşəcəkdir.



Şəkil 15.4. Fiqurları hərəkət etdirən əlavə

`SmallChange` – *bir sətir yerdəyişmə zamanı məkiyin yerini dəyişməsi – 2* daxil edin;

`LargeChange` – *bir səhifə yerdəyişmə zamanı məkiyin yerini dəyişməsi– 20* daxil edin.

Bütün bu xassələri, eyni qiymətlərlə, şaquli fırlatma zolağı üçün də müəyyən edin.

İndi isə formaya həndəsi fiqurları əlavə edək. Komponentlər palitrasının **`Additional`** səhifəsindən `Shape` komponentini seçərək, forma üzərində mausu dartmaqla, fiqur çəkin. Bu fiqurun `Height` və `Width` xassələrinə *11* daxil edin. Qəbul edək ki, bu fiqur aktiv olacaqdır, ona görə də onun `Left` və `Top` xassələrinə *76* daxil edin. `Shape` xassəsi üçün Obyektlər inspektorunda `stCircle` (*dairə*) seçin, `Brush` xassəsindən isə `Color` alt xassəsinə `clAqua` (*mavi*) rəng müəyyən edin. İkinci fiquru mübadilə buferi ilə yaradaq. Bunun üçün *Ctrl+C* və *Ctrl+V* əmrlərini icra edin. Yaranan `Shape2` fiqurunu panelin istənilən hissəsində yerləşdirin. Bu fiquru seçərək onun `Shape` xassəsinə `stSquare` (*kvadrat*), `Brush.Color` xassəsinə isə `clFuchsia` (*bənövşəyi rəng*) qiymətləri verin.

İndi isə layihəni proqramlaşdıraq. Hansı fiqurun cari olduğunu müəyyən etmək üçün modula `Boolean` tipli `num` qlobal dəyişəni əlavə edək. Forma üçün `OnCreate` (forma yaradıldıqda) hadisə emaledicisi yaradaraq oraya yalnız bir sətir – `num:=True;` yazın.

Məkik hərəkət etdikdə `OnChange` hadisəsi baş verdiyi üçün bu proseduru yaradaq. `ScrollBar1` komponentini seçərək, Obyektlər inspektorunda `OnChange` hadisəsi qarşısında mausun düyməsini iki dəfə basaraq, modula bu kodları yazın:

```
if num then
   Shape1.Left:= ScrollBar1.Position
else Shape2.Left:= ScrollBar1.Position;
```

Bu kodla hansı fiqurun cari fiqur olduğu müəyyən edilir. `ScrollBar2` komponentini seçib, `OnChange` prosedurunu yaradaraq, eyni kodu `ScrollBar2` komponenti və `Top` xassəsi üçün yazmaq lazımdır, yəni:

```
Shape1.Top:= ScrollBar2.Position;
```

Biz fiqurları mausla seçməliyik. Mausu fiqura yönəltdikdə bu fiqur cari və mavi rəngli (passiv fiqur isə bənövşəyi rəngli) olmaqla, məkiyin vəziyyəti bu fiqurun koordinatlarına uyğun olmalıdır. Bunun üçün isə `OnMouseMove` (maus hərəkət etdikdə) proseduru yaradılmalıdır. Ona görə də `Shape1` fiqurunu seçərək `Shape1MouseMove` proseduruna bu kodları yazın:

```
Shape1.Brush.Color:= clAqua;
Shape2.Brush.Color:= clFuchsia;
num:= True;
ScrollBar1.Position:= Shape1.Left;
ScrollBar2.Position:= Shape1.Top;
```

Analoji olaraq Shape2 komponenti üçün Shape2MouseMove proseduru belə yazılacaqdır:

```
Shape1.Brush.Color:= clFuchsia;
Shape2.Brush.Color:= clAqua;
num:= False;
ScrollBar1.Position:= Shape2.Left;
ScrollBar2.Position:= Shape2.Top;
```

Beləliklə, layihənin modulunun tam mətni aşağıdakı kimi olacaqdır.

