

Alqoritm, xassələri və təsvir üsulları.

Alqoritm – qarşıya qoyulan məsələni həll etmək üçün yerinə yetirilməsi vacib olan əməliyyatlar ardıcılığıdır.

Latınca qayda-qanun deməkdir. Alqoritm 783- 850-ci illərdə Xorezmdə (indiki Özbəkistanda şəhər) yaşamış IX əsrin məşhur özbək riyaziyyatçısı Məhəmməd İbn Musa əl-Xarəzminin (yəni Xarəzmi Musa oğlu Məhəmməd) adının latın hərflərilə olan “alqoritm” yazılışıyla bağlıdır. Əl-Xarəzminin yazdığı traktatın XII əsrdə latın dilinə tərcümə olunması sayəsində avropalılar mövqeli say sistemi ilə tanış olmuş, onluq say sistemini və onun hesab qaydalarını alqoritm adlandırmışlar. Ümumiyyətlə, alqoritm-verilmiş məsələnin həlli üçün lazım olan əməliyyatları müəyyən edən və onların hansı ardıcılıqla yerinə yetirilməsini göstərən formal yazılışdır.

Uzun müddət alqoritm anlayışından yalnız riyaziyyatçılar müxtəlif məsələnin həll qaydası kimi istifadə etmişlər. Riyaziyyat elminin inkişafı əsas riyazi anlayış olan alqoritm anlayışının dəqiqləşdirilməsini tələb etmiş və bu da riyaziyyatın yeni sahəsi olan “Alqoritm nəzəriyyəsinin “yaranmasına səbəb olmuşdur. Elektron hesablama texnikasının və proqramlaşmanın inkişafı ilə əlaqədar olaraq alqoritm anlayışının böyük əhəmiyyəti daha da qabarıq şəkildə meydana çıxmışdır alqoritm qurulması EHM-inköməyi ilə məsələ həlli prosesinin zəruri mərhələsidir. Həlli üçün alqoritm qurulmuş və kompüter proqramı şəklində yazılmış məsələləri kompüterdə asanlıqla realizə etmək olar Hesablama maşınlarının əsas fərqləndirici xüsusiyyətlərindən biri də onun proqramla idarə olunmasıdır. Yəni, istər sadə, istərsə də mürəkkəb məsələni maşının həll etməsi üçün proqram tərtib edilməlidir

Alqoritm xassələri

Məsələnin maşında həlli üçün tərtib edilən alqoritm bir çox şərtləri ödəməlidir. Bu şərtlərə alqoritm xassələri deyilir. Həmin xassələr aşağıdakılardır:

Diskretlik xassəsi alqoritm ayrı-ayrı elementar və qabaqcadan müəyyən edilmiş addımlara ,bəndlərə pqrçalanmasını göstərirhər bir mərhələnin yerinə yetirilməsi üçün müəyyən vaxt lazım gəlir. İlk verilənlərdən nəticənin alınması üçün müəyyən vaxt ərzində diskret yerinə yetirilir.

Nəticəlik və ya sonluluq xassəsi bu xassə onu bildirir ki, alqoritm sonlu sayda addımdan sonra başa çatmalı və tamami ilə müəyyən nəticə verməlidir

Müəyyənlik

Alqoritmin hər bir addımı dəqiq və birqiymətli təyin olunmalıdır. Bu alqoritmin müəyyənlik xassəsidir.

Alqoritmin müəyyən sayda giriş qiymətləri (məsələnin başlanğıc şərtləri) olmalıdır. Bu şərtlər proqram icra olunmamış və ya olunduqca maşına daxil edilə bilər.

Alqoritmin yerinə yetirilməsi nəticəsində giriş qiymətlərindən asılı olan bir və ya bir neçə çıxış qiymətləri alınmalıdır.

Alqoritm sadə və səmərəli olmalıdır, yəni alqoritmin nəticəsi (cavabı) mümkün qədər sadə əməliyyatlar vasitəsilə və ən qısa yolla alınmalıdır.

kütləvilik xassəsi Alqoritm ümumi olmalıdır, yəni müəyyən məsələ üçün tərtib olunmuş alqoritm, həmin tiptən (sinifdən) olan bütün məsələlər üçün yararlı olmalıdır.

Bu alqoritmin kütləvilik xassəsidir

Riyaziyyatda və informatikada məsələnin həllinin alqoritm yerinə yetirilibsə, məsələ qismən həll edilmiş sayılır.

Alqoritmin təsvir üsulları

Mətn şəkilində (adi dildə);

Qrafik – blok-sxem;

Cədvəl;

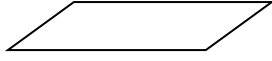
Proqram (alqoritmik dil).

Alqoritmin adi dildə təsviri (nəqli). Bu zaman əməliyyatlar, icra olunacaq hərəkətlərin nəqli şəkilində ardıcıl sadalanması kimi verilir. Məsələn, kofenin hazırlanmasını ifadə edən alqoritmin təsviri buna misal ola bilər.

Alqoritmin blok-sxem təsviri. Mürəkkəb alqoritmlərin təsviri zamanı blok-sxemlərdən istifadə olunması daha geniş yayılmışdır, çünki bu halda alqoritmin blok-sxem şəklində təsviri daha əyani olur. Bu zaman, adətən alqoritmin bir addımına bir blok uyğun olur. Lakin bir blokda bir neçə eyni tipli mərhələ və ya bir mərhələ bir neçə blokda təsvir oluna bilər. Bloklar standart işarələr şəklində ifadə olunur və bir-birləri ilə şaquli və ya üfüqi xətlərlə birləşdirilir. Birləşdirici xətlərin uclarında istiqaməti göstərən ox işarəsi qoyulur.



Alqoritmin başlanğıcı və sonu bu fiqur icərisində yazılır.



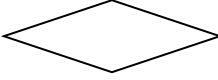
İlkin verilənlərin daxil edilməsi paraleloqram fiquru ilə təsvirolunur və onun içərisində qiymətləri daxil edilməli olan dəyişənlərin adı yazılır



Hesablama blokunun daxilində yerinə yrtirməli olan əməliyyatlar yazılır



İçərisində qiyməti çap edilməli olan dəyişənlərin adı yazılır.



Şərtin yoxlanma əmri romb şəklində təsvir olunur. Ödəniləcək şərt onun içərisində yazılır . şərtin ödənilib-ödənilməməsindən asılı olaraq hesablama prosesi iki mümkün istiqamətdən biri üzrə davam etdirilir.

Alqoritm ayrı-ayrı ədədlərlə yox, verilmiş hər hansı obyektlərlə işləyir. Proqramlaşdırmanın əsas obyekti dəyişəndir. Məsələn, x adlı dəyişənə 5 qiymətinin mənimsənilməsini belə müəyyən etmək olar:

x := 5 yazılır və x = 5 olur.

Proqramlaşdırmada məsələni alqoritmləşdirməkdən qabaq aşağıdakı addımlar yerinə yetirilməlidir:

Məsələnin riyazi qoyuluşu:

Nə verilir – ilkin verilənlərin sadalanması;

Nə tələb olunur – nəticələrin sadalanması ;

İlkin verilənlərin məhdudiyət şərtləri.

Riyazi model: nəticələri almaq üçün lazım olan bütün qayda və qanunlar.

Həll metodu: riyazi modelin optimal istifadə olunması.

Aşağıdakı misala baxaq:

Verilmiş kvadrat tənliyin həlli üçün alqoritm:

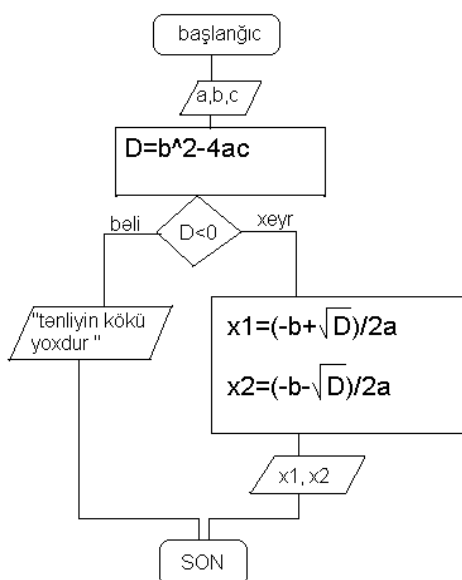
1) tənliyin a,b,c əmsallarını daxil etmək;

2) $D = b^2 - 4ac$ ifadəsini hesablamaq;

3) əgər $D < 0$ olarsa, 5 bəndinə, əks halda 4 bəndinə keç;

4) $X1 = (-b + \sqrt{D}) / 2 * a$, $X2 = (-b - \sqrt{D}) / 2 * a$ hesablamaq;

5) hesablamaları qurtarmalı. Son



Kompilyasiya və interpretasiya

Translyasiyanın iki qaydası var: interpretasiya və kompilyasiya. Interpretasiya – şifahi tərcüməyə oxşayır. Giriş proqramının hər bir təlimatı tərcümə olunur və yerinə yetirilir. Bu qaydada təkrar təlimatlar hər dəfə kodlaşdırılır. Kompilyasiya isə yazılı tərcüməyə bənzəyir. Proqram yerinə yetirilməzdən qabaq proqramın bütün tərcüməsi yığılır.

İnterpretasiya böyük çevikliyə malik olmaqla asan realizə olunur. Kompilyasiya isə daha effektiv proqram yaradır.

Proqramçı isə proqramlaşdırma dillərini bilməklə, qarşıya qoyulan məsələnin kompüterdə həllini həyata keçirmək üçün proqram yazır və onu kompüterdə yerinə yetirir.

Proqramlaşmanın bütün dilləri verilənlərin aşağıda göstərilən tipləri ilə işləməyə imkan verilir:

Tam ədədlər;

Məntiqi ədədlər;

Həqiqi ədədlər;

Simvollar;

Mətn tipli ədədlər;

Birtipli verilənlər cədvəli;

Fayllar.

Kompyuterin alqoritmi başa düşməsi üçün proqramlaşdırma dillərindən istifadə edilir. Məsələ həll edərkən əvvəlcə yerinə yetiriləcək əməliyyatların alqoritmi tərtib edilir, daha sonra bu əməliyyatlar hər-hansı alqoritm (proqramlaşdırma) dilində əmrlər şəklində yazılır. Tərtib olunmuş proqram xüsusi əlavələr (translyator proqramlar) vasitəsilə yerinə yetirilir və ya maşın koduna çevrilir.

Əsas alqoritmik baza strukturları

Alqoritm tərtibi prosesində aşağıdakı sadə tələblərin ödənilməsi məqsəduyğundur:

- alqoritm asan baza düşülən olmalıdır, bu başqasının tərtib etdiyi alqoritlərdən istifadə üçün lazımdır;
- alqoritm asanlıqla yoxlana bilməlidir;
- alqoritm yenidən tərtib edilmədən təkmilləşdirilə bilməlidir.

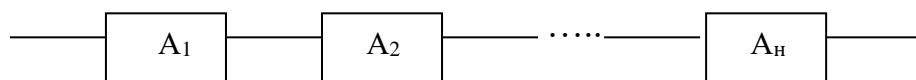
Alqoritm tərtibinə struktur yanaşmanın əsas prinsiplərini aşağıdakılar təşkil edir:

- alqoritm mərhələlər (addımlar) üzrə tərtib edilməlidir;
- mürəkkəb məsələ kifayət qədər sadə, asan qavranılan hissələrə parçalanmalı və onların hər birinin ancaq bir girişi və bir çıxışı olmalıdır;
- alqoritm mənəti kifayət qədər sadə olan minimal sayda idarəedici baza strukturlarına əsaslanmalıdır.

Alqoritm qurulmasına struktur yanaşma zamanı bütün alqoritmlər xətti (ardıcıl gəlmə), budaqlanan və dövrü (təkrarlanan) strukturlara ayrılırlar. Baza strukturlarına bir giriş və onlardan bir çıxış olur. Baza strukturlarını sxemlər vasitəsilə təsvir etmək üçün funksional blok anlayışını daxil etmək lazımdır.

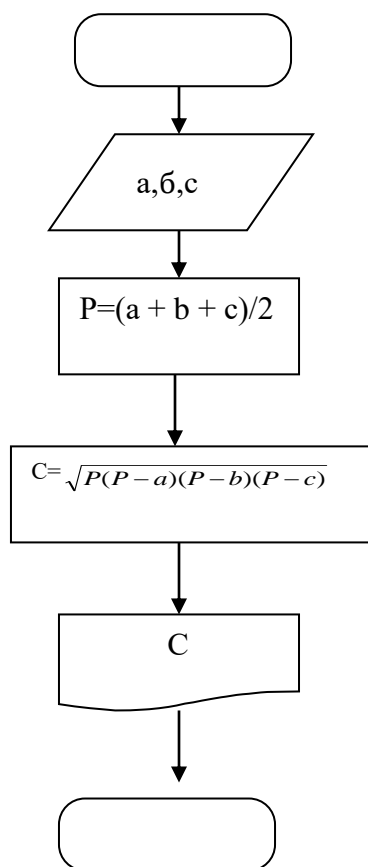
Funksional blok informasiyanın emalında əməllərin göstərilməsi üçün qrafik təsviri düzbucaqlı şəkildə olan blokdir. Bu əməllər ya mənimsətmə əmri, ya da bir girişi və bir çıxışı olan əməllər ardıcılığıdır. Əməllər düzbucaqlının içərisində yazılır.

Xətti alqoritmdə məntiqi şərtlər olmur və bir hesablama budağına malik olur. Hesablama budağı dedikdə hesablama istiqaməti nəzərdə tutulur. Xətti alqoritm bir-biri ilə əlaqəli bloklar ardıcılığı şəkildə təsvir olunur:



burada A_1, A_2, \dots, A_n müxtəlif əməliyyatdardır.

Misal: 1. Tərəfləri a, b, c olan üçbucağın sahəsini hesablamaq üçün alqoritm tərtib edək:



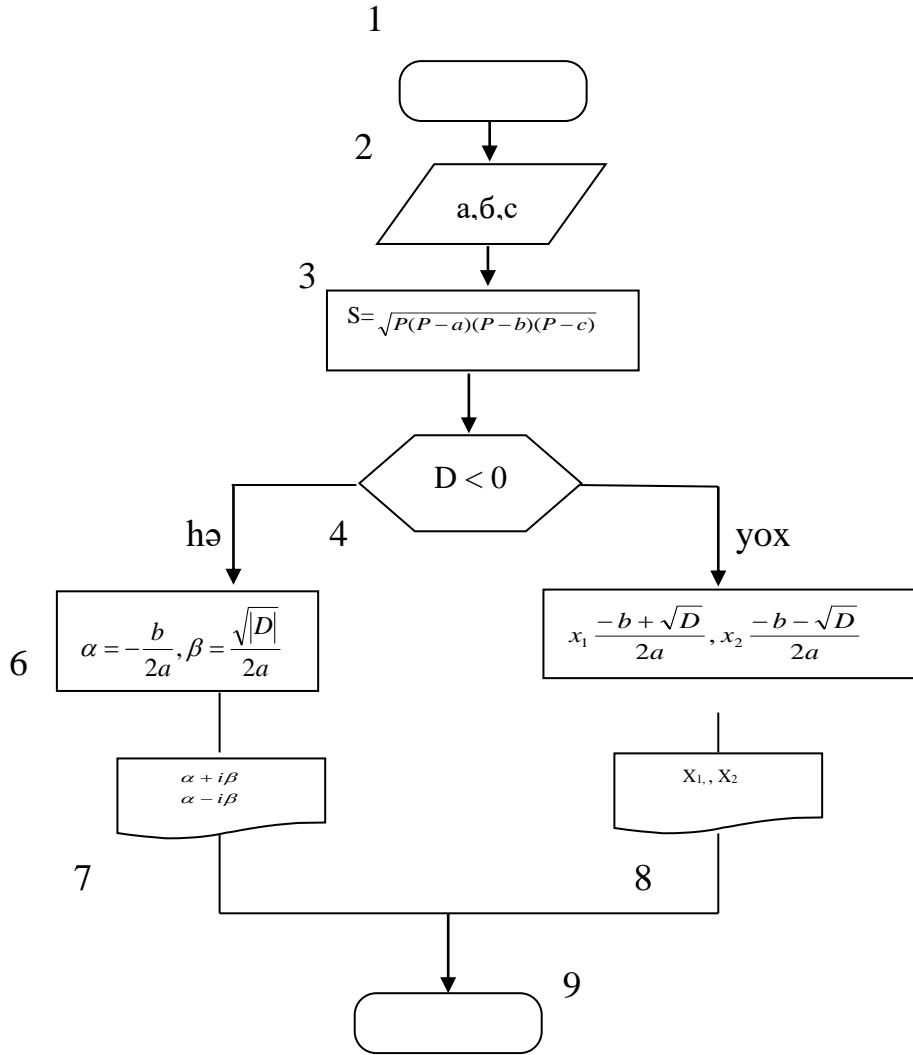
Budaqlanan alqoritmlər – tərkibində məntiqi blok olan hesablama prosesinin təsvir edir. Hər bir budaqlanma nöqtəsi uyğun məntiqi blokla təyin edilir. Bu blokda müəyyən kəmiyyətlərin (ilkin verilənlərin, aralıq nəticələrin və s.) bu və ya digər şərti ödəyib-ödəməməsi yoxlanılır və nəticədən asılı olaraq, bu və ya digər hesablama istiqaməti seçilir.