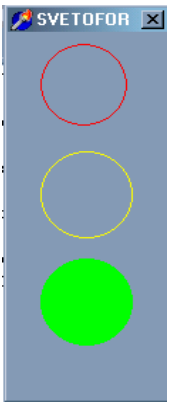
```
unit Unit1;

interface

uses
    Windows,Messages,SysUtils,Classes,Graphics,
    Controls,Forms,Dialogs,ExtCtrls,StdCtrls;

type
  TForm1 = class(TForm)
    Panel1: TPanel;
    ScrollBar1: TScrollBar;
    ScrollBar2: TScrollBar;
    Shape1: TShape;
    Shape2: TShape;
    procedure FormCreate(Sender:TObject);
    procedure ScrollBar1Change(Sender:TObject);
    procedure ScrollBar2Change(Sender:TObject);
    procedure Shape1MouseMove(Sender:TObject;
    Shift:TShiftState;X,Y:Integer);
    procedure Shape2MouseMove(Sender:TObject;
              Shift:TShiftState;X,Y:Integer);

  private
    { Private declarations }

  public
    { Public declarations }

  end;

var
   num:Boolean;
   Form1: TForm1;

implementation

{$R *.DFM}
```

```
procedure TForm1.FormCreate(Sender:TObject);
begin
num:=True;
end;


procedure TForm1.ScrollBar1Change(Sender:TObject);
begin
  if num then
      Shape1.Left:= ScrollBar1.Position
  else Shape2.Left:= ScrollBar1.Position;
end;


procedure TForm1.ScrollBar2Change(Sender: TObject);
begin
  if num then
      Shape1.Top:= ScrollBar2.Position
  else Shape2.Top:= ScrollBar2.Position;
end;


procedure TForm1.Shape1MouseMove(Sender:TObject;
           Shift:TShiftState;X,Y:Integer);
begin
  Shape1.Brush.Color:= clAqua;
  Shape2.Brush.Color:= clFuchsia;
  num:= True;
  ScrollBar1.Position:= Shape1.Left;
  ScrollBar2.Position:= Shape1.Top;
end;


procedure TForm1.Shape2MouseMove(Sender:TObject;
          Shift:TShiftState;X,Y:Integer);
begin
  Shape1.Brush.Color:= clFuchsia;
  Shape2.Brush.Color:= clAqua;
  num:= False;
  ScrollBar1.Position:= Shape2.Left;
  ScrollBar2.Position:= Shape2.Top;
end;

end.
```

F9 klavişini basdıqdan sonra, hazır layihə şəkil 15.4 – də olduğu kimi görünəcəkdir.

Misal. ”*Canlı işıqfor*”un yaradılması.

Bu layihədə biz “işıqfor” yaradacağıq ki, mausun göstəricisini ona yönəltdikdə “lampalar” rənglərini dəyişəcəkdir (şəkil 15.5).



Şəkil 15.5.
“İşıqfor”

Bu proqramda bizə yalnız bir `Shape` komponenti lazım gələcəkdir. *Shift* klavişini basılı saxlayaraq, **`Additional`** səhifəsindən `Shape` komponentini seçib, forma üzərində bir–birinin altında üç fiqur yerləşdirin. Sonra, mausla dartaraq bu fiqurları çəkin. Bu fiqurlar forma üzərində qara haşiyəli ağ düzbucaqlıdan ibarət olacaqdır. İndi bu fiqurları dairəyə çevirək. Bunun üçün onların `Width` və `Height` xassələrinə *61* ədədi daxil edərək onları əvvəlcə kvadrata, sonra isə `Shape` xassəsinə `stCircle` qiyməti verərək dairəyə çevirək. Dairə kontur və daxili oblastdan ibarətdir. Kontur `Pen` xassəsi ilə müəyyən olunduğu üçün, Obyektlər inspektorunda onun qarşısındakı + işarəsini basaraq açılan `Color` xassəsinə `clRed` (*qırmızı*) qiyməti seçin. Daxili oblast isə əvvəlcə şəffaf olmalı, lampanı qoşduqda isə rənglənməlidir. Fiqurun şəffaflığı `Brush.Style` xassəsi ilə müəyyən olunur. Fiqurun şəffaf olması üçün `Style` alt xassəsinə `bsClear` (*şəffaf*) qiyməti verilməlidir. Lakin, Delphi bu halda rəngi yadda saxlamır. Ona görə də bu əməliyyat proqramlaşdırma yolu ilə yerinə yetirilməlidir.

Eyni qayda ilə `Shape2` və `Shape3` komponentlərini sazlayaraq onlara `clYellow` (*sarı*) və `clLime` (*açıq yaşıl*) rəngləri verin.

İndi isə formada səliqə yaradaq. Bunun üçün formanın sərlövhəsini `İşıqfor` adlandırıb onun `Height` və `Width` xassələrinə *300* və *120* ədədləri daxil edin. Formanın sərlövhəsində artıq düymələrin olmaması üçün `BorderIcon` xassəsinin `biMinimize` və `biMaximize` alt xassələrinə *False* qiyməti verin. Formanın ölçülərinin dəyişməməsi üçün `BorderStyle` xassəsinə `bsSingle` qiyməti verin. “Lampaları” formada simmetrik və bərabər məsafələrlə yerləşdirmək üçün onların hər üçünü seçərək (*Shift* klavişini basılı saxlayıb mausun düyməsini hər bir “lampa” üzərində basmaqla) *Edit*/*Align* əmrini icra edin. Açılan dialoqun sol tərəfində *Center in Window* (*Pəncərəyə görə simmetrik*), sağ tərəfində isə *Space Equally* (*Bərabər məsafə*) dəyişdiricilərini qoşun. ***F9*** klavişini basmaqla bu mərhələdə layihəni icra edin və yuxarıdakı əməliyyatların düzgünlüyünə əmin olun. ***Alt+F4*** klavişlərini basaraq yenidən layihəyə qayıdın.

Bütün bu quruculuq işlərindən sonra proqramlaşdırmaya başlaya bilərik. Forma üzərində istənilən “lampanı” seçin. Görəcəksiniz ki, baxmayaraq ki, “lampa” dairəvidir markerlər hələ də düzbucaqlının tərəfləri üzərində qeyd olunur, başqa sözlə, Delphi bu fiqurları hələ də düzbucaqlı kimi başa düşür. Bu isə o deməkdir ki, mausun göstəricisini fiqurun küncünə yönəltdikdə o “yanacaqdır”. Bu isə yaxşı hal deyildir. Bu halı aradan qaldırmaq üçün “lampanın” `Enabled` xassəsinə *False* qiyməti verin. “Lampalar” mausun

göstəricisi onların üzərində yerləşdikdə yanacaqdır. Bu isə `OnMauseMove` hadisəsi baş verdikdə icra oluna bilər. "Lampalar" isə forma obyektinə mənsub olduğu üçün, biz `FormMauseMove` proseduru yaradacağıq. Odur ki, formanın boş yerində mausun düyməsini basıb Obyektlər inspektorunda `OnMauseMove` hadisəsini aktivləşdirin. Eyni əməliyyat bütün "lampalar" üçün təkrar olunur: mausu "lampaya" yönəltdikdə "yanır", onun oblastından çıxardıqda isə "sönür". Ona görə də bu əməliyyatı ayrı funksiya kimi tərtib edəcəyik. Bütöv rəngləmə üçün `Brush.Style` xassəsinə `bsSolid` qiyməti, "lampanı" söndürmək üçün isə ona `bsClear` qiyməti vermək lazımdır. Ona görə də bizim tərtib edəcəyimiz funksiya əsas proqrama bu qiymətləri qaytarmalıdır. Bu funksiyanı `ColShape` adlandıraq. Bu funksiyaya faktik parametr kimi, qoşulacaq "lampanın" adını və mausun koordinatlarını ötürmək lazımdır. Beləliklə, `FormMouseMove` prosedurunda birinci "lampanın" qoşulmasını belə yaza bilərik:

```
Shape1.Brush.Color:= clRed;
Shape1.Brush.Style:= ColShape(Shape1,X,Y);
```

`Shape1` komponentini `Shape2` və `Shape3` komponentləri ilə əvəz edib, `Color` xassəsinə, uyğun olaraq, `clYellow` və `clLime` qiymətləri mənimsətməklə, analoji kodları həmin digər iki "lampa" üçün də yazmaq lazımdır.

İndi isə `ColShape` funksiyasını yaradaq. Bu funksiya `FormMouseMove` prosedurundan əvvəldə yerləşdirilməlidir. Funksiyanın sərlövhəsini belə adlandıraq:

```
function ColShape(sh:Tshape; X,Y:Integer):TbrushStyle;
```

Burada, `sh` – funksiyaya ötürüləcək "lampanın" adı olduğu üçün, o `TShape` tipli elan edilir. Funksiya özü isə əsas proqrama bütöv rənglənmə üslubu ötürməli olduğu üçün, o, `TBrushStyle` tipli təyin olunur. Biz əvvəlcə mausun "lampaya" yönəldilməsini müəyyən etməliyik. Bunun üçün "lampanın" radiusunu hesablayaq:

```
r:= sh.Width div 2;
```

Fiqurun mərkəzinin koordinatlarını isə belə hesablamaq olar:

```
cx:= sh.Left+r;
cy:= sh.Top+r;
```

Sonra isə fiqurun mərkəzindən mausun göstəricisinə qədər olan məsafənin kvadratını tapaq:

```
d:= sqr(X-cx)+sqr(Y-cy);
```

Əgər mausun göstəricisi "lampa" daxilində olmazsa, biz yazmalıyıq:

```
ColShape:= bsClear;
```

Göstərici "lampa" daxilində olduqda isə kod belə yazılmalıdır:

```
if d<r*r then ColShape:= bsSolid;
```