İki budaqdan ibarət olan prosesə sadə, ikidən çox budağı olan prosesə isə mürəkkəb budaqlanan struktur deyilir. Blok-sxemdə hər hansı şərtədən asılı olaraq, bütün hesablama istiqamətləri göstərilməlidir. Lakin alqoritmin icrası zamanı istiqamətlərdən yalnız biri üzrə hesablama aparılır.

Alqoritmlərin iki cür budaqlanma strukturu vardır:

Tam budaqlanma və natamam budaqlanma:

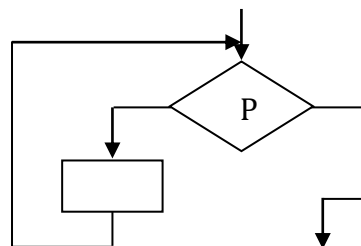
Misal . $ax^2 + bx + c = 0$ kv. tənliyinin həll alqoritmini verək:



Burada 4-cü blokda $D < 0$ şərti yoxlanır və nəticədən asılı olaraq ya 5-ci, ya da 6-cı blok seçilir. 5-ci blokda həqiq köklər, 6-cı blokda isə kompleks köklərin həqiqi və xəyali hissələri hesablanır.

Dövrü alqoritmik strukturlar. Təcrübədə çox rast gəlinən dövrü hesablama proseslərində məsələnin həlli eyni hesablama düsturları ilə dəyişənlərin müxtəlif qiymətləri üçün bir neçə dəfə təkrarən hesablamaların aparılmasını tələb edir. Hesablama prosesinin təkrar yerinə yetirilən hissəsinə dövr deyilir.

Dövrü strukturlar sadə və mürəkkəb hissəsinə bilər. Sadə struktur bir, mürəkkəb struktur isə bir-birinə daxil olan iki və daha çox dövrədən ibarət olur.

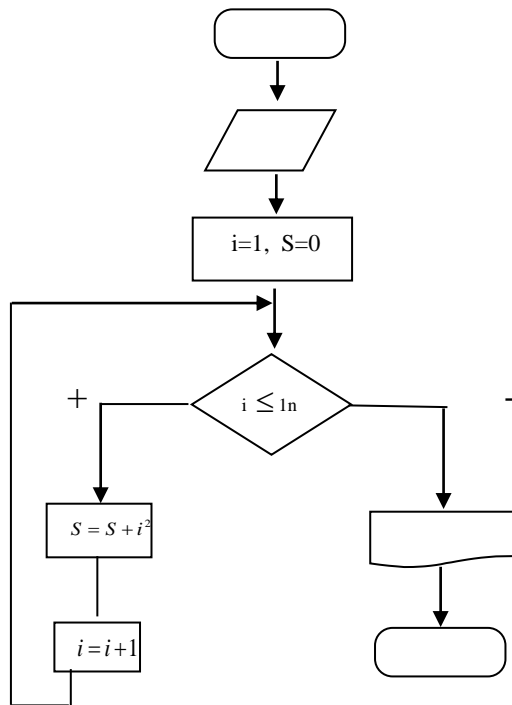


hə

yox

Dövr baza strukturuna məntiqi blok (şərtin yoxlanması bloku) və bir funksional blok (A bloku) daxildir. R şərti doğru olduqda A bloku yerinə yetirilir, R yalan olduqda isə dövrdən çıxış alınır. Əgər, R şərti elə hesablama prosesinin əvvəlində ödənilməz və (yalan olarsa) A bloku heç bir dəfə də yerinə yetirilməyəcək. Bu struktur ön şərtli dövr (dövr-hələ) adlanır.

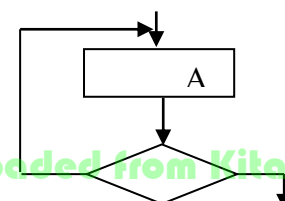
Tutaq ki, $S = \sum_{i=1}^n i^2$ cəminin hesablanmasına baxaq.



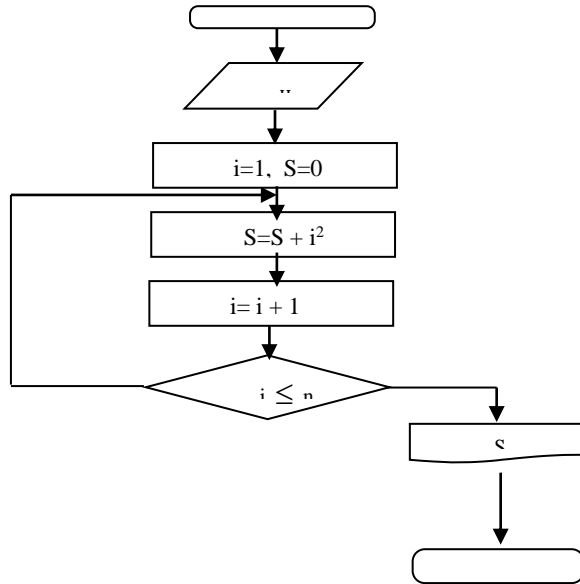
3 – cü blokda s dəyişənlərinə ilkin qiymətlər mənimsədilir. 4-cü blokda i parametrinin $n-i$ aşmış-şmadığı yoxlanılır. $i \leq n$ ödəndikdə $S = S + i^2$ mənimsədilir və $i = i + 1$ blokuna keçilir, yəni parametrin qiyməti bir vahid artırılır və şərtin yoxlanmasına qayıdır. Bu proses şərt pozulana qədər davam etdirilir.

Bundan əlavə daha iki dövr strukturundan istifadə olunur: son şərtli (dövr-qədər) dövr və parametrli dövr.

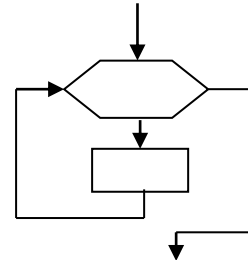
Son şərtli dövr. Bu strukturda təkrarlanma şərti A blokundan sonra yerləşir. Bu halda R şərti dövrün başa çatması şərti adlanır.



Burada A bloku məntiqi bloktan əvvəl yerləşdiyindən heç olmasa bir dəfə yerinə yetiriləcək. Qeyd edək ki, dövrün gövdəsi adlanan A blokunda şərtin parametrini dəyişən hər hansı bir əmr (blok) olmalıdır. $S = \sum_{i=1}^n i^2$ cəminin hesablanması üçün son şərtli dövrü alqoritm tərtib edək:



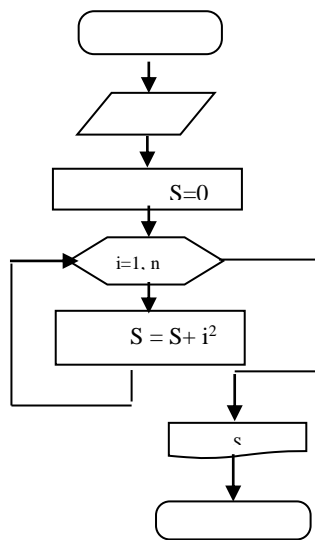
Parametrlı dövr strukturu dövrün təkrarlanmaları sayı əvvəlcədən məlum olduqda əlverişlidir. Dövrün başlanğıcında parametrin ilk və son addımı, parametrin ilk və son qiymətləri göstərilir.



Məsələn : $i = 1, n, h$ bu o deməkdir ki, i parametri 1-dən n -ə qədər h addımı ilə dəyişir.

$S = \sum_{i=1}^n i^2$ misalına müraciət edək. Burada addım 1-ə bərabərdir. Qeyd edək ki, addım 1-ə bərabər olduqda onu blokda göstərməmək də olar.

Mürəkkəb dövrlər. Praktiki məsələlər əksər hallarda daxilində dövrü strukturlar olan mürəkkəb dövrü proseslər şəklində verilir. Daxili dövrlər ya müstəqil



şəkildə, ya da biri digərinin içərisində xarici dövrə daxil ola bilərlər. Dövrələrin birinin digərinə daxil olmasının sayı məhdudlaşdırılmır. Sadə dövrü strukturları kombinasiya edərək lazımi mürəkkəb struktur almaq olar. Bu zaman aşağıdakılar nəzərə alınmalıdır :

- daxili dövrlərin, parametrlərinin başlanğıc qiymətləri dövrə daxil olana qədər hazırlanmalıdır;
- daxili dövrədən xarici dövrə çıxış daxil bitdikdən sonra və ya hər hansı şərtin ödənməsi nəticəsində yerinə yetirilir;
- xarici dövrədən daxili dövrə yalnız onun başlanğıcı vasitəsilə daxil olmaq mümkündür;
- xarici və daxili dövrlərin parametrlərinin eyni adlandırılmasına yol vermək olmaz

MƏSƏLƏNİN KOMPÜTERDƏ HƏLLİ MƏRHƏLƏLƏRİ

Kompüterdə məsələnin həlli aşağıdakı mərhələlərdə aparılır:

1. Məsələnin qoyuluşu. Məsələnin qoyuluşu aşağıdakıları nəzərdə tutur: ilkin verilənlərin siyahısı, tipi, dəqiqliyi və ölçüləri, dəyişənlərin dəyişmə oblastları, başlanğıc və sərhəd şərtləri, nəticələrin siyahısı, tipi, dəqiqliyi və ölçüləri, məsələnin həllini təmin edən düsturlar və tənliklər.
2. Həll algoritminin yaradılması. Bu mərhələdə çəkilən həll üsuluna uyğun olan həll algoritmi tərtib edilir. Məsələnin həlli ayrı-ayrı müstəqil bloklara bölünür və həmin blokların yerinə yetirilmə ardıcılığı təyin edilir. Nəticədə algoritmin blok-sxemi qurulur.

3. Verilənlərin strukturlarının təyini. Bu mərhələdə alqoritmə iştirak edən verilənlərin tipinə, formasına, mümkün qiymətlərinə və aparılan əməliyyatlara görə onların strukturları seçilir. Yəni verilənlərin tam, həqiqi, simvol və s. tipli olması müəyyənləşdirilir.
4. Proqramlaşdırma dilinin seçilməsi və ilkin proqramın tərtibi. Hazırda proqramlaşdırma üçün müxtəlif dillər mövcuddur. Həll olunan məsələnin xarakterinə, tətbiq olunan kompüter üçün mövcud olan translyatorlara, proqramçının hazırlıq səviyyəsinə görə proqramlaşdırma dili seçilir. Sonra məsələnin həlli alqoritmi əsasında seçilən dildə proqram tərtib edilir. Ona ilkin proqram deyilir.
5. İlkin proqramın kompüter dilinə çevrilməsi və sazlanması. Kompüter dilində proqram alqoritminə uyğun əmrlər ardıcılığıdır. Bu mərhələdə proqramlaşdırma dilində yazılmış ilkin proqram kompüter dilinə çevrilir. Bu iş translyator adlanan proqram vasitə ilə yerinə yetirilir. Bu zaman ilkin proqramda buraxılmış morfoloji və sintaksis səhvləri aşkar edilib proqramçıya çatdırılır. Səhlər aradan qaldırıldıqdan sonra tərcümə prosesi davam etdirilir və kompüter dilində proqram alınır. Bu proqram mütləq proqram və ya işçi proqram adlanır. Proqramdakı məntiqi səhlər aşkarlanır və aradan qaldırılır, bu proses proqramın sazlanması adlanır.
6. İşçi proqramın icrası, nəticənin alınması və təhlili. Proqram sazlandıqdan sonra ondan tətbiqi məsələnin həlli üçün istifadə etmək olar. Bu zaman proqram müxtəlif ilkin verilənlər yığını üçün bir neçə dəfə icra olunur. Alınan nəticələr mütəxəsis və ya məsələni qoyan istifadəçi tərəfindən təhlil olunur. Uzun müddət istifadə olunan proqram kompüterin xarici yaddaşında (diskdə) hazır proqram kimi saxlanır.

Proqramlaşdırma dillərinin təsnifatı

Proqramlaşdırma texnologiyasında əsasən aşağıdakı üslublardan istifadə olunur:

- prosedur proqramlaşdırma
- funksional proqramlaşdırma
- məntiqi proqramlaşdırma
- obyektönlü proqramlaşdırma

Prosedur proqramlaşdırma. Prosedur proqramlaşdırma 1940-ci ildə Fon Neyman tərəfindən təklif olunan kompüterin arxitekturasına əsaslanır və onun nəzəri modeli kimi «*Türinq maşını*» adlanan alqoritmik sistem götürülmüşdür.

Prosedur proqramlaşdırma dilində proqram operatorlar ardıcılığmdan ibarətdir. Burada əsas operator, yaddaş sahəsinin məzmununu dəyişən mənimsətmə operatorudur.

Prosedur dil aşağıdakı xüsusiyyətlərlə xarakterizə olunur:

- yaddaşın idarə olunması vacibliyi, xüsusən dəyişənlərin təsviri;
- simvolların emalı üçün imkanların məhdudluğu;
- ciddi riyazi əsasın olmaması;
- müasir kompüterdə yüksək səmərəlilik reallaşdırma.

Prosedur dilin əsas tənifat əlamətlərindən biri onun səviyyəsidir.

Proqramlaşdırma dilinin səviyyəsi onun konstruksiyasının semantik ölçüsü və onun proqramçıya yönümü dərəcəsi ilə təyin olunur. Səviyyənin artma dərəcəsi ilə dillərdən bir neçəsini göstərək: İkilik dil bilavasitə maşın dilidir.

Assembler dili - maşın dilinə yaxın olub, maşın əmrlərinin simvolik formada təsvirini təmin edir.

Makroassembler dili - Assembler dilinə makro vasitələr daxil edilməsi ilə alınan dildir.

Peşəkar sistem proqramçıları kompüterin bütün qurğularından istifadə etmək üçün Assembler və Makroassembler dilindən istifadə edirlər. Bu dillərdən əsasən sistem proqramı təminatının tərkibinə daxil olan - drayver, utilit və s. proqramların yaradılmasında istifadə olunur.

S dili ilk dəfə 1970-ci ildə Unix əməliyyat sistemini reallaşdırmaq üçün yaradılmışdır.

Basic dili 1965-ci ildə proqramlaşdırmanı yeni öyrənməklə üçün yaradılmışdır.

Pascal dili prosedur proqramlaşdırma dilinin içərisində çox istifadə olunan dildir. Bu dil 1970-ci ildə İsveçrəli Niklou Virt tərəfindən yaradılmışdır. Pascal dilində proqramın bir sıra konsepsiyaları yaradılmışdır.

Funksional proqramlaşdırma. Funksional proqramlaşdırmanın mahiyyəti A.P.Erşov tərəfindən təyin olunmuşdur. Funksional dilin konstruksiyasında ifadə əsas

rol oynayır. İfadəbrə skalyar sabitbr, strukturlaşdırılmış obyektbr, funksiyalar, funksiyaların gövdəsi və funksiyaların çağırılması aiddir.

Funksional proqramlaşdırma dilinə aşağıdakı elementbr daxildir:

- funksiyaların manipulyasiya edə bildiyi sabitbr sinfi;
- proqramçının əvvəldən təsvir etmədən istifadə etdiyi baza funksiyalar yığımı;
- baza funksiyalardan yeni funksiyaların tərtibi qaydası;
- çağırılan funksiyalar əsasmda ifadələrin yaradılma qaydası Funksional proqramlaşdırmanın ilk dili LİSP (List Processing - siyahıların

email) dilidir. LİSP dili 1959-cu ildə Massaçusets texnologiya institutunda Con Makkarti tərəfindən yaradılmışdır. Bu dilin yaradılmasında əsas məqsəd simvol tipli informasiyanın emalını əlverişli təşkil etmək olmuşdur. Dilin əsas xüsusiyyəti proqram və veribnəbrin strukturunun unifikasiyasıdır, yəni bütün ifadəbr siyahı şəklində yazılır.

Məntiqi proqramlaşdırma. Məntiqi proqramlaşdırma Prolog (Programming in logic - məntiqi terminbrb proqramlaşdırma) dilinin meydana gəlməsinə səbəb oldu. Bu dil 1973-cü ildə fransız alimi A. Kolmerol tərəfindən yaradılıb. Hazırda bir çox məntiqi proqramlaşdırma dili mövcuddur, lakin Prolog dili qə çox inkişaf etmiş və yayılmış dildir. Məntiqi proqramlaşdırma dilləri, xüsusən Prolog, süni intellekt sistemlərində geniş istifadə olunur. Məntiqi proqramlaşdırmanın əsas anlayışı münasibətdir. Proqram obyekt və məqsəd arasındakı münasibətin təyinindən təşkil olunur. Məntiqi proqramlaşdırmada yalnız alqoritmə əsaslanan faktlfrən spesifik xüsusiyyətbrini göstərmək lazımdır. Burada yerinə yetirilməsi təbb olunan addımlar ardıcılığını təyin etmək lazım deyil.

Məntiqi proqramlaşdırma aşağıdakılara görə xarakterizə olunur:

- yüksək səviyyə;
- simvol hesabatma istiqamətləndirmə;
- tərsinə (intensiv) hesablama imkanı, yəni prosedurlardakı dəyişənlər giriş və çıxışa ayrılır;
- məntiqi natamalığın mümkünlüyü, çünki proqramda müəyyən məntiqi münasibətbri əks etdirmək, həmçinin bütün nəticəbrin düzgün alınması mümkün deyil.

Məntiqi proqramm prinsipcə çox da olmayan sürətə malikdir. Bəb ki, hesablama əvvəlki addıma qayıtmaq şərti ib axtarış, smaq və səhvbr üsulu ib həyata keçirilir.

Obyektivönlü proqramlaşdırma. Obyektivönlü proqramlaşdırmanın bir çox vasitələri Simula-67 dilindən götürülmüşdür.

Proqramlaşdırmanın obyektivönlü üslubu obyekt anlayışına əsaslanır, mənası isə «**obyekt - verilənlər + prosedurlar**» düsturu ib ifadə olunur. Hər bir obyekt veribnbrin strukturunu birbşdirir və onlara müraciət bu veribnbrin email proseduru ib mümkündür ki, bu da metod adlanır.

Veribnlər və prosedurun bir obyektə birləşməsi inkapsulyasiya adlanır. Obyektbrin təsvirinə siniflər xidmət edir. Sinif bu sinfə aid olan obyektbrin xassə və metodlarını təyin edir. Uyğun olaraq, istənilən obyekt sinfin nüsxəsini təyin edə bilər. Müasir obyektivönlü proqramlaşdırma dillərinə Smalltalk, C++, Object Pascal, Java və s. aiddir.

C++ dilini 80-ci ilin əvvəlbrində **AT & T** korporasiyasının Bell laboratoriyasının əməkdaşı V. Strastrup təklif etmişdir. Bu dilin İnternetdəki versiyası Java adlandı. Java və S++ dilbrin arasındakı prinsiplial fərq ondan ibarətdir ki, birinci dil interpretasiya, ikinci isə kompilyasiya olunur.

Obyektivönlü proqramlaşdırma ideyası bir çox universal prosedur dilbrdə də istifadə olunur. Məsələn, Pascal proqramlaşdırma dilinin 5.5 versiyasından başlayaraq inteqrallaşdırılmış sistemə xüsusi Turbo Vision obyektivönlü proqramlaşdırma kitabxanası daxil edilib. Son zamanlar bir çox proqramlar, xüsusən obyektivönlü vizual proqramlaşdırma sistembrində reallaşdırılır. Obyektivönlü vizual proqramlaşdırma sisteminə Vizual Basic, Delphi, S++ Builder və Visual S++ və s. aid etmək olar.

Turbo PAscal dilinin elementləri.

İstənilən təbii dil simvollar, söz, sözbirləşmələri və cümlələrdən ibarətdir. Proqramlaşdırma dillərində də buna analogi elementlər var. Bunlar simvollar, sözlər,

ifadələr (sözbirləşmələri) və operatorlardır(cümlələr). Bunlar danışq dili olmayan Turbo Pascal proqramlaşdırma dilinə də aiddir. Dilin *əlifbasına* aşağıdakılar aiddir:

1. A-dan Z-ə kimi böyük və a-dan z-ə kimi kiçik latın hərfləri
2. 0-dan 9-a kimi onluq rəqəmlər
3. münasibət işarələri : = , < , > , <= , >= , < >
4. hesab işarələri : + , - , * , /
5. altından xətt simvolu « - » və boşluq (probel) « »
6. xüsusi simvollar: # ,) , @ , { , \$, [, } , ' , (, ^
7. durğu işarələri : , | : | ; |
8. İdarəedici simvollar (ASCII kodu 0-dan 31-ə kimi)

9. Xidməti (açar) sözlər. Bütün proqramlaşdırma dillərində olduğu kimi Turbo Pascal dilində xidməti sözlər öz vəzifəsinə görə dəqiq təyin edilərək, dəyişdirilə bilməz. Buna görə də identifikatorun yazılışında xidməti sözlərdən istifadə etmək olmaz. Bunlardan : begin, end, read, write, if və s.

İdentifikatorlar. İdentifikator rəqəm, hərf və altından xətt çəkmə «- » işarələrindən ibarət olmaqla, hərf və ya «- » işarələri ilə başlayır. İdentifikatorda probeldən istifadə olma bilməz, uzunluğu isə ixtiyari ola bilər, kompüter onun ilk 63 simvolunu oxuyur.

Nişanlar. Nişan operatora verilən addır. Turbo Pascalda nişanlar ədəd və simvollar ola bilərlər. Nişan kimi identifikatordan da istifadə oluna bilər və o, operatorndan « : » işarəsi ilə ayrılır.

Ədədlər. Turbo Pascalda tam onluq, tam onaltılıq və həqiqi onluq ədədlərdən istifadə olunur. Həqiqi ədədlər iki yazılış formasında təsvir olunur: adi(qeyd olunmuş nöqtəli) və eksponensial (tərtibli, yaxud sürüşən nöqtəli). Tam onluq ədədlər standart şəkildə yazılır, tam onaltılıq ədədlərin isə qarşısında \$ işarəsi qoyulur. Məsələn, -273, 468213, \$ 0, \$ E35, \$ 13A4C

Sətirlər. Sətirlər birqat dırnaq işarəsinə alınmış ASCII kodunun simvollar ardıcılığından təşkil olunur. Sətirlər proqramın bir sətirinə yerləşdirilməlidir.

Şərhlər. Şərh və ya (*,*) simvolları arasında olan proqramın mətni fraqmentidir ki, kompilyator tərəfindən nəzərə alınmır.

Ayırıcılar. Turbo Pascalda ayırıcı kimi aşağıdakı simvollarndan istifadə olunur:

- probel, tabulyasiya, növbəti sətirin başlanğıcına keçid (Enter,)

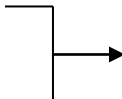
Turbo Pascalda bunlardan başqa ASCII-nin 0 ilə 31 kodu arasındakı idarəedici simvollardan da ayırıcı kimi istifadə olunur.

TP- Programın strukturu

Turbo Paccal dilində proqramın sstrukturunu aşağıdakı kimi vermək olar:

```
Proqram proqramın adı;           {Proqramın başlığı}  
{İstifadə olunan modulların təsviri bölməsi}  
Uses İstifadə ollunan modulların siiyahısı;  
    {Təsvirlər bölməsi}
```

```
label      Nişanların təsviri;  
const     Sabitlərin təsviri;  
type      Tiplərin təsviri;  
var       Dəyişənlərin təsviri;
```

```
Procedure  Altproqramların təsviri;  
function  
exports   Eksport edilən adların təsviri;  
begin
```

```
    operator 1;  
    operator 2;  
    .....  
    operator n;  
end
```

Standart Pascaldan fərqli olaraq TP-da proqramın başlığı və istifadə olunan modulların təsviri bölməsi istisna olmaqla bölmələrin yerləşmə ardıcılığı dəyişmək olar, lakin standartda nəzərdə tutulan ardıcılığa riayət etmək məqsədəuyğundur. Yalnız prinsipcə vacib olduqda ardıcılığı dəyişmək olar.

Proqramın gövdəsi «begin» sözü ilə başlayır və «end» sözü ilə qurtarır, bu «end»-dən sonra nöqtə qoyulur.

Misal: $z = \frac{x^2 + y^2}{2xy}$ ifadəsinin hesablamasının Pascal proqramı:

```
Proqramın İfadənin_ hesablanması;
var
    x, y, z: real; {Dəyişənlərin təsviri}
begin
    {Proqramın gövdəsinin başlanğıcı }

Write (' x, y ədədlərini daxil edin' ); {verilənlərin ekrana çıxarılması}
Read ln (x, y) ; {x, y –in qiymətlərinin oxunması }

Z := (x * x + y * y) / ( 2 * x * y); {ifadənin hesablanması }

Writeln (' z = ' , z)   {nəticənin ekrana çıxarılması }

end
    {proqramın sonu }
    Programlaşdırma dillərinin təsnifatı.
```

Proqramlaşdırma dilləri aşağıdakı əsas növlərə bölünür:

- prosedur proqramlaşdırma;
- funksional proqramlaşdırma;
- məntiqi proqramlaşdırma;
- obyektönlü proqramlaşdırma.

Prosedur proqramlaşdırma dilində proqram operatorlar ardıcılığından ibarətdir. Belə dillər maşinyönlü və alqoritmik dillər olmaqla iki yerə bölünürlər. Maşinyönlü dillərdən İkilik dil (maşın dili), assembler dili və makroassembler dilini göstərmək olar. Alqoritmik prosedur dillərinə misal olaraq aşağıdakıları göstərmək olar:

C dili 70-ci ilin əvvəlində UNIX əməliyyat sistemini reallaşdırmaq üçün yaradılmışdır.

Basic - (Beginers All-purpose Symbolic Instruction Code) 1964-cü ildə proqramlaşdırmanı yeni öyrənənlər üçün yaradılmışdır.

Pascal dili prosedur proqramlaşdırma dilləri içərisində ən çox istifadə olunan dildir. Pascal dili 1970-ci ildə hesablama texnikası sahəsi üzrə ixtisasçı isveçrəli professor Niklou Virt tərəfindən yaradılmışdır.

Funksional proqramlaşdırmanın mahiyyəti A.P. Yerşov tərəfindən verilmişdir. Belə dillərdə skalyar sabitlər, strukturlaşdırılmış obyektlər, funksiyalar, funksiyaların gövdəsi və çağırılması kimi obyektlərin aid olduğu ifadələr əsas rol oynayır. Funksional proqramlaşdırmanın ilk dili LİSP(List Prossesing - siyahıların emalı) 1959-cu ildə ABŞ-da yaradılmışdır.

Məntiqi proqramlaşdırma süni intellekt sistemlərinin yaradılması zərurətindən meydana gəldi. Məntiqi proqramlaşdırma dillərindən PROLOG (Programming in Logic – məntiqi terminlərlə proqramlaşdırma) dili 1972-cu ildəfransız alimi A.Kolmerol tərəfindən yaradılıb.

Obyektyönlü proqramlaşdırmanın bir çox vasitələri Simula – 67 dilindən götürülmüşdür. Belə dillər obyekt anlayışına əsaslanır və «obyekt = verilənlər + prosedural» dusturu ilə ifadə olunur. Müasir obyektyönlü proqramlaşdırma dillərinə Smalltalk, C++, Object Pascal , Java və s.aiddir.

Turbo PAscal dilinin elementləri.

İstənilən təbii dil simvollar, söz, sözbirləşmələri və cümlələrdən ibarətdir. Proqramlaşdırma dillərində də buna analoji elementlər var. Bunlar simvollar, sözlər, ifadələr (sözbirləşmələri) və operatorlardır(cümlələr). Bunlar danışiq dili olmayan Turbo Pascal proqramlaşdırma dilinə də aiddir. Dilin əlifbasına aşağıdakılar aiddir:

9. A-dan Z-ə kimi böyük və a-dan z-ə kimi kiçik latın hərfləri

10. 0-dan 9-a kimi onluq rəqəmlər

11. münasibət işarələri : = , < , > , <= , >= , <>

12. hesab işarələri : + , - , * , /

13. altından xətt simvolu « - » və boşluq (probel) « »

14. xüsusi simvollar: # ,) , @ , { , \$, [, } , ‘] , (, ^

15. durğu işarələri : , | · | : | ; |

16. İdarəedici simvollar (ASCII kodu 0-dan 31-ə kimi)

9. Xidməti (açar) sözlər. Bütün proqramlaşdırma dillərində olduğu kimi Turbo Pascal dilində xidməti sözlər öz vəzifəsinə görə dəqiq təyin edilərək, dəyişdirilə bilməz. Buna görə də identifikatorun yazılışında xidməti sözlərdən istifadə etmək olmaz. Bunlardan : begin, end, read, write, if və s.

İdentifikatorlar. İdentifikator rəqəm, hərf və altından xətt çəkmə «- » işarələrindən ibarət olmaqla, hərf və ya «- » işarələri ilə başlayır. İdentifikatorda probeldən istifadə oluna bilməz, uzunluğu isə ixtiyari ola bilər, kompüter onun ilk 63 simvolunu oxuyur.

Nişanlar. Nişan operatora verilən addır. Turbo Pascalda nişanlar ədəd və simvollar ola bilərlər. Nişan kimi identifikatordan da istifadə oluna bilər və o, operatorndan « : » işarəsi ilə ayrılır.

Ədədlər. Turbo Pascalda tam onluq, tam onaltılıq və həqiqi onluq ədədlərdən istifadə olunur. Həqiqi ədədlər iki yazılış formasında təsvir olunur: adi(qeyd olunmuş nöqtəli) və eksponensial (tərtibli, yaxud sürüşən nöqtəli). Tam onluq ədədlər standart şəkildə yazılır, tam onaltılıq ədədlərin isə qarşısında \$ işarəsi qoyulur. Məsələn, -273, 468213, \$ 0, \$ E35, \$ 13A4C

Sətirlər. Sətirlər birqat dırnaq işarəsinə alınmış ASCII kodunun simvollar ardıcılığından təşkil olunur. Sətirlər proqramın bir sətirinə yerləşdirilməlidir.

Şərhlər. Şərh və ya (*,*) simvolları arasında olan proqramın mətni fraqmentidir ki, kompilyator tərəfindən nəzərə alınmır.

Ayırıcılar. Turbo Pascalda ayırıcı kimi aşağıdakı simvoldan istifadə olunur:

- probel, tabulyasiya, növbəti sətirin başlanğıcına keçid (Enter,)

Turbo Pascalda bunlardan başqa ASCII-nin 0 ilə 31 kodu arasındakı idarəedici simvoldan da ayırıcı kimi istifadə olunur.

TP- Proqramın strukturu

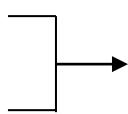
Turbo Paccal dilində proqramın ststrukturunu aşağıdakı kimi vermək olar:

Proqram proqramın adı; {Proqramın başlığı}
{İstifadə olunan modulların təsviri bölməsi}

Uses İstifadə ollunan modulların siyahısı;

{Təsvirlər bölməsi}

label Nişanların təsviri;
const Sabitlərin təsviri;
type Tiplərin təsviri;
var Dəyişənlərin təsviri;

Procedure  Altproqramların təsviri;
function
exports Eksport edilən adların təsviri;
begin

operator 1;
operator 2;
.....
operator n;

end

Standart Pascaldan fərqli olaraq TP-da proqramın başlığı və istifadə olunan modulların təsviri bölməsi istisna olmaqla bölmələrin yerləşmə ardıcılığı dəyişməkdir, lakin standartda nəzərdə tutulan ardıcılığa riayət etmək məqsədəuyğundur. Yalnız prinsipcə vacib olduqda ardıcılığı dəyişməkdir.