Beləliklə, proqram yekunlaşdı və indi r, cx ,cy və d dəyişənlərini tam tipli elan etmək lazımdır. Yaratdığımız bu layihənin proqramının tam mətnini aşağıda göstəririk.

```
unit Unit1;

interface

uses
    Windows,Messages,SysUtils,Classes,Graphics,
    Controls,Forms,Dialogs,ExtCtrls;

type
  TForm1 = class(TForm)
   Shape1: TShape;
   Shape2: TShape;
   Shape3: TShape;
   procedure FormMouseMove(Sender:TObject;
             Shift:TShiftState;X,Y:Integer);

  private
    { Private declarations }

    public
    { Public declarations }

  end;

var
   Form1: TForm1;

implementation

{$R *.DFM}

function ColShape(sh:Tshape;X,Y:Integer):TbrushStyle;
Var
   r,cx,cy,d:Integer;
begin
  r:= sh.Width div 2;
  cx:= sh.Left+r;
  cy:= sh.Top+r;
  d:= sqr(X-cx)+sqr(Y-cy);
  ColShape:= bsClear;
  if d<r*r then ColShape:= bsSolid;
end;


procedure TForm1.FormMouseMove(Sender:TObject;
```

```
                Shift:TShiftState;X,Y:Integer);
begin

  Shape1.Brush.Color:= clRed;
  Shape1.Brush.Style:= ColShape(Shape1,X,Y);
  Shape2.Brush.Color:= clYellow;
  Shape2.Brush.Style:= ColShape(Shape2,X,Y);
  Shape3.Brush.Color:= clLime;
  Shape3.Brush.Style:= ColShape(Shape3,X,Y);

end;

end.
```

Misal. Naxışların yaradılması.

Sizə təqdim edəcəyimiz bu misalda kompüter animasiya rejimində ən müxtəlif naxışlar çəkəcək, istifadəçi isə tamaşaçı kimi onu izləyəcəkdir. Proqramın iş prinsipi aşağıdakılardan ibarətdir. Rənglər ardıcıl olaraq nömrələnir və bir rəng növbəti rənglə əvəz olunur. Sonuncu rəng istifadə olunduqda yenidən birinci rəng tətbiq olunur. Bunlardan başqa, şəkilçəkmə səthi xanalara bölünür və hər xana dörd "qonşu" xanadan ibarət olur. Hər qonşu xananın rəngi yoxlanır və onlar içərisində "növbəti" rəngli xana mövcud olarsa, cari xana həmin rənglə rənglənir. Proqramın animasiyasını isə taymer idarə edəcəkdir.

Proqram tərtib etmək üçün forma üzərinə Komponentlər palitrasının **System** səhifəsindən `PaintBox` şəkilçəkmə oblastı komponenti yerləşdirin. Bu komponentin ölçülərini *320*320* müəyyən edin. Formanı «`Naxış`» adlandırın. Formaya **System** səhifəsindən `Timer` komponenti yerləşdirib, onun `İnterval` xassəsinə *100* (*0,1* san) qiyməti daxil edin.

`FormCreate` və `Timer1Timer` prosedurlarından ibarət olan modulun tam mətni aşağıda göstərilmişdir. `FormCreate` prosedurunda təsadüfi qaydada xanalar massivi yaradılır, `Timer1Timer` prosedurunda alqoritmin proqramı verilmişdir. Bu proqram ənənəvi Pascal proqramlaşdırmasından ibarətdir.

```
unit Unit1;

interface

uses
    Windows,Messages,SysUtils,Classes,Graphics,
    Controls,Forms,Dialogs,ExtCtrls;

type
  TForm1 = class(TForm)
    PaintBox1: TPaintBox;
    Timer1: TTimer;
```

```
    procedure Timer1Timer(Sender:TObject);
    procedure FormActivate(Sender:TObject);

  private
    { Private declarations }

  public
    { Public declarations }

  end;

const
  size=40;
  csize=15;
  Colors:array[1..16] of TColor=
     (clRed,clGreen,clYellow,clBlue,clWhite,
     clGray,clFuchsia,clTeal,clNavy,clMaroon,
     clLime,clOlive,clPurple,clSilver,
     clAqua,clBlack);

var
   Form1: TForm1;
   Points: array[1..size,1..size] of integer;

implementation

{$R *.DFM}


procedure TForm1.FormActivate(Sender: TObject);
Var
   i,j: integer;
begin
  Randomize;
  for i:=1 to size do
  for j:=1 to size do
  Points[i,j]:= 1+Random(csize);
end;


procedure TForm1.Timer1Timer(Sender:TObject);
Var
   i,j: integer;
   c,l,r,u,d: integer;
   newPoints: array[1..size,1..size] of integer;
begin
  for i:=1 to size do
  for j:=1 to size do
```

```
    begin

      c:=Points[i,j]+1;
      if c>csize then c:=1;
      u:=i-1;
      if u=0 then u:=size;
      d:=i+1;
      if d>size then d:=1;
      l:=j-1;
      if l=0 then l:=size;
      r:=j+1;
      if r>size then r:=1;
      newPoints[i,j]:= Points[i,j];
      if(Points[u,j]=c)or(Points[d,j]=c)
      or(Points[i,l]=c)or(Points[i,r]=c)
      then newPoints[i,j]:=c;
    end;
  c:=320 div size;
  for i:=1 to size do
  for j:=1 to size do
    begin
      Points[i,j]:= newPoints[i,j];
      PaintBox1.Canvas.Pen.Color:= Colors[Points[i,j]];
      PaintBox1.Canvas.Brush.Color:= Colors[Points[i,j]];
      PaintBox1.Canvas.RectAngle(c*(i-1),c*(j-1),c*i,c*j);

      {Bu sətrin əvəzinə
      PaintBox1.Canvas.RectAngle(c*(i-1),c*(j-1),
                                 c*i-1,c*j-1);
      yazmaq olar}

    end;
  end;

  end.
```