Proqramın gövdəsi «begin» sözü ilə başlayır və «end» sözü ilə qurtarır, bu «end»-dən sonra nöqtə qoyulur.

Misal: $z = \frac{x^2 + y^2}{2xy}$ ifadəsinin hesablamasının Pascal proqramı:

Proqramın İfadənin_ hesablanması;

var

x, y, z: real; {Dəyişənlərin təsviri}

```

begin
                                {Proqramın gövdəsinin başlanğıcı }

Write ( ' x, y ədədlərini daxil edin' ); {verilənlərin ekrana çıxarılması}
Read ln (x, y) ; {x, y –in qiymətlərinin oxunması }

Z := (x * x + y * y) / ( 2 * x * y); {ifadənin hesablanması }

Writeln ( ' z = ' , z)   {nəticənin ekrana çıxarılması }

end                                {proqramın sonu }
                                Verilənlərin tipləri

```

TP dilindəki tipləri iki qrupa bölmək olar:

- standart tiplər;
- istifadəçi tipləri.

Standart tiplərin adları əvvəldən təyin olunmuş identifikatorlardır və proqramın istənilən yerində iştirak edə bilər.

İstifadəçi tipləri – əlavə abstrakt tiplərdir ki, onları istifadəçi-proqramçı sərbəst təyin edir.

Tiplərin sintaksisi aşağıdakı kimi təsvir olunur:

Type identifikator = tip;

Standart tiplər. TP-də standart tiplərə aşağıdakılar aiddir:

1. Tam tiplər:

Adı	İdentifikator	Yaddaş ölçüsü
İşarəli qısa tam	Shortint	1 bayt
İşarəli tam	Integer	2 bayt
İşarəli uzun tam	Longint	4 bayt
İşarəsiz qısa tam	Byte	1 bayt
İşarəsiz tam	Word	2 bayt

2. Həqiqi tiplər

Adı	İdentifikator	Yaddaş ölçüsü
Birqat dəqiqlikli həqiqi	Single	4 bayt
Həqiqi	Real	6 bayt
İkiqat dəqiqlikli həqiqi	Double	8 bayt
Yüksək dəqiqlikli həqiqi	Extended	10 bayt
Tam həqiqi formatda	Comp	8 bayt

3. Məntiqi (Bul) tip:

Adı	False	True
Boolean- 1 bayt	0	≠ 0
Byte Bool — 1 bayt	0	≠ 0
Word Bool — 2 bayt	00	≠ 0
Long Bool — 4 bayt	0000	≠ 0

- 4. Simvol tipi - Char
- 5. Sətir tipləri - String, Pchar
- 6. Göstərici tipi - Pointer
- 7. Mətn tipi - Text

Standart funksiyalar

Pascal dilində ifadələrdə hazır element kimi istifadə olunan əvvəlcədən hazırlanmış altproqram-funksiyalar mövcuddur:

1. Hesabi funksiyalar:

$ x $ - abs (x)	e^x - exp (x)
x^2 - sqr (x)	$\ln x$ - ln (x)
\sqrt{x} - sqrt (x)	ədədin kəsr hissəsi - frac (x)
arctg x - arctan (x)	ədədin tam hissəsi - int (x)
cos - cos (x)	π - Pi
sin x - sin (x)	

2. Tipin çevrilməsi funksiyaları:

- chr (x) - ASCII kodunu simvolda çevirir chr (97)
- high (x) - kəmiyyətin maksimal qiymətini verir;
- low (x) - kəmiyyətinin minimal qiymətini verir;
- ord (x) - sıra tipini tam tipə çevirir; ord ('a') → 97

round (x) – həqiqi ədədin qiymətini ona yaxın tama yuvarlaşdırır;

trunc (x) – həqiqi ədədin tam hissəsi

3.Sıra tipi kəmiyyətləri üçün f-yalar:

odd(x) – x-in tək olmasını yoxlayır, x-təkcə nəticə true, çüt olduqda isə false olur

pred (x) – x-dən əvvəlki qiymət

succ (x) – x-dən sonrakı qiymət

Turbo Pascal-da adi hesab əməliyyatlarından əlavə div və mod əməliyyatlarından da istifadə olunur.

div - tam bölmə, mod – bölmədən alınan qalıqı göstərir.

Məsələn,

$$5 \text{ div } 6 = 0; \quad 5 \text{ mod } 6 = 5;$$

$$12 \text{ div } 4 = 3; \quad 12 \text{ mod } 4 = 0$$

$$15 \text{ div } 4 = 3 \quad 15 \text{ mod } 4 = 3$$

$$(12 \text{ div } 5) * 2 + (13 \text{ div } 2) + (13 \text{ mod } 5) = 2 * 2 + 6 + 3 = 13$$

İfadələr.Dəyişənlər.Sabitlər.

İfadələr. Ədədi ifadələr ədədlərdən, əməl işarələri və mötərizələrin köməyi ilə düzəldilir. Məsələn $2+6,9*6+4...$ Ədədi ifadələrdə əllər yerinə yetdikdə alınan sonlu ədəd ədədi ifadənin qiyməti adlanır.

Dəyişənli ifadələr ədəd və dəyişənlərdən əməl işarələri və mötərizələrin köməyi ilə düzəlir. İstənilən dəyişənli ifadədən onun xüsusi halı olan sonsuz sayda ədədi ifadə alınır. Məsələn: $3*x+69$ dəyişənli ifadədə x-ə verilən müxtəlif ədədi qiymətlər aldıqda müxtəlif ədədi ifadələr alınır.

Pascal alqoritmik dilində ifadələr soldan sağa bir və ya bir neçə sətirdə ardıcıl yazmaqla konstantlardan, dəyişənlərdən, standart və ya proqramçı tərəfindən təyin edilmiş funksiyalardan, məntiq əməllərindən, simvollar üzərində əməllərdən, dairəvi mötərizələr və hesab əməl işarələrindən istifadə etməklə tərtib edilir.

Hesabi ifadələr hesabi konstant, dəyişən və funksiyalarda əməl işarələri və mötərizələrə köməyi ilə düzəlir. Eyni tipli verilənlərdən təşkil edilən hesabi ifadənin tipi

verilənlərin tipi ilə eyni olur. Giriş verilənləri tam və həqiqi tipli olduqda nəticədə həqiqi tipli çıxış verilən alınır. Pascalda ifadələr aşağıdakı şərtlər əsasında düzəlidir.

1. İfadənin tərkib hissələri eyni sətirdə yazılır. Aşağı və yuxarı indekslərdən istifadə edilmir.
2. ifadələrin yazılışında ancaq kiçik mötərizədən istifadə olunur, neçə mötərizə açılırsa o qədər mötərizə bağlanmalıdır.
3. ifadələrin yazılışı zamanı istənilən iki əməl əməliyyat işarəsinin ardıcıl yazılması yol verilməzdir.
4. hesabi ifadələrin qiyməti hesablanarkən əməliyyatlar soldan sağa olmaqla müəyyən ardıcılıqla yerinə yetirilir.
 - Mötərizə daxili qiymət hesablanır
 - Funksiyanın qiyməti hesablanır
 - Not inkar əm.yer.yet.
 - * və / əm. Yer.yet.
 - Tam bölmə(Div),qalığın tapılması(Mod) funksiyaları və And məntiqi vurma əməli
 - +, - və Or məntiqi əm.
 - Münasibət əm.

Məntiqi ifadələr. Bu ifadələr məntiqi konstant və dəyişənlərdən məntiq əməlləri, münasibət işarələri və mötərizələrin köməyi ilə düzəlidir. Bu ifadələr üzərində əməllərin nəticəsində ya doğru(True) ya da yalan(False) qiymətlərindən birini alır.

Məntiqi ifadələrin yazılışında əməliyyatlar aşağıdakı ardıcılıqla yerinə yetirilir.

1. Not inkar əməliyyatı
2. *,/,Div,Mod,And əm.
3. +,-,Or,Xor əməlləri
4. məntiq əməlləri

Burada ancaq kiçik mötərizədən istifadə olunur. əvvəlcə mötərizənin içi sonra isə xaricindəki əməl. Yerinə yetirilir. And və Or əməllərindən əvvəl və sonra yazılan ifadələrmötərizələr daxilində yazılır. Məsələn

(A and B) Or (Not A)

DƏYİŞƏNLƏR.

Proqramda istifadə olunan dəyişənlər dəyişənlərin təsviri bölməsində təsvir edilir.

Dəyişənləri təsvir etmək üçün

Var

Dəyişənin adı : dəyişənin tipi; yazılışından istifadə olunur. Burada

Var(variable)-dəyişən mənasını

Dəyişənin adı-istifadə edilən bir-birindən “,” ilə ayrılan dəyişənləri

Dəyişənin tipi isə onun hansı tipə aid olduğunu bildirir

Məsələn proqramda is. Edilən tam tipli n dəyişəni bu cür təsvir olunur

Var

N:integer;(*tam tipli dəyişən*)

Proqramda bir neçə eyni tipli dəyişəndən istifadə olunursa ,onları bir-birindən vergüllə ayırmaqla eyni sətirdə yazmaq olar.

Var n,x,s : integer;

Dəyişənin təsviri hissəsində müxtəlif tip dəyişənləri eyni var açar sözündən sözündən istifadə etməklə siyahı ilə vermək olar. Məsələn

Var

Z: integer; (*tam tipli*)

X: real; (*həqiqi tipli*)

S : char; (*simvol tipli*)

F: boolean (*məntiqi tipli*)

Verilənlərin dəyişən tipi. Turbo Pascalada ixtiyari verilən (sabit,dəyişən,ifadə)öz tipi ilə xarakterizə olunur.Tip obyektin ala biləcəyi mümkün qiymətlər çoxluğunu və onlar üzərində yerinə yetiriləcək əməliyyatları təyin edir. Tip təsvirlər bölməsində aşağıdakı şəkildə verilir.

Type Tipin adı = tip;

Burada Type-tip mənasına uyğun gəlir.

Tipin adı –daxil edilən tipinadını(identifikatoru)

Tip isə daxil edilən tipin(nizamlı,həqiqi və s.)adını bildirir.

Məsələn : Type

Int=integer;

B=boolean;

İstifadəçinin tipləri. Proqramçı bəzi məsələlər üçün proqram tərtib edərkən onun məzmununa uyğun olaraq yeni tip verilənlərdən istifadə etməli olur. Yeni tiplər isə verilənlərin standart tiplərinə əsaslanaraq yaradılır. Yaradılan tiplərə verilənlərin sadalanan və interval tipləri deyilir.

Sadalanan tip verilənlər. Pascal dilində məhdud sayda qiymətləri ciddi nizamla təyin olunub saymaqla verilən sadalanan tipli verilənlərdən istifadə olunur. Sadalanan tipli verilənlər hər birinin adı olan konstantlar siyahısından ibarət nizamlı çoxluq olub elementləri dairəvi mötərizələr daxilində bir-birindən vergüllə ayrılmaqla verilir.Sadalanan tipli verilənlər proqramın type təsvirlər bölməsində Type və Var açar sözlərinin köməyi ilə aşağıdakı şəkildə təsvir olunur.

Type

Sad_tipin_adı= (konstantların siyahısı);

Var

Sad_tipli_dəyişən:Sad_tipin_adı;

Sadalanan tipli verilənlər dəyişənlərin təsviri bölməsində belə verilir.

Var

Sad_tipli_dəy : (konstantların siyahısı)

Burada Sad_tip_adı- sadalanan tipin qəbul olunmuş adını

Konstantların siyahısı- istifadəçi tərəfindən verilən ,bir-birindən “,” ayrılan və dairəvi mötərizə içərisində yazılan,maksimum sayı 256 olan konstantları

Sad_tip_dəy- proqramın icrası zamanı sabitlərin mənsub edildiyi dəyişəni bildirir.

Məsələn: Type qiymət=(1,2,3,4,5);

Var X: qiymət;

Burada qiymət sadalanan tipin adını 1,2,3,4,5 sadalanan konstantları, X isə bu konstantlardan istənilən birini ala bilən sadalanan dəyişəni bildirir.

Qeyd edək ki, type ilə təyin edilməmiş sadalanan tipləri də Var ilə daxil etmək olar.

Məsələn: Type qiymət-(1,2,3,4,5);

Var X:qiymət;

İnterval tipli verilənlər. Nizamlı çoxluğun məhdud alt çoxluğunu ədədi interval şəklində təyin etmək olar. Pascal da belə tipli verilənlər interval tipli verilənlər adlanır və proqramın tipləri bölməsində aşağıdakı şəkildə təsvir olunur.

Type

Interval_tipin_adı=const1..const2;

Var

Interval_tipli_dəyişənin_adı :interval_tipin_adı;

Bu tipli dəyişənlər Type açar sözündən istifadə edilmədən də, dəyişənlərin Var bölməsində aşağıdakı şəkildə təsvir olunur.

Var

Interval_tipli_dəyişən : (const1..const2);

Burada interval tipin adı-interval tipə verilmiş şərti ad

Interval tipli dəyişənin adı-proqram icra edilən zaman sabitlərin mənsub edildiyi dəyişəni Const1..const2 isə konstantların min və max qiymətləri olub, iki nöqtə ilə ayrılmaqla intervalı bildirir və const1 const2 olmaqla tam, məntiqi, simvol və ya sadalanan tipli konstantlar ola bilər.

Standart və sadalanan tiplər birlikdə *skalyar tip* adlanır.

real tipdən başqa ixtiyari skalyar tip üçün yeni – interval tip yaratmaq olar. Bu da skalyar tipə aiddir.

Tiplər bölməsində standart tiplərdən başqa, bütün tiplər e'lan olunmalıdır. Əgər proqramda yalnız standart tipli kəmiyyətlərdən istifadə olunarsa, e'lanlar bölməsinin tiplərə aid hissəsi olmur.

Konstantlar. Proqramda konstantlar açıq şəkildə qiyməti və işarə edildiyi adla (identifikatorla) verilir. Adı ilə verilən konstant verilənlərin təsviri bölməsində Const açar sözü ilə təsvir olunur

Const ad=qiymət ;

Burada const-konstant, sabit mənasını verir

Ad-sabitin işarə edildiyi identifikatoru

Qiymət-identifikatora aid edilən konstantın qiymətini bildirir.

Pascal dilində eyni və ya müxtəlif tipli bir neçə konstantı Const açar sözündən bir dəfə istifadə etməklə təsvir etmək olar. Bu zaman proqramın oxunuşunu asanlaşdırmaq üçün Const açar sözünü ayrıca sətirdə yazmaq məqsədə uyğun hesab edilir. Məsələn

Const

A=18; (*tam tipli konstant*)

Pi=3.1415; (*həqiqi t.*)

N= "X"; (*simvol t.*)

S=`true`; (*məntiq t.*)

Proqramda hər bir sabidə bir identifikator uyğun gəlir və identifikator təyin olunduqdan sonra ondan istifadə olunur

TP – də əməliyyatlar.

Əməliyyatlar yerinə yetirdikləri əməllər görə aşağıdakı qruplara bölünür:

1. Hesab əməliyyatları :

- a) unar : +, -
- b) binar : +, -, *, /, div, mod

2. Münasibət əməliyyatları :

=, <>, <, >, <=, >=

3. Məntiqi əməliyyatlar (Bul) :

not, and, or, xor

4. İnformasiya bitləri üzrə əməliyyatlar:

Pot, and, or, xor, shl, shr

5. Sətir əməliyyatı (konkatenasiya):

+

6. Çoxluqlar üzrə əməliyyatlar:

+, -, *, in, <=, >=

7. Ünvan əməliyyatı:

@

Binar əməliyyatlar aralarında əməliyyat simvolu olaraq, iki operatorlardan ibarət olur.

Məsələn, $(15 - 6) * 7$, $a \text{ and } b$, $x \text{ or } y$

Hesab əməliyyatları

Hesab əməliyyatları yalnız həqiqi və tam tipli kəmiyyətlərə tətbiq olunur. Bunlar binar və unar əməliyyatlara bölünür.

Unar «+» əməliyyatı tam yaxud həqiqi ədədin qarşısında qoyulur və qiymətə heç bir təsir etmir.

Unar «-» əməliyyatı tam və ya həqiqi ədəd qoyulur və qiymətin işarəsini dəyişir.

Binar hesab əməliyyatları üçün aşağıdakı cədvəli vermək olar:

Əməl	Əməliyyat	Operatorun tipi	Nəticənin tipi
+	toplama	tam, həqiqi	tam, həqiqi
-	çıkma	tam, həqiqi	tam, həqiqi
*	vurma	tam, həqiqi	tam, həqiqi
/	bölmə	tam, həqiqi	həqiqi
div	tam bölmə	tam	tam
mod	Bölmədən alınan qalıq	tam	tam

Məsələn: $6 \text{ div } 7 = 0$ $8 \text{ mod } 4 = 0$
 $15 \text{ div } 2 = 7$ $15 \text{ mod } 2 = 1$
 $(27 \text{ div } 4) * 3 + (13 \text{ mod } 5) = 6 * 3 + 3 = 21$

Münasibət əməliyyatları

Münasibət əməliyyatlarının nəticəsi Bull qiymətləridir (true, false). Sətir qiymətlərinin müqayisəsi ASCII –nin simvollar üzrə həyata keçirilir.

Göstərici tiplərin müqayisəsində yalnız = və < > əməliyyatlarından istifadə edilir.

Münasibət əməliyyatları:

=, <>, <, >, <=, >=.

Misal:

İfadə:	Nəticəsi:
$10 = 15$	false
$5 < > 5$	false
'a' = 'a'	true
'GDU' < 'ziba'	true
'beta' > 'b'	true

Sətir əməliyyatı (konkatenasiya).

TP-da iki sətir və simvolların konkatenasiya (birləşmə) əməliyyatında '+' simvollarından istifadə olunur. Bu əməliyyatın nəticəsində ikinci operand birinci operandın sonuncu simvolundan başlayaraq, onunla bitişdirilir. Alınan sətir 255 simvoldan çox olmalıdır, əks halda artıq simvollar atılır.

Misal:

'Turbo' + 'Pascal' → 'Turbo Pascal'

'G' + 'D' + 'U' → 'GDU'

'Riyaziyyat' + '-' + 'İnformatika' → 'Riyaziyyat - İnformatika'

Məntiqi (Bull) əməliyyatları

Məntiqi əməliyyatlar məntiqi tip kəmiyyətlərə tətbiq olunur və nəticə də məntiqi tipdə olur. Məntiqi əməliyyatlar Bull cəbrinə əsasən yerinə yetirilir:

Operatorlar		Əməliyyatlar				
x	y	not x	x and y	x or y	x xor y	
f	f	t	f	f	f	
f	t	t	f	t	t	
t	f	f	f	t	t	
t	t	f	t	t	f	

(f- false) (t- true)

TP-da məntiqi ifadələrin iki növ hesablanması mövcuddur: tam və qısaldılmış

Tam hesablamada bütün ifadənin qiyməti məlum olduqda belə hər bir operand hesablanır.

Qısaldılmış hesablamada isə bütün ifadənin qiyməti məlum olan kimi hesablama dayandırılır.

Məsələn, (a and b) ifadəsi üçün tam hesablamada həm a-nın, həm də b-nin qiyməti hesablandıqdan sonra bütün ifadənin qiyməti hesablanır. Qısaldılmış hesablamada isə a-nın qiyməti false olarsa, b-nin qiyməti hesablanmır. Belə ki, b-nin qiymətindən asılı olmayaraq ifadənin qiyməti false olacaq.

Coxluqlar 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 elementləri çoxluğudur. Məsələn,

$$[1, 3, 5, 7] + [5, 7, 9, 11] = [1, 3, 5, 7, 9, 11]$$

İki çoxluğun fərqi, yəni $A-B$ əməliyyatının nəticəsi, A çoxluğunun B çoxluğuna daxil olmayan elementlərindən təşkil olunur. Məsələn,

$$[1, 3, 5, 7] - [5, 7, 9, 11] = [1, 3]$$

İki çoxluğun kəsilməsi, yəni $A*B$ əməliyyatının nəticəsi A və B çoxluqlarının eyni elementlərindən təşkil olunur. Məsələn,

$$[1, 3, 5, 7] * [5, 7, 9, 11] = [5, 7]$$

A və B –nin elementləri eyni olduqda $A = B$ əməliyyatının nəticəsi True, $A < > B$ əməliyyatının nəticəsi isə false olur.

Əgər A çoxluğu B-nin altçoxluğudursa, $A \leq B$ əməliyyatının nəticəsi true olur.

Məsələn,

$$[3, 4, 5] \leq [1, 2, 3, 4, 5, 6] \rightarrow true$$

Əgər A çoxluğu B-nin bütün elementlərini saxlayırsa, $A \geq B$ əməliyyatının nəticəsi true olur.

Məsələn, $[3, 4, 5] \geq [3, 4] true$; $[3, 4, 5] \geq [3, 4, 5, 6] \rightarrow false$

Əgər hər hansı x kəmiyyəti, A-nin elementdirsə, onda $x \in A$ əməliyyatının nəticəsi true olur.

Məsələn,

$$3 \in [3, 4, 5] \rightarrow true$$

Ünvan əməliyyatı

@ əməliyyatı unar əməliyyat olmaqla yerinə yetirilmənin nəticəsi operandın göstəricisidir. @ əməliyyatında operand kimi dəyişən, prosedur, funksiya və s.-dən istifadə etmək olar.

İnformasiya bitləri üzərində əməliyyatlar

TP-da bitlər üzərində əməliyyatlarda yalnız tam tipli operandlar iştirak edirlər. Bu əməliyyatlar operandların ikilik təsvirində mərtəbələr üzrə yerinə yetirilir:

not – tam ədədin bütün bitlərinin unar inversiya əməliyyatı;

and – iki ədədin bitləri üzrə məntiqi \forall əməliyyatı;

or – iki tam ədədin bitləri üzrə məntiqi $\forall \exists$ əməliyyatı;

xor – iki tam ədədin bitləri üzrə istisnalı məntiqi;

$\forall \exists$ əməliyyatı;

shl – $A \text{ shl } B$ əməliyyatının nəticəsi, A operandının ikilik təsvirinin B bit qədər sola sürüşdürülməsindən alınan tam ədəddir.

shr – $A \text{ shr } B$ əməliyyatının nəticəsi, A operandının ikilik təsvirinin B bit qədər sağa sürüşdürülməsindən alınan tam ədəddir. Sürüşdürmə nəticəsində başalan mərtəbələr sıfırlarla doldurulur.

Misal, A və B Byte tiplidir.

$$A = 11_{(16)} = 0000 1011, \quad B = 2_{(16)} = 0000 0010;$$

$$\text{not } A = 1111 0100 = F4_{(16)} = 15 \cdot 16 + 4 = 244_{(10)}$$

A and B = 0000 0010 = 2
A or B = 0000 1011 = 11
A xor B = 0000 1001 = 9
A shl B = 0000 0010 = 2
A shr B = 0010 1100 = $2 \cdot 16 + 12 = 44$

Sərtsiz keçid operatoru.

Proqramlaşdırmada bəzən operatorların yerinə yetirilmə ardıcılığını dəyişmək lazım gəlir. Bunun üçün goto şərtsiz keçid operatorundan istifadə edilə bilər. Əmrin ümumi şəkli:

goto nişan;

TP-də istifadə olunan nişanın iki tipii var:

- 0-dan 999 – a kimi tam ədədlər;
- adi identifikatorlar.

İstifadə olunan bütün nişanlar label xidmət sözü ilə başlayan nişanın təsvir bölməsində göstərməlidir. Məsələn,

Label 0, 5, il_2;

Qeyd edək ki, goto operatoru struktur proqramlaşdırmanın əksinədir və ondan yalnız zəruri hallarda istifadə olunur.

Misal: iki ədədin bölünməsindən alınan qisməti tapmalı.

proqramm «bölmə»

label son;

var

x, y, nat: integer;

begin

write ('bölünəni daxil edin');

readln (x) ;

```

write ('bölünəni daxil edin');
readln (y) ;
if y = 0 then
begin
write ('Sıfır bölmə');
goto son;
end
nat: = x div y;
writeln ('qismət = ', nat);
son:
end

```

Boş operator. Boş operator heç bir əməliyyatı yerinə yetirmir və şərtsiz keçid operatorunda keçid üçün istifadə olunur. Yuxarıdakı misalda son: operatoru.

If şərt operatoru

İf operatoru bəzi şərtlərin doğru və ya yalan olmasından asılı olaraq operatorların yerinə yetirilmə ardıcılığını dəyişir. Bu operator vasitəsilə verilən əmrin ümumi şəkli:

```

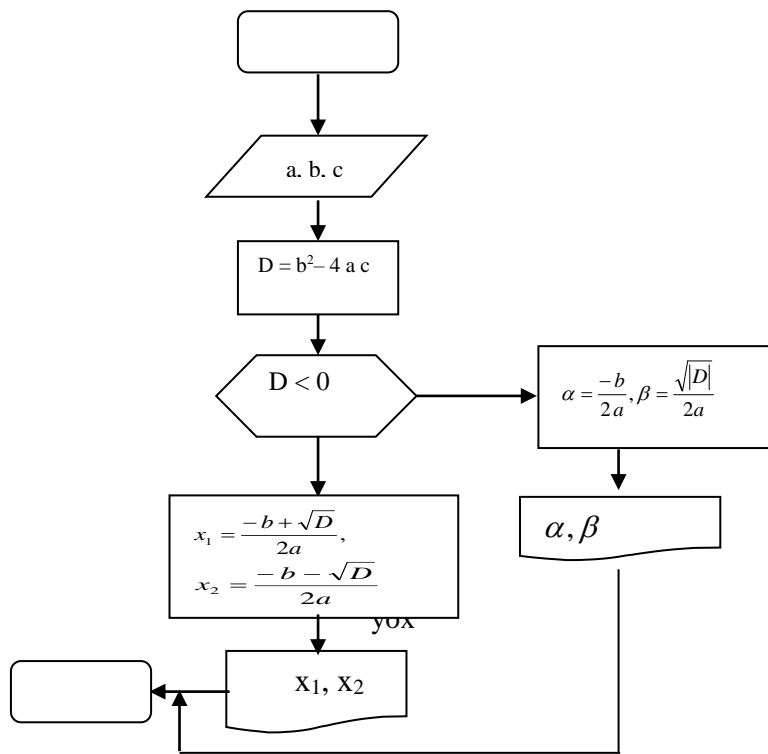
if məntiqi ifadə then operator;
və ya
if məntiqi ifadə then oper_1 else oper_2;

```

İf operatorunun I yazılış formasında məntiqi ifadənin qiyməti yalan olarsa, idarəetmə İF operatorundan sonrakı operatora verilir. II yazılış formasında isə məntiqi ifadənin qiyməti doğrudursa oper_1, yalandırsa oper_2 yerinə yetiriləcək. Bu operatorada aşağıdakı sintaksis xassələrinə əməl olunmalıdır:

- else xidmət sözündən əvvəl «;» işarəsi qoyulmur;
- then və else xidmət sözlərindən sonra yalnız bir operator olmalıdır, iki və ya daha çox operator yazılmalıdırsa, həmin operatorlar begin və end arasında yazılır.

Misal: $ax^2 + bx + c = 0$ kv. tənliyinin həlli: Əvvəlcə həllin blok-sxemini quraq



Onda həllin proqramı;

```

Proqram kv_tənlik;
var
  a, b, c, x1, x2, alfa, beta, k: real;
begin
  writeln ('a, b, c- ni daxil edin' );
  read (a, b, c );
  d = sqr (b) - 4 * a * c;
  if d < 0 then
    begin
      alfa = - b / (2 * a) ;
      beta = sqrt (abs (d) / (2 * a)) ;
      writeln ('alfa =', alfa, 'beta = ', beta)
    end
  else
    begin
      K = sqrt (d) ;
      x1 = (- b + K) / (2 * a) ;
      x2 = (- b - K) / (2 * a) ;
      writeln ('x1=', x1, 'x2=', x2)
    end
  end.

```

Case şart operatoru

operatorunu çox vaxt onu seçmə və ya variant operatoru da adlandırırlar. Bu operator if operatorunun ümumiləşməsidir. Operator selektorun qiymətindən asılı olaraq, bir neçə əməliyyatdan birini yerinə yetirməyə imkan verir.

Selektor kimi case və of xidmət sözləri arasında ifadədən istifadə olunur. İfadənin nəticəsi sıra tipli olmalı və 65535-i aşmamalıdır. Bu operator vasitəsilə verilən əmrin ümumi şəkli aşağıdakı kimidir.

Case ifadə of

sabit_1-in siyahısı: operator 1;

sabit_2-nin siyahısı: operator 2;

sabit_N₀-in siyahısı: operator n;

else

operator s

end

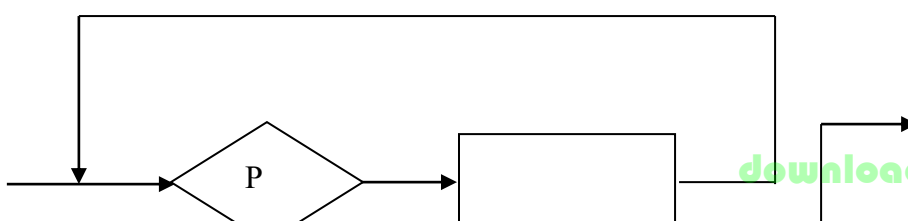
Operatoru niş prinsipini verək: ifadənin qiyməti hesablanır, əgər bu qiymət hər hansı sabit_i-nin ($i = 1, n$) qiymətlərindən biri ilə üst-üst düşərsə, onda operator_i ($i = 1, n$) yerinə yetirilir və digər operatorlar buraxılır.

Əgər ifadənin qiyməti sabit_ilər hər birinin qiymətləri ilə üst-üstə düşməzsə, operator_S yerinə yetirilir.

Case operatorunda else budağı olmaya da bilər. Bu halda ifadənin qiyməti sabit_i-lərdəki qiymətlərdən hər biri ilə üst-üstə düşməsə, idarəetmə end xidmət sözündən sonrakı operatora keçir.

Dövrü strukturlu hesablama proseslərinin proqramlaşdırılması

Ön şərtli dövr operatoru. Ön şərtli dövrlərin algoritminiaşağıdakı blok-sxemlə vermək olar:



Burada P məntiqi ifadə S-isə dövrün gövdəsidir. Dövrün gövdəsində Pşərtinə təsir edən hər hansı bir əmr olmalıdır.

Belə prosesləri proqramlaşdırmaq üçün while, do operatorlarından istifadə edilir. Bu operatorların adətən dövrlərin sayı əvvəlcədən məlum olmayan hallarda istifadə olunur. Operatorun ümumi şəkli aşağıdakı kimidir:

While şərt do operator;

şərt məntiqi ifadəsinin qiyməti yalnız true və false ola bilər. Burada dövrə daxil olmazdan əvvəl şərtə uyğun ifadənin qiyməti hesablanır. Əgər qiymət false olarsa dövrdən çıxış alınır və dövrün gövdəsi heç bir dəfə də yerinə yetirilmir. Bu zaman idarəetmə dövrün gövdəsindən sonrakı operatora ötürülür. Əgər qiymət true olarsa dövrə giriş baş verir və dövr gövdəsinin operatorları bir dəfə yerinə yetirilir. Bu zaman dövr gövdəsinin sonundan idarəetmə yenidən şərtin yoxlanılmasına qayıdır. Bu proses şərtin qiyməti false olana qədər davam etdirilir.