F9 klavişini basmaqla proqramı işə salın.

Siz, ekranda cürbəcür naxışlar təsviri görəcəksiniz (şəkil 15.6). İlk anlar naxışlar tərpənməz olacaq, bütün oblast əhatə olunduqda isə naxışlar aydınlaşmağa başlayacaqdır. Ən maraqlı nəticələr *10–12* rəng istifadə olunduqda alınır. Naxışın evolyusiyasını əvvəlcədən demək çətindir. Rənglərin sayını (`csize`) azaltdıqda spiralların sayı o qədər artır ki, adi insan gözü ilə rənglərin dəyişmə qanunauyğunluğunu izləmək olmur. Rənglərin sayını artırdıqda isə spiralların sayı azalır və naxış birrəngli şəklə çevrilir. Bəzi hallarda naxış "sürünən zolağa" çevrilir. Bütün bu incəlikləri `size` – xanaların sayı və `csize` – rənglərin sayı sabitlərini dəyişməklə müşahidə etmək olar (şəkil 15.6). "Qonşu" xanaların sayını – dörd (`u`, `d`, `l` və `r` dəyişənləri) yox, səkkiz qəbul etdikdə tamam başqa naxışlar alınacaqdır.

Şəkil 15.6. Parametrlərin müxtəlif qiymətlərində alınan naxışlar

Misal. Lissaju əyrilərinin çəkilməsi.

Orta məktəbdən məlumdur ki, əgər ossilloqrafın *X* və *Y* girişlərinə sinxron sinusoidal siqnal verilərsə, ekranda düz xətt parçası əmələ gələr, fazanı bir az dəyişdikdə parça ellipsə, fazanın π/2 qiymətində və sinusoidin eyni amplitud qiymətlərində isə çevrəyə çevrilir. Ümumi halda bu əyrilər

$X=aSin(\omega_1 t+\varphi_1)$,
$Y=aSin(\omega_2 t+\varphi_2)$

parametrik tənlikləri ilə yazılır və dörd parametrdən: iki ω_1, ω_2– tezlik və iki φ_1, φ_2– faza yerdəyişmələrindən asılıdır. Proqramda biz *X*, *Y* dəyişənlərini `Fx` və `Fy` dəyişənləri ilə, ω_1, ω_2, φ_1 və φ_2 dəyişənlərini isə uyğun olaraq `wx`, `wy`, `w1`, `w2` dəyişənləri ilə işarə edəcəyik.

Layihəni yaratmaq üçün boş forma üzərinə `Image1` komponenti yerləşdirin. Bu komponentin ölçülərini `FormActivate` prosedurunda proqramla müəyyənləşdirəcəyik. Proqram icra olunduqda `wx`, `wy`, `w1` və `w2` qiymətlərini dəyişdirmək üçün forma üzərinə dörd `SpinEdit` komponentləri və onların sərlövhələrini işarə etmək üçün dörd `Label` komponentləri yerləşdirin (şəkil 15.7). `Label` komponentlərini `SpinEdit` komponentləri qarşısında yerləşdirərək onların `Caption` xassələrinə, uyğun olaraq, `wx=`, `wy=`, `w1=` və `w2=` mətnləri yazın. `SpinEdit` komponentlərinin adlarını (`Name`) da eyni qayda ilə dəyişdirin: `SpinEditwx`, `SpinEditwy`, `SpinEditw1` və `SpinEditw2` .