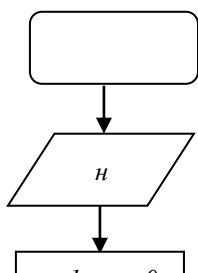
Əgər dövrün gövdəsi operatorlar qrupundan ibarət olarsa, onda həmin qrup

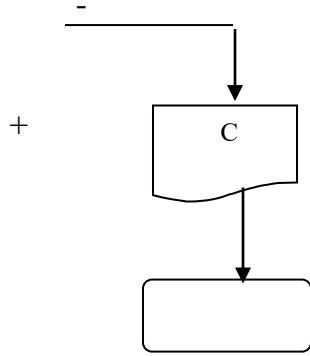
begin və end arasında verilir:

```
While şərt do
begin
    operator 1 ;
    operator 2 ;
    .....
    operator n ;
end;
```

Sadə bir misala baxaq. Tutaq ki, hər hansı n üçün $S = \sum_{i=1}^n \frac{1}{i^2}$ hesablanmalıdır.

Həllin blok sxemi:



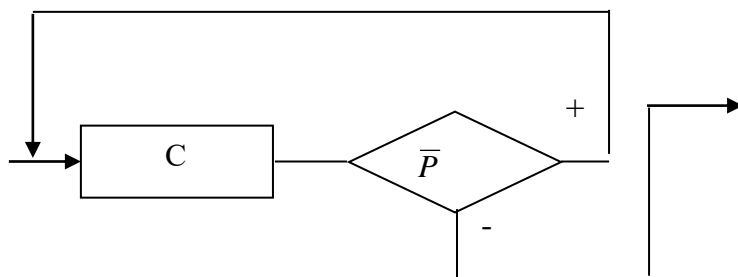


Həllin proqramını:

```

Proqram Ön_şərtli_dövr;
var
i, n : integer;
S : real;
begin
writeln ('n = '); readln (n);
i := 1 ; S := 0;
while i <= n do
begin
S := S + 1 / (i * i);
i := i + 1;
end;
write ('s = ', S);
end
  
```

Son şərtli dövr operatoru. Son şərtli dövrün blok - sxemini aşağıdakı kimi vermək olar:



Burada S - dövrün gövdəsi, P – şərti göstərən məntiqi ifadədir.

Belə proseslərin TP-də proqramlaşdırılması üçün repeat və until operatorlarından istifadə edilir. Bu operatorlar vasitəsilə dövrü proses aşağıdakı kimi proqramlaşdırılır:

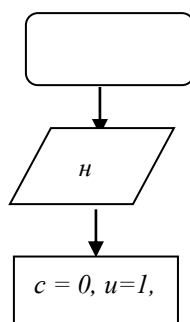
```
repeat
    operator 1 ;
    operator 2 ;
    .....
    operator n ;
until şərt;
```

repeat operatorunda da şərt məntiqi ifadədir. Operatoru niş prinsipi ön şərtli dövrə olduğu kimidir. Amma while – dən fərqli olaraq şərtin yoxlanılması dövrün gövdəsi yerinə yetirildikdən sonra baş verir. repeat dövr operatorunun idarə edilməsi while dövr operatorunun idarə edilməsi while dövr operatorunun idarə edilməsinin əksinədir. Yəni, burada while – dən fərqli olaraq, şərt false -yə bərabər olduqda dövr davam edir, əks halda isə sona yetir.

Qeyd etmək lazımdır ki, while – dən fərqli olaraq, repeat operatorundan istifadə etdikdə operatorlar qrupu dövrün gövdəsini təşkil edərsə begin – end operatorları tələb olunmur. Lakin repeat operatoru while operatoru kimi universal deyil. Belə ki, burada şərt dövrün gövdəsindən sonra yerləşdiyindən , lazım olmasa belə dövrün gövdəsi heç olmazsa bir dəfə yerinə yetirilir. Bu isə o deməkdir ki, bəzi dövrləri təşkilində son şərtli dövr operatorundan istifadə zamanı ehtiyatlı olmaq lazımdır.

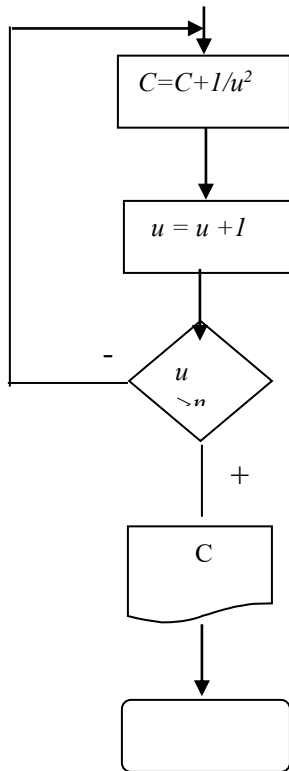
Yenə $S = \sum_{i=1}^n i^2$ cəminin hesablanması misalına baxaq:

Həllin blok sxemi:



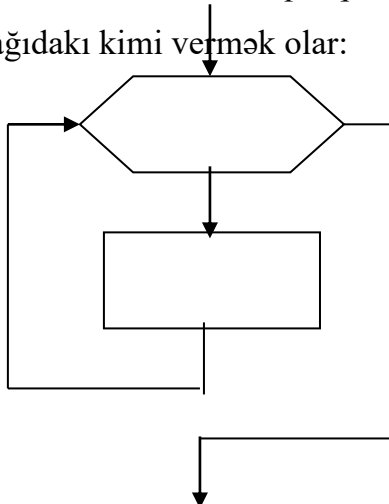
Həllin proqramı:

```
Proqram Ön_şərtli_dövr;
var
S : real;
i, n : integer ;
begin
```



```
writeln ('n = '); readln (n);
i: =1 ; S : =0;
repeat
S := S+1/(i * i)
i: = i +1;
until i > n
write ('s = ', S);
end
```

Parametrlı dövlərin proqramlaşdırılması. Parametrlı dövrün təşkli üçün blok sxemi aşağıdakı kimi vermək olar:



Parametrlı dövr blokunda parametrlin ilk qiyməti (i, q) və son qiyməti (s.q) verilir. Parametrlin dəyişmə addımı 1-ə bərabərdir

Belə dövllərin TP-də proqramlaşdırılması üçün for dövr operatorundan istifadə edilir. Bu operatorndan dövrün parametrlinin ilk və son qiymətləri məlum olduqda istifadə edilir. Bu isə onun while və repeat universal dövr operatorlarına nisbətən daha geniş sahələrə tətbiqinə imkan verir.

for dövr operatoruna sayğaclı dövr operatoru da deyirlər. Onun aşağıdakı iki variantı var:

1. Dövrün yerinə yetirilmə prosesində sayğacın qiyməti artır:

```
for parametr := ilk qiymət to son qiymət do operator;
```

2. Dövrün yerinə yetirilmə prosesində sayğacın qiyməti azalır:

for parametr := ilk qiymət downto son qiymət do operator;

Qeyd edək ki, while və repeat operatorlarından fərqli olaraq, for operatorunda sayğacın ilk qiyməti dövrün başlanğıcından əvvəl deyil, dövrün elə başlanğıcında verilir və sayğacın qiymətinin dəyişməsi üçün xüsusi operator tələb olunmur.

for dövr operatorunda yerinə yetirilmə zamanı sayğacın ilk və son qiymətləri yadda saxlanır və parametrə ilk qiymət mənimsədilir. Sonra dəyişənin qiyməti son qiymətlə müqayisə edilir. Dövrün parametri son qiymətdən \leq isə (I variant) isə dövrün növbəti iterasiyası yerinə yetirilir. Əks halda dövrədən çıxış alınır. Dövrün gövdəsi yəinə yetirildikdən sonrasayğacın qiymətinin artması və ya azalması ilə növbəti iterasiya başlayır. Bu artma (azalma) avtomatik yerinə yetirilir. TP-də for dövr operatorunun iki əsas məhdudiyyəti var:

1. Dövr sayğacının dəyişmə addımı I variantda +1, II variantda – 1 ola bilər.
2. Dövrün parametri yalnız sıra tipindən olmalıdır və for operatorunun yerləşdiyi blok üçün lokaldır.

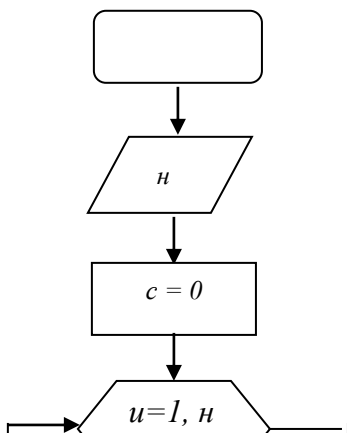
Misal: $S = \sum_{i=1}^n i^2$ - n-i hesablayaq:

II variantda : dövrün blokunda $i = n, 1$ yazılır, TP proqramda isə dövr operatoru:

for $i = n$ downto 1 do kimi verilir.

Break və continue standart prosedurlarından repeat, while və for dövr operatorlarında istifadəsi, repeat, while və for dövrlərində standart break və continue prosedurlarından istifadə etmək olar.

Həllin alqoritmi:



Həllin TP-proqramı:

Proqram parametrli _dövr;

var

i, n : integer ;

```

S : real;

begin

writeln ('n = '); readln (n);

S := 0 ;

for i := 1 to n do

S:= S +1 / (i * 1);

write ('s = ', S )

end

```

Break proseduru çıxış şərtinin yerinə yetirilməsini gözləmədən dövrədən çıxmağa imkan verir. Continue proseduru isə dövrün əvvəlki iterasiyası sonna çatmadan yeni iterasiyanın başlanmasına imkan verir.

İterasiyalı dövrlər.

Təkrarlanmaların sayı məlum olmayan dövrlər içərisində əsas yeri, təkrarlanma prosesində dövrün gövdəsinin $y_1, y_2, \dots, y_n, \dots$ qiymətlər ardıcılığının müəyyən bir limitə yığılması (yəni $\lim_{n \rightarrow \infty} y_n = a$) dövrləri tutur.

Burada ardıcılığın hər bir $y_n = a$ qiyməti, əvvəli y_{n-1} qiyməti vasitəsilə təyin olunur və əvvəlki qiymətə nisbətən axtarılan nəticəyə (a) daha dəqiq yaxınlaşma olur. Belə ardıcıl yaxınlaşmaları təmin edən dövrlər iterasiyalı dövrlər, hər bir yaxınlaşma isə iterasiya adlanır. İterasiyalı dövrlərdə dövrün davamı şərti, n artdıqca y_n qiymətlərinin a limitinə yaxınlaşmasına əsaslanır. İterasiyalı dövrün qurtarması üçün n-in müəyyən qiymətlərində

$$|y_n - y_{n-1}| < \varepsilon$$

şərti ödənməlidir, burada ε - nəticənin hesablanmasında müəyyən yol verilən xətdir.

İterasiyalı dövrü prosesə tipik misal kimi sonsuz sıranın cəminin hesablanmasını göstərmək olar. Cəm anlayışı sonsuz sıranın yığılması anlayışı sonsuz sıranın yığılması anlayışı ilə əlaqəlidir. Əgər,

$$S_n = y_0 + y_1 + \dots + y_n \text{ olarsa, onda } \lim_{n \rightarrow \infty} S_n = S \text{ yaxud } \lim_{n \rightarrow \infty} \sum_{i=0}^n y_i = S \text{ olmalıdır.}$$

Sıranın ümumi həddi isə $\lim_{n \rightarrow \infty} y_n = 0$ yaxud $\lim_{n \rightarrow \infty} (S_n - S_{n-1}) = 0$ olmalıdır. Buradan dövrün qurtarması şərti

$$|S_n - S_{n-1}| \leq \varepsilon \quad \text{yaxud} \quad |y_n| \leq \varepsilon \quad \text{kimi təyin olunur.}$$

Misal . Kosinusun sıraya ayrılışından istifadə edərək $s = \cos x$ funksiyasının qiymətini $\varepsilon = 10^{-4}$ dəqiqliyi ilə hesablayın:

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots = \sum_{i=0}^{\infty} y_i(x)$$

burada : $y_i(x) = (-1)^i \frac{x^{2i}}{(2i)!}$

Rekurent münasibəti tapmaq üçün :

$\frac{y_i(x)}{y_{i-1}(x)}$ hesablayaq.

$$\frac{y_i(x)}{y_{i-1}(x)} = \frac{(-1)^i \frac{x^{2i}}{(2i)!}}{(-1)^{i-1} \frac{x^{2(i-1)}}{(2i-2)!}} = -\frac{x^2}{(2i-1)(2i)}$$

buradan:

$$y_i(x) = -\frac{x^2}{(2i-1) \cdot 2i} \cdot y_{i-1}(x) \quad y_0(x) = 1$$

Onda hesablama proqramı aşağıdakı kimi olar:

Proqram kosinus;

var

i: integer ;

eps, S, x, y : real;

```

begin
  read (x, eps);
  S := 0 ;
  Y := 1
  i := 1
while abs (y) > eps do
begin
  S:= S +Y;
  u:= -u * x * x/(2*i * (2*i -1));
  i:= i +1;
end;
writeln ('cos x = ' S);
end

```

Verilənlərin strukturları

Proqramlaşdırmada verilənlər iki əsas qrupa bölünürlər:

- a) statik strukturlu verilənlər;
- b) dinamik strukturlu verilənlər.

Elementlərinin sayı, onların qarşılıqlı yerləşməsi və qarşılıqlı əlaqəsi proqramın realizasiyası zamanı müəyyən qayda üzrə dinamik dəyişən verilənlər dinamik strukturlu verilənlər adlanır.

Statik strukturlu verilənlər. Statik strukturlu verilənlər hər hansı qayda üzrə sadə strukturlardan təşkil olunmaqla sadə (skalyar) və mürəkkəb (aqrəqativ) ola bilər.

Proqramlaşdırma dillərində sadə verilənlərə verilənlərin standart (əvvəlcədən təyin olunmuş) tipləri uyğundur. Bura hesabi (natural, tam, həqiqi, kompleks), simvol, məntiqi və göstərici tipləri uyğundur. TP-da Byte, Word natural tiplər, İnteger, Shortint, Longint tam tiplər, Real, Single, Double, Extended, Comp həqiqi tiplər, Boolean, Byte Bool,

Word Bool, Long Bool məntiqi tiplər, Char simvol tipi və Pointer göstərici tipləri vardır.

Bunlardan əlavə TP-da istifadəçi tərəfindən təyin olunan sadalanan və interval tiplərindən də istifadə olunur.

Mürəkkəb strukturlu verilənlərə bircins, yəni bütün elementləri eyni tiptən olanlar və qeyri-bircins (kombinə edilmiş), yəni müxtəlif tip elementlərdən təşkil olunmuş verilənlər aiddir.

Bircins strukturlu verilənlərə massivlər, sətirlər və çoxluqlar, qeyri-bircins strukturlu verilənlərə isə sadə yazı, variantlı yazı, birləşmə və obyekt tipləri aiddir.

Dinamik strukturlu verilənlər. Dinamik strukturlu verilənlərə fayllar, əlaqəsiz və əlaqəli dinamik verilənlər aiddir.

Fayllar, mətn, tipləşdirilmiş və tipləşdirilməmiş olurlar.

Əlaqəsiz dinamik verilənlər statik strukturlu verilənlərə analogi olaraq təsnif olunur. Əlaqəli dinamik verilənlər isə xətti, dairəvi və budaqlanan struktura malikdir.

Statik strukturlu verilənlərlə iş.

TP-da standart daxiletmə Read, Readln prosedurları, standart xaricetmə isə Write, Writeln prosedurları vasitəsilə həyata keçirilir.

Bu prosedurların yazılış forması aşağıdakı kimidir:

Read (fayl dəyişəninin adı, dəyişənlərin siyahısı);

Readln (fayl dəyişəninin adı, dəyişənlərin siyahısı);

Write (fayl dəyişəninini adı, xaricedilən elementlərin siyahısı);

Writeln (fayl dəyişəninini adı, xaricedilən elementlərin siyahısı)

Standart daxiletmə əvvəldən təyin olunmuş, klaviatura ilə əlaqəli Input adlı mətn faylından yerinə yetirilir. Standart xaricetmə isə əvvəlcədən təyin olunmuş, monitor ilə əlaqəli Output adlı mətn faylında yerinə yetirilir. Qeyd edək ki, aşağıdakı proqramm fraqmentləri ekvivalentdir:

Readln (Input, a, b);

Writeln (Output, 'a = ', a, ' b =', b)

və

Readln (a, b);

Writeln ('a = ', a , ' b =', b) ;

Standart daxiletmə və xaricetmə prosedurlarından istifadə edərkən, aşağıdakı nəzərdə tutulur:

a) Read və Readln prosedurları ilə yalnız tam, həqiqi, simvol və sətir tipli verilənlər oxunur

b) Write və Writeln prosedurları ilə yalnız tam, həqiqi, simvol, sətir və bull tipli verilənlər xaric edilir.