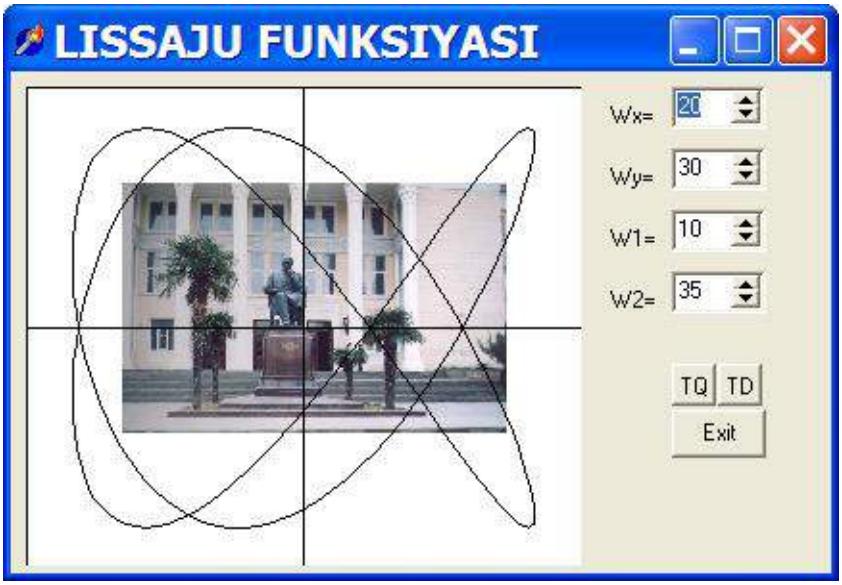
Lissaju funksiyalarını hesablamaq üçün `Fx` və `Fy` funksiyalarını yaradaq:

```
function TForm1.Fx(t:real):real;
begin
  Fx:= Sin(Wx*t+W1);
end;
```

```
function TForm1.Fy(t:real):real;
begin
  Fy:= Sin(Wy*t+W2);
end;
```

Şəkilçəkmə pəncərəsinin koordinatlarını `xI[x1,x2], yI[y1,y2]` müəyyən edək. *(x1,y1), (x2,y2)* kağız koordinatlarından *(I1,J1), (I2,J2)* ekran koordinatlarına keçmək üçün `II(x), JJ(y)` miqyaslaşdırma funksiyalarını yaradaq (bu funksiyalarla səkkizinci fəsildə tanış olduq):

```
function TForm1.II(x:real):integer;
// X oxy uzrə miqyaslaşdırma funksiyası
begin
  II:= I1+Trunc((x-X1)*(I2-I1)/(X2-X1));
end;

function TForm1.JJ(y:real):integer;
// Y oxy uzrə miqyaslaşdırma funksiyası
begin
  JJ:= J2+Trunc((y-Y1)*(J1-J2)/(Y2-Y1));
end;
```



Şəkil 15.7. Lissaju əyriləri

Əsas əməliyyatları icra etmək üçün `DrawGraphic` alt proqramını tərtib edək.

Bu alt proqramda parametrlərə başlanğıc qiymətlər verilir və `SpinEdit` komponentlərinin qiymətləri funksiyanın uyğun parametrlərinə ötürülür. Bu parametrlərin qiymətlərinin daha hamar dəyişməsi üçün, `SpinEdit`

komponentlərinin əsas xassəsi olan Value xassəsinin qiyməti *10* ədədinə bölünmüşdür. Bundan sonra, bu alt proqramda Image1 komponentinə aid parametrlər təyin edilir: Rectangle(0,0,Width,Height); metodu çağrılmaqla polotnoda düzbucaqlı çəkilir, sonra (X1,0) nöqtəsindən (X2,0) və (0,Y1) nöqtəsindən (0,Y2) nöqtəsinədək düz xətlər – koordinat oxları çəkilir. MoveTo(x,y) metodu ilə ekran göstəricisi (x,y) nöqtəsində yerləşdirilir və LineTo(x,y) metodu ilə (x,y) nöqtəsindən xətt, sonra isə qrafik çəkilir. Qrafik əvvəlcə t=0 nöqtəsində çəkilir, sonra isə dövr altında t=t+h nöqtələrində davam etdirilir.

SpinEdit komponenti üçün prosedur yaradaq ki, onun Value xassəsi dəyişdikdə qrafik yenidən qurulsun:

```
Procedure TForm1.SpinEditWxChange(Sender: TObject);
begin
  DrawGraphic;
end;
```

Digər SpinEdit komponentləri üçün OnChange hadisə emaledicisinə bu proseduru təyin edin.

Qələm və fırçanın rəngini təsadüfi dəyişmək imkanı daxil etməklə layihəni bir az mürəkkəbləşdirək. Bunun üçün forma üzərinə iki SpeedButton düyməsi və Timer komponenti yerləşdirin. SpeedButton1 düyməsinin sərlövhəsini – TQ (taymerin qoşulması), SpeedButton2 düyməsinin sərlövhəsini TD (taymerin dayandırılması) adlandırın. Bu düymələr üçün OnClick hadisə emaledicisini yaradaq. SpeedButton1 düyməsi taymeri qoşaraq DrawGraphic alt proqramını çağırır. SpeedButton2 düyməsi isə taymerin işini dayandırır. Taymeri dayandıraraq, SpinEdit komponentlərinin qiymətlərini istifadəçi özü dəyişdirməklə, proqramın icrasını əl ilə davam etdirə bilər.

Taymer üçün yeganə Timer1Timer(Sender); hadisəsi Interval xassəsi ilə müəyyənləşdirilən interval vaxtı ilə çağrılır (əgər Timer1Enabled:=True; olarsa). Əvvəlcə Timer1 üçün Enabled xassəsinə *False* qiymətini təyin edək.

SpeedButton1 düyməsini basdıqda *1000* millisaniyə intervalla taymer işə düşəcək və SpinEdit komponentlərinin Value xassəsinin qiymətləri, qələm və fırçanın rəngləri təsadüfi qaydada dəyişəcəkdir. Bunun üçün Timer1Timer prosedurunu yaradaq:

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Randomize;
  SpinEditWx.Value:= Random(100);
  SpinEditWy.Value:= Random(100);
  SpinEditW1.Value:= Random(100);
  SpinEditW2.Value:= Random(100);
```

```
  Image1.Canvas.Brush.Color:= Random($FFFFFF);
  Image1.Canvas.Pen.Color:= Random($FFFFFF);
end;
```

Layihəyə sonuncu dəyişiklik əlavə edək. `Image1` polotnosuna şəkil yerləşdirək. Bunun üçün `DrawGraphic` alt proqramında `Ris:TBitMap` dəyişəni yaratmaq lazımdır:

```
Ris:= TBitMap.Create;
Ris.Transparent:= True;
Ris.TransparentMode:= tmAuto;
Ris.LoadFromFile('C:\UZEYIR.bmp');
Ris.TransParentColor:= 3707096;
```

Bundan başqa, `DrawGraphic` alt proqramına

```
StretchDraw(Rect(50,50,250,180),Ris);
```

sətri əlavə edilməlidir ki, bu sətir şəkli `Image1` komponenti üzərində `Rect(60,60,200,220)` düzbucaqlısı daxilində yerləşdirir.

Nəhayət, bütün alt proqramlar modulun `Public` bölməsinə əlavə edilməlidir.

Beləliklə, həll etdiyimiz məsələnin modulu belə olacaqdır:

```
unit Unit1;

interface

uses
   Windows,Messages,SysUtils,Classes,Graphics,Controls,
   Forms,Dialogs,StdCtrls,Spin,ExtCtrls,Buttons,ExtDlgs;

type
  TForm1 = class(TForm)
    Image1: TImage;
    SpinEditWx: TSpinEdit;
    SpinEditWy: TSpinEdit;
    SpinEditW1: TSpinEdit;
    SpinEditW2: TSpinEdit;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    ColorDialog1: TColorDialog;
    Timer1: TTimer;
    OpenPictureDialog1: TOpenPictureDialog;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    Button1: TButton;
    procedure SpinEditWxChange(Sender:TObject);
```

```
    procedure SpeedButton1Click(Sender:TObject);
    procedure Timer1Timer(Sender:TObject);
    procedure SpeedButton2Click(Sender:TObject);
    procedure Button1Click(Sender:TObject);

  private
     { Private declarations }

  Public
    { Public declarations }

     procedure DrawGraphic;
     function Fx(t:real):real;
     function Fy(t:real):real;
     function II(x:real):integer;
     function JJ(y:real):integer;

  end;

var
   Form1: TForm1;
   BitMap: TBitMap;
   x,y: real;
   Wy,W2: real;
   Wx,W1: real;
   X1,X2,Y1,Y2: real;
   I1,I2,J1,J2: Integer;
   n: Integer;
   h: real;

implementation

{$R *.DFM}

function TForm1.Fx(t:real):real;
begin
  Fx:= Sin(Wx*t+W1);
end;


function TForm1.Fy(t:real):real;
begin
  Fy:= Sin(Wy*t+W2);
end;


function TForm1.II(x:real):integer;
// X oxu üzrə miqyaslaşdırma funksiyası
begin
```