Ədədlər formatsız və formatlı şəkildə çap oluna bilər.

Formatsız çapda verilənlər xaricetmə əmrində bir-birindən vergüllə ayrılmış şəkildə verilir.

Formatlı çapda isə dəyişənlərin adı ilə yanaşı, çap formatı da verilir:

< dəyişənin adı > : 1-ci tam ədəd : 2-ci tam ədəd

1-ci tam ədəd çap üçün sahənin uzunluğunu;

2-ci tam ədəd isə çap olunan ədəd həqiqi olduqda yazılır və onluq kəsrdəki rəqəmlərin sayını göstərir.

İstifadəçi tipləri

Pascal proqramlaşdırma dilində baza tiplərdən əlavə verilənlərin digər tiplərini də yaratmaq mümkündür. Proqramlaşdırıcılar tərəfindən yaradılan bu tiplər istifadəçi tipləri yaxud düzəltmə tiplər adlanırlar.

Sadalanan tip.

Sadalanan tip, alacağı qiymətlər proqramçı tərəfindən müəyyənləşdirilən yeni tip verilənlər tərtib etməyə imkan verir.

Sadalanan tipin elementləri mötərizə içərisində aralarında vergül işarəsi olmaqla yazılır.

Məsələn,

```
type
  həftə = (b_e, ç_a, Ç, c_a, c, s, b);
var`
    1-ci, 2-ci: həftə;
```

Həftə tipində həftənin günlərinin adı göstərilir, sonra isə 1-ci və 2-ci-nin həftənin günləri olduğu verilir.

Tipin hər bir elementi bir identifikatordur. Eyni identifikator müxtəlif tiplərin elementi ola bilməz.

Sadalanan tip sıra tiplidir. Onun elementlərinə nizamlanmış sabitlər kimi baxmaq olar, və onları müqayisə etmək olar. Hər bir elementin sıra nömrəsi Ord funksiyası ilə təyin edilə bilər. Bu tipin elementlərinə Pred və Succ funksiyaları tətbiq oluna bilər.

Məsələn:

$\text{Ord}(c_a) = 2, \text{Pred}(s) = c, \text{Succ}(s) = b$

Sadalanan tipin elementlərinə hesab əməlləri tətbiq oluna bilmir, həmçinin onları daxil və çıxarmaq olmaz. PASCAL-da yalnız tam, həqiqi, simvol və sətir tipli verilənləri daxil etmək olar.

Sıra tipli (tam, məntiqi, simvol, sadalanan tiplər) hər hansı dəyişənin dəyişmə intervalını məhdudlaşdırmaq istədikdə interval tipindən istifadə olunur. Buna bəzən məhdud tip və ya diapozon tipi də deyilir. İnterval tipinin təsvirində bərabərliyin solunda tipin adı, sağında isə bir-birindən iki yanaşı nöqtə ilə ayrılan interval sərhədləri göstərilir. Sərhəd elementləri hər hansı sıra tipii verilən olmalıdır. Məsələn:

```
type
  t = 0.....60;
  Gün = b_e .....s;
  Rəqəm = 0.....9;
  Hərflər = 'a' ..... 'z' ;
var
  x : t; y: Gün; t: Rəqəm;

yaxud
var
  Y1: b_e .....s;
  m: -5.....5;
```

Sıra tipləri

Tam, məntiqi, simvol və sadalanan tiplər sıra tipləri adlanırlar. Bu tipləri nizamlı tiplər də adlandırılırlar. Sıra tipləri aşağıdakı xassələrə malikdirlər:

1. Sıra tipin elementləri 0-dan başlayaraq nömrələnirlər. tiplərində elementlərin sıra nömrələri öz qiymətləridir.

2. Sıra tipli istənilən elementin sıra nömrəsi Ord funksiyası vasitəsilə təyin oluna bilər.

3. Pred funksiyası, funksiyasının argumentindən əvvəlki elementi göstərir. Bu element birinci elementdirsə, onda sonuncu elementi göstərir.

4. Succ funksiyası, funksiyasının argumentindən sonrakı elementin göstərir. Əgər, bu element sonuncu elementdirsə, onda, birinci elementi göstərəcəkdir.

Bircins strukturlu mürəkkəb verilənlər. Qeyri-bircins strukturlu mürəkkəb verilənlər

Plan:

1. Çoxluq tipi
2. Çoxluqlar üzərində əməliyyatlar
3. Massiv tipi
4. Yazı tipi
5. Variantlı yazılar
6. Yazı sahələrinə With birləşdirici operatoru vasitəsilə müraciət

Çoxluq tipi

Proqramlaşdırmada çoxluq anlayışı ilə eyni mənada işlədilir. Lakin bəzi fərqləri də vardır. Belə ki, Pascal – proqramlarda ancaq sıra (tam, məntiqi, simvol, sadalanan) tipli elementləri olan çoxluqlardır istifadə olunur. Çoxluğun bütün elementləri eyni tipli olmalıdır və həmin tip baza tipi adlanır. Çoxluğun tərtibi [0.....255] intervalından kənara çıxma bilməz. Ona görə də tam tiplər qrupuna aid olan Shortint, Integer, Longint və Word tipli verilənlərdən baza tipi kimi istifadə etmək olmaz.

Çoxluq tipinin təsviri ümumi şəkildə aşağıdakı kimi verilir:

```
type  
  < identifikator > = set of < baza tip > ;  
Məsələn:
```

```
simvollar = set of char;  
rəqəm = set of 0.....9;  
hərflər = 'a' ..... 'z' ;
```

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 elementləri çoxluğudur. Məsələn,

$$[1, 3, 5, 7] + [5, 7, 9, 11] = [1, 3, 5, 7, 9, 11]$$

İki çoxluğun fərqi, yəni A-B əməliyyatının nəticəsi, A çoxluğunun B çoxluğuna daxil olmayan elementlərindən təşkil olunur. Məsələn,

$$[1, 3, 5, 7] - [5, 7, 9, 11] = [1, 3]$$

İki çoxluğun kəsilməsi, yəni $A * B$ əməliyyatının nəticəsi A və B çoxluqlarının eyni elementlərindən təşkil olunur. Məsələn,

$$[1, 3, 5, 7] * [5, 7, 9, 11] = [5, 7]$$

A və B –nin elementləri eyni olduqda $A = B$ əməliyyatının nəticəsi *True*, $A < > B$ əməliyyatının nəticəsi isə *false* olur.

Əgər A çoxluğu B -nin altçoxluğu varsa, $A < = B$ əməliyyatının nəticəsi *true* olur.

Məsələn,

$$[3, 4, 5] < = [1, 2, 3, 4, 5, 6] \rightarrow \text{true}$$

Əgər A çoxluğu B -nin bütün elementlərini saxlayırsa, $A > = B$ əməliyyatının nəticəsi *true* olur.

Məsələn, $[3, 4, 5] > = [3, 4] \rightarrow \text{true}$; $[3, 4, 5] > = [3, 4, 5, 6] \rightarrow \text{false}$

Əgər hər hansı x kəmiyyəti, A -nin elementdirsə, onda $x \in A$ əməliyyatının nəticəsi *true* olur.

Məsələn,

$$3 \in [3, 4, 5] \rightarrow \text{true}$$

Massiv tipi

Massiv verilənlərin elə strukturudur ki, bu strukturun elementləri bircins olub, sayı və konfigurasiyası proqramın icrası zamanı dəyişmir, elementlər sistem tərəfindən nömrələnərək, nömrəyə görə nizamlanır.

Massiv, massivin adı, koordinatların dəyişmə diapazonu və elementlərin tipi ilə təsvir olunur. Koordinatların sayı – massiv ölçüsü, koordinatların özü isə elementlərin indeksi adlanır. massivin təsviri üçün array xidmət sözündən istifadə olunur və massiv tipi ümumi şəkildə aşağıdakı kimi təsvir olunur:

$\langle \text{identifikator} \rangle = \text{array} [\text{indekslər}] \text{ of } \langle \text{tip} \rangle ;$

Burada indekslər hər hansı tipə aid sabit və ya dəyişən, $\langle \text{tip} \rangle$ - isə Pascalda hər hansı tipdir.

Məsələn

type

sətir = array [1.....5] of real ;

matris = array [1.....6] of sətir ;

Burada sətir massivi 5 həqiqi ədəddən ibarət xətti cədvəli, matris massivi isə 5 sütun və 6 sətiri olan həqiqi ədədlərdən ibarət düzbucaqlı cədvəli göstərir. Bu təsviri TP-da aşağıdakı kimi də vermək olar:

type

matris = array [1...5, 1...6] of real

Massivin hər hansı elementinə müraciət, massivin adı və düzbucaqlı mötərizə içərisində bir-birindən vergüllü ayrılan indekslər vasitəsilə aparılır:

Məsələn :

var

A: matris;

begin

Y= A[5, 2] + A[3, 4];

Misal 1. 10 elementdən ibarət massivin ən böyük elementini və həmin elementin massivdəki yerini tapmalı

Proqram M1;

const n =10;

type

vektor = array [1...n] of real;

var

A : vector;

max : real

k , i : integer;

begin

writeln ('Massivin elementlərini daxil edin');

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

max := A [i]; k:= 1;

for i :=1 to n do

if max < A [i] then begin

max: A [i]; k:= i

end ;

writeln ('max =', max , 'k = ', k)

end.

Misal 2. A(n,m) massivin sətir elementlərinin hasiləri cəminin hesablanması

Proqram M2;

const n =3; m =2;

type matris = array [1...n, 1...m] of real;

var

A : matris;

i, j : integer;


```

z, s : real ;
begin
  for i:=1 to n do
    begin
      write (i, '- ci sətir elementləri: ');
      for j :=1 to m do read (A [i,j]);
        end ;
      S := 0;
      for i :=1 to n do
        begin
          z := 1;
          for j := 1 to n do
            z := z * A [i,j] ;
          s := s + z ;
          end;
          writeln ('cəm =', max, s) ;
        readln
      end.

```

Yazı tipi

Massivlərdən, çoxluqlardan, fayllardan fərqli olaraq, yazı tipii qeyri-bircins tipdir. Massiv, çoxluq və fayl öz tərkibinə eyni tip elementləri daxil etdiyi halda, yazı tipinin tərkibinə ixtiyari sayda müxtəlif tipli verilənləri (sadə dəyişənlər, massivlər, çoxluqlar, yazılar və fayllar) daxil edir. Yazıya daxil olan bu verilənlər yazının sahələri adlanır. Yazı tipinin təsvirində record və end xidmət sözlərindən istifadə olunur. Yazı tipinin ümumi yazılış forması aşağıdakı kimidir:

```

type
  < identifikator > = record S1 : T1; S2 :T2; ....., Sn: Tn ; end;

```

Burada S_i - yazı sahələrinin adları, T_i isə sahələrin tipləridir. Məsələn:

```

type
  Complex = record      Re : real ; İm: real; end

  Tarix = record        İl : integer ; Ay : 1.....12; gun : 1.....31; end;

```

Yazı tipi təsvir edildikdən sonra bu tipin dəyişənləri və tipləşdirilmiş sabitləri verilə bilər. Yazı tipli sabitlərin təsvirində yazının bütün sahələrinin qiymətləri tlə bərabər, onların identifikatorları da göstərilir: Məsələn:

```

var
  x, y, z : Complex;
  Tar : Tarix;
  const

  Respub: Tarix (İl : 1918; Ay :5; Gun : 28);

```

Dəyişənləri birbaşa yazı tipi vasitəsilə təyin etmək olar:

```
var
  x, y, z : record Re, İm: real; end;
```

Yazı sahələrinə müraciət bir-biri ilə nöqtə ilə ayrılan dəyişən və sahənin adı vasitəsilə verilir. Məsələn:

```
x. Re := 3,8; Y. Re := - x . Re
```

```
x. İm := 4,5; Y. İm := x .Re+3;
```

Variantlı yazılar

Bəzi proqramlarda yalnız bəzi sahələrinə görə bir-birindən fərqlənən bir neçə «yazı» dan istifadə etmək lazım gəlir. Belə hallarda proqramı təşkil edən operatorların sayını azaltmaq, yaddaşa qənaət etmək, proqramın tərtibini sadələşdirmək və proqramı daha kompakt etmək məqsədilə bir neçə adi «yazı» əvəzinə bir variantlı yazıdan istifadə məqsədəuyğundur.

Variantlı yazı iki hissədən ibarət olur:

1. Birinci hissə adi yazıdan;
2. İkinci hissə seçmə əlamətindən asılı olaraq, variant (Case) operatoru vasitəsilə seçilən variantlar siyahısından.

Tutaq ki, iki semestrin imtahan qiymətləri əsasında tələbələrin şəxsi kartoçkasını tərtib etmək lazımdır. Bu iki kartoçka bir-birindən imtahanların adları və alınan qiymətlərə görə fərqlənəcəklər. Tələbələrin ad, soyad və qrup nömrələri isə dəyişməz qalacaq. Hər iki semestrin nəticələrini variantlı yazıdan istifadə etməklə bir yazı şəklində vermək olar:

```
type
```

```
String 7 = string [7]; string 20 = string [20];
```

```
Qsem 1 = record Riyanal_1 : Byte; İnformatika: Byte;
```

```
Fiz : Byte; Tarix : Byte; end;
```

```
Qsem 2 = record Riyanal_2: Byte; Xar-dil: Byte; Alq- dil: Byte;
```

```
Riy-ment: Byte; end;
```

```
Tcard = record Ad: string 20; soyad: string 20;
```

```
Ata – adı: string 20; Qrup- nom: string 7;
```

```
Case semestr : Byte of
```

```
1: (Qiymət 1 : Qsem 1);
```

```
2: (Qiymət 2 : Qsem 2);
```

```
end;
```

```
end;
```

Qeyd edək ki, variantlı yazıya daxil olan Case operatorundan sonra heç bir yazı sahəsi vermək olmaz.

Yazı sahələrinə With birləşdirici operatoru vasitəsilə müraciət

Yazılar ilə işi sadələşdirmək və proqramma əyanilik vermək məqsədilə Pascalda xüsusi With birləşdirici operatorundan istifadə olunur. Bu operator_qoşma operator adlanır. Operatorun təsvirini ümumi şəkildə aşağıdakı kimi vermək olar:

With < Yazı dəyişəninin adı > do < operator > ;

Əgər, With operatorundan istifadə olunursa, onda yazı sahələrinə müraciətdə bir-birindən nöqtələrlə ayrılmış identifikatorlar zəncirindən təşkil olunmuş ad bütöv göstərməlidir.

Məsələn:

tupe

```
t_rec = record      A: record
                    B: record
                    x : char
                    y: byte;
                    end;
                    C: real ;
                    end;
                    D : string ;
                    end;
                    var  rec: t_rec;
```

Burada y sahəsinə 2, C sahəsinə 2,71828 qiymətlərinin mənimsədilməsi tələb olunursa, onu aşağıdakı kimi vermək olar:

```
rec. A. B.Y: = 2;
rec. A. C := 2.71828 ;
```

Sadə halda With birləşdirmə operatoru yazı sahələrinin adlarını aşağıdakı kimi ixtisar etməyə imkan verir:

```
With rec do
begin
```

Burada 2 With operatorundan istifadə etmək olar:

```

With rec do
With A do
begin
    B.Y: = 2;
        C: = 2.71828;
end;

```

Burada `rec` və `A`-nın adını bir siyahıda göstərməklə `With` operatorunun köməyi ilə yuxarıdakı proqram fragmentini daha yığcam yazmaq olar:

```

With rec, A do
begin
    B.Y: = 2;
        C: = 2.71828;
end;

```

Yalnız `X` və `Y` sahələrinə müraciət tələb olunursa, onda bunu aşağıdakı kimi yazmaq olar:

```

With rec, A, B do
begin
    x: = '5';
        y: = 2;
end.