```
  II:= I1+Trunc((x-X1)*(I2-I1)/(X2-X1));
end;


function TForm1.JJ(y:real):integer;
// Y oxu üzrə miqyaslaşdırma funksiyası
begin
  JJ:= J2+Trunc((y-Y1)*(J1-J2)/(Y2-Y1));
end;


Procedure TForm1.DrawGraphic;

Var
   i: Integer;
   t: Real;
   Ris: TBitMap;

begin
  n:= 200;
  X1:= -1.2;X2:=1.2;Y1:=-1.2;Y2:=1.2;
  Ris:= TBitMap.Create;
  Ris.Transparent:= True;
  Ris.TransparentMode:= tmAuto;
  Ris.LoadFromFile('C:\UZEYIR.bmp');
  Ris.TransParentColor:= 3707096;
  // Ekran pəncərələrinin ölçülərinin verilməsi
  With Image1 do
   begin
    I1:=0; J1:=0; I2:= Width; J2:= Height;
   end;
  // X oxu üzrə addımın hesablanması
  h:= 2*Pi/n;
  Wx:= SpinEditWx.Value/10;
  Wy:= SpinEditWy.Value/10;
  W1:= SpinEditW1.Value/10;
  W2:= SpinEditW2.Value/10;
  With Image1.Canvas do
   begin
    RectAngle(0, 0, Width, Height);
    StretchDraw(Rect(50, 50, 250, 180), Ris);
  // Koordinat sisteminin qurulması
    MoveTo(II(0),JJ(Y1)); LineTo(II(0),JJ(Y2));
    MoveTo(II(X1),JJ(0)); LineTo(II(X2),JJ(0));
  //  Funksiyanın parçalarla qrafikinin qurulması
    t:=0; x:= Fx(t); y:= Fy(t); MoveTo(II(x),JJ(y));
    for i:=1 to 5*n do
     begin
      t:=t+h; x:=Fx(t);
      y:=Fy(t); LineTo(II(x),JJ(y));
```

```
      end;
    end;
end;


Procedure TForm1.SpinEditWxChange(Sender: TObject);
begin
  DrawGraphic;
end;


procedure TForm1.SpeedButton1Click(Sender: TObject);
begin
  Timer1.Enabled:= True;
  DrawGraphic;
end;


procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Randomize;
  SpinEditWx.Value:= Random(100);
  SpinEditWy.Value:= Random(100);
  SpinEditW1.Value:= Random(100);
  SpinEditW2.Value:= Random(100);
  Image1.Canvas.Brush.Color:= Random($FFFFFF);
  Image1.Canvas.Pen.Color:= Random($FFFFFF);
end;


procedure TForm1.SpeedButton2Click(Sender: TObject);
begin
  Timer1.Enabled:= False;
end;


procedure TForm1.Button1Click(Sender: TObject);
begin
  Close;
end;

end.
```

ƏDƏBİYYAT

1. Vəliyev Ə.A., Məhərrəmov Z.T., Əbilov Y.Ə. Delphi: nəzəriyyə və təcrübə. Ali məktəb tələbələri üçün dərs vəsaiti. – Bakı: Çaşıoğlu, 2004. – 336 s.
2. Məhərrəmov Z.T. Alqoritm və onun təsvir üsulları. Bakı, 2006.– 30 s.
3. Kərimov S.Q., Həbibullayev S.B., İbrahimzadə T.İ. İnformatika. Ali məktəb tələbələri üçün dərslik. – Bakı: 2009. – 434 s.
4. Vəliyev N.N. Turbo Pascal Windows üçün. Bakı: 2003. – 142 s.
5. Немнюгин С.А. Turbo Pascal. – СПб.: Питер, 2002. – 496 с.
6. Федоренко Ю. Алгоритмы и программы на Turbo Pascal. Учебный курс. – СПб.: Питер, 2001. – 240 с.
7. Гофман В.Э., Хомоненко А.Д. Delphi 5 – СПб.: БХВ – Петербург, 2001. – 800 с.
8. Тюкачев Н., Свиридов Ю. Delphi 5. Создание мультимедийных приложений. Учебный курс. – СПб.: Питер, 2001. – 400 с.
9. Фленов М.Е. Библия Delphi.– СПб.: БХВ-Петербург, 2008.–800 с.

Məhərrəmov Zakir Tülü oğlu

Texnika elmləri namizədi,
dosent

PASCAL – dan DELPHİ – yə