```

Fayllar.

Fayl, xarici yaddaşda adlandırılmış hər hansı bir sahədir. Bu faylın fiziki mövcudluğudur ki, ona görə də ona çox vaxt fiziki fayl deyirlər.

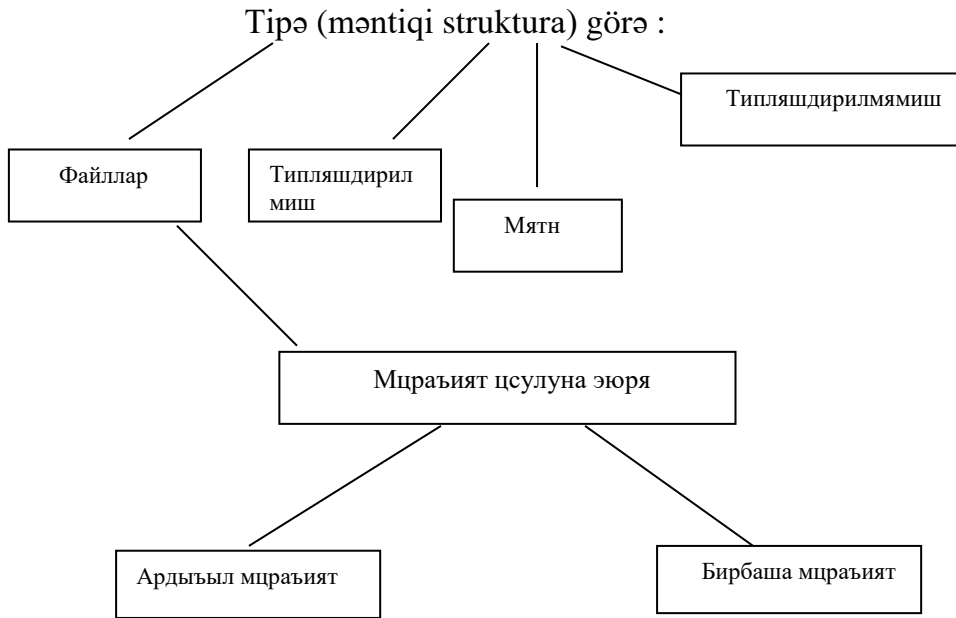
Digər tərəfdən fayl proqramlaşdırmada istifadə olunan çoxlu sayda verilənlər strukturundan biridir. Bu halda fayl-məntiqi fayl adlanır. Proqramlarda məntiqi fayllar fayl dəyişəni ilə təsvir olunur.

Fiziki faylın strukturu informasiya daşıyıcılarında ardıcıl baytlardan ibarətdir.

Faylın məntiqi strukturu prinsipə massivə oxşayır. Amma fayl və massivin bəzi fərqləri vardır.

Massiv əməli yaddaşda təşkil olunur və onun elementlərinin sayı məlumdur. Faylda isə proqramın işi prosesində elementlərin sayı dəyişir və o, xarici yaddaşda yerləşir. Faylın sonuna ASCII kodu 26 (Ctrl+Z) olan Eof simvolu qoyulur.

Faylların Turbo Pascal-da təsnifatını aşağıdakı kimi vermək olar:



Qeyd edək ki, tipləşdirilmiş və tipləşdirilməmiş fayllara hər iki müraciət üsulu mümkündür, mətn fayllarına isə yalnız ardıcıl müraciət etmək olar.

Faylların təyini, açılması, bağlanması

Fayl tipinin təyini üçün file və of xidmət sözlərindən istifadə olunur. Bu xidmət sözlərindən istifadə olunur. Bu xidmət sözlərindən sonra fayl komponentlərinin tipii göstərilir. Məsələn,

type

N = file of integer;

Simvol = file of "A" "..." "Z" ;

Text standart tipii simvoldan təşkil olunmuş sətirlərdən ibarət fayl tipini təyin edir. Lakin Text və Char tipləri ekvivalent deyil.

Məntiqi və fiziki faylların əlaqələndirilməsini Assign proseduru yerinə yetirir. Bu prosedurdan yalnız bağlı fayllar üçün istifadə etmək olar. Prosedurun I parametri fayl

dəyişəni, II parametri isə qiyməti fiziki faylın adı olan sətir sabiti və ya sətir dəyişəninin identifikatorudur. Məsələn,

Assign (f, 'Kafedra. doc') ;

Faylın oxunub, yazılması əməliyyatını yerinə yetirməzdən əvvəl bu fayl açılmalıdır.

Faylların açılması Reset və Rewrite prosedurları, bağlanması isə Close proseduru ilə həyata keçirilir:

Reset (f); Rewrite (f) ; Close (f) ;

Reset proseduru f dəyişəni ilə əlaqədə olan mövcud fiziki faylı açır.

Rewrite proseduru f faylı dəyişəni ilə əlaqədə olan yeni fiziki fayl yaradır. Əgər, bu adda fiziki fayl mövcudsa, o silinir və onun yerində yeni boş fayl yaradılır.

Fayllarla iş zamanı Eof (f) funksiyasından da istifadə olunur. Əgər, göstərici faylın sonuncu mövqeyində olarsa, yaxud fayl boş olarsa bu funksiyanın qiyməti True, əks halda False olur.

Tipləşdirilmiş fayllar

Tipləşdirilmiş faylların bütün elementləri bir tiptən olmalıdır. Tipləşdirilmiş faylın elementləri fayl tipindən başqa istifadəçi tiptən ola bilər. Tipləşdirilmiş fayllara ardıcıl və birbaşa müraciət etmək olar. Tipləşdirilmiş faylların elementləri sıfırdan başlayaraq nömrələnir.

Tipləşdirilmiş fayllardan oxumaq üçün yalnız Read proseduru, yazmaq üçün isə Write prosedurundan istifadə edilir:

Read (fayl dəyişəninin adı, dəyişənlərin siyahısı);

Write (fayl dəyişəninin adı, dəyişənlərin siyahısı);

Tipləşdirilmiş fayllarla birbaşa əməliyyatlar üçün aşağıdakı prosedur və funksiyalardan istifadə edilir:

FilePos – göstəricinin fayldakı cari mövqeyinin nömrəsini verir;

FileSize – faylın cari ölçüsünü (elemen.sayı) verir;

Seek – göstəricinin fayldakı cari mövqeyini verilmiş nömrələri elementə dəyişir;

Truncate – faylın ölçüsünü göstəricinin cari mövqeyinə qədər qısaldır. Qalan elementlər silinir.

Tipləşdirilməmiş fayllar

Tipləşdirilməmiş faylları təsvir edərkən yalnız file xidmət sözündən istifadə edioir. Məsələn,

var F: file ;

Tipləşdirilməmiş fayl dəyişənləri fayllarla aşağısəviyəli iş üçün nəzərdə tutulub. Bunun köməyi ilə ixtiyari tipə və struktur quruluşa malik fayla müraciət etmək olar. Bu Byte tipli fayl dəyişəninin köməyi ilə simvol faylına müraciətə analogi olaraq yerinə yetirilir.

Tipləşdirilməmiş fayllarla işləmək üçün demək olar bütün prosedur və funksiyalardan istifadə etmək olar. Yalnız Read və Write prosedurlarının əvəzinə BlockRead və BlockWrite prosedurlarından istifadə olunur.

Reset və Rewrite prosedurlarında isə yazının ölçüsünü təyin etmək üçün Word tipli ikinci parametrdən istifadə olunur. Bu parametr olmadıqda yazının ölçüsü susmaya görə 128 bayta bərabər götürülür.

Mətn faylları

Mətn faylların təsvir etmək üçün əvvəldən təyin olunmuş Text tipindən istifadə olunur; Məsələn;

var

Mətn: Text;

Mətn faylında faylın sonu Eof işarəsindən başqa, sətirin sonu Eoln işarəsindən də istifadə olunur.

Mətn fayllarından oxuma və fayla yazma üçün standart Read, Readln, Write və Writeln prosedurlarından istifadənin formatı aşağıdakı kimidir:

Read (f, A,B) ;

Readln (f, C,D) ;

Write (g, ' A =' A, 'B ='B);

Writeln (g, ' C =' , 'D ='D)

Mətn faylları üçün əlavə aşağıdakı prosedur və funksiyalardan istifadə etmək olar:

Append – faylın sonuna elementləri əlavə etmək üçün mövcud faylı açır;

Flush – faylın cari ölçüsünü verir;

Readln – Read proseduru kimi işləyir. Əlavə olaraq cari sətirdə qalan bütün simvolları buraxaraq göstəricini mətn faylının növbəti sətrinə gətirir;

SeekEof – mətn faylı üçün Eof vəziyyətini verir.

SeekEoln - mətn faylı üçün Eoln vəziyyətini verir.

Set TextBuf – mətn faylı üçün daxiletmə-xaricetmə buferi təyin edir;

Writeln – Write proseduru kimi işləyir. Əlavə olaraq mətn faylına Eoln «sətrin sonu» işarəsini yazır.

TurboPaskal – fayllarla iş

Fayl – eyni tipli verilənlər toplanmış yaddaşın müəyyən adlı hissəsidir. Faylın adı nöqtə ilə ayrılmış 2 hissədən ibarətdir. Birinci hissə istənilən ad, 2-ci hissə (genişlənmə) isə faylın növünü(formatını) bildirən 3 simvoldan ibarətdir. Məs. *.txt* genişlənməsi faylın mətn faylı olduğunu göstərir. Faylda verilənlərin sayına heç bir məhdudiyyət qoyulmur.

TurboPaskal-da fayllar da bütün dəyişənlər kimi proqramın əvvəlində - elanlar bölməsində elan olunmalıdır:

Fayl dəyişəninin adı: **File of** <tip>

Fayl dəyişəni faylı identifikasiya edən (tanıtıdır) dəyişənin adıdır.

Məs.: *Text:***File of** Char; - simvollardan ibarət mətn faylı

f: **File of** Integer; - tam ədədlərdən ibarət fayl

Simvollardan ibarət fayl mətn və ya simvol tipli fayl adlanır və o aşağıdakı kimi elan olunur:

Fayl dəyişəninin adı: **TextFile**;

Faylın adı **AssignFile** prosedurasının köməyi ilə verilir:

AssignFile(var f, faylın adı:string);

Faylın adı sistem (MS DOS və ya WINDOWS) tələblərinə uyğun verilir və “_” işarələri arasında yazılır. Məs.:

AssignFile(f, “D:\TP\netice.txt”:**string**); - D diskində TP qovluğunda netice.txt adlanan sətir tipli verilənlərdən ibarət fayl

və ya

fname: “D:\TP\netice.txt”;

AssignFile(f, fname:**string**);

1. Mətn faylına informasiyanın yazılması

Mətn faylına informasiyanın (proqram verilənlərinin qiymətləri) yazılması üçün **Write()** və ya **Writeln()** proseduralarından istifadə olunur. Fayla yazılacaq verilənlər “-” işarələri arasında və ya dəyişənlərin adları ilə verilir və bir-birindən vergüllə ayrılır. Məsələn, f - əgər **TextFile** tipli dəyişəndirsə

Write(f, “Tənliyin kökləri:”,x1,x2);

icra olundan sonra f identifikatorlu fayla Tənliyin kökləri: <x1-in qiyməti>, <x2-nin qiyməti> yazılacaq.

Fayla hər hansı bir informasiyanın yazılması üçün o aşağıdakı 2 cür əməliyyata görə açılmalıdır və faylla iş başa çatan kimi o mütləq bağlanmalıdır:

1. köhnə fayldakı yazıları silib yeniləri ilə əvəz etmək– **Rewrite()** prosedurası.
2. mövcud faylda olan mətnin sonuna yeni informasiyaların əlavə edilməsi – **Append()** prosedurası.
3. Faylın bağlanması üçün **CloseFile()** prosedurasından istifadə olunur.

Misallar:

Var

f: **TextFile**;

i: **Integer**

Begin

AssignFile(f,”a:\test.txt”);

Rewrite(f);

For i=1 to 3 **do**

Writeln(f, “Sətir N”,i);

CloseFile(f);

End;

Bu proqram fraqmenti icra olunandan sonra **a** diskində olan **test.txt** faylindəki bütün informasiyalar silinəcək (**Rewrite**) və başdan 1,2 və 3-cü sətirlərdə müvafiq olaraq *Sətir N1*, *Sətir N2* və *Sətir N3* yazıları yazılacaq. Proqramın bu hissəsi icra olunandan sonra aşağıdakı fraqment icra olunarsa,

.....

```
Begin  
  AssignFile(f,"a:\test.txt");  
  Append(f);  
    For i=4 to 6 do  
      Writeln(f, "Sətir N",i);  
  CloseFile(f);  
End;
```

a diskində olan **test.txt** faylindəki yazılara 4,5və 6-cı sətirlərdə *Sətir N4*, *Sətir N5* və *Sətir N6* yazıları əlavə (**Append**) olunacaq.

2. Mətn faylından informasiyanın oxunması

TurboPaskal proqramından əvvəlcədən mətn faylına (genişlənməsi .txt olan) yazılmış informasiyaları çağırmaq və onlardan istifadə etmək olar. İnformasiya oxunan zaman faylda yazılmış verilənlər proqram dəyişənlərinə mənimsədir. Bundan ötrü verilənlərin qiymətləri yazılmış fayl açılmalıdır. Faylın açılması üçün **Reset(f)** prosedurasından istifadə olunur. Bu proseduradan əvvəl **f** (fayl dəyişəni) mütləq konkret faylla bağlanmalıdır. Məs.:

AssignFile(f,"c:\data.txt"); f – c diskindəki *data.txt* faylıdır

Reset(f); f-lə işarələnən faylı aç

Əgər fayl yoxdursa və ya düzgün açılmayıbsa **IO Result** funksiyasının köməyi ilə bunu yoxlamaq olar:

```
Repeat  
  res:= IO Result;  
  if res<>0  
    then answ:="Faylın açılışında səhv var"  
Until (res=0)
```

Fayldan verilənləri oxumaq (onların qiymətlərinin proqram dəyişənlərinə mənimsədilməsi) üçün **Read()** və **Readln()** proseduralarından istifadə olunur:

Read(fayl dəyişəni, siyahı);

Fayl dəyişəni mütləq **TextFile** tipli olmalıdır. Məs.:

Var

f: **TextFile**;

Siyahıda verilənlərin adları bir-birindən vergüllə ayrılır.

Mətn faylına yazılmış rəqəmlər oxunma zamanı ədəd tipli dəyişənə mənimsədilsə avtomatik olaraq ədədə çevrilir. Bu zaman əgər həmin yerdə rəqəm yazılmayıbsa müraciət səhv sayılır. **Readln()**-dan sonra növbəti dəyişənin qiyməti təzə sətirdən oxunur. Mətn dəyişənində sətirin uzunluğunu **string[n]** (burada *n* –sətirdəki simvolların sayını göstərir) operatoru ilə göstərmək olar. Məs.: **c** diskindən *data.txt* faylından verilənləri oxuyaq:

Tutaq ki, bu faylın 1-ci səhifəsində belə yazı vardır:

12,5,98,10

15,5,95,12

Var

f: **TextFile**;

a,b,c:**Integer**;

Begin

AssignFile(f,"c:\data.txt");

Reset(f);

Read(f,a); - *faylin 1-ci sətirindəki 1-ci qiyməti a-ya mənimsət*

Readln(f,b); - *faylin 1-ci sətirindəki 2-ci qiyməti b-yə mənimsət*

Read(f,c); - *faylin 2-ci sətirindəki 1-ci qiyməti c-yə mənimsət*

End;

Proqramın bu hissəsi yerinə yetiriləndən sonra **a**-nın qiyməti 12-yə, **b**-nin qiyməti 5-ə, **c**-nin qiyməti isə 15-ə bərabər olacaq.

TurboPaskalda fayllarla iş üçün aşağıdakı bir neçə başqa prosedura və funksiya da istifadə olunur:

Raname() prosedurası

Proqramdan istifadə olunan hər hansı bir faylın adının dəyişdirilməsi üçün istifadə olunur. Bu proseduradan istifadə olunmaqdan əvvəl adı dəyişilən fayl mütləq bağlı (**CloseFile()** prosedurası) olmalıdır. Məs.:

```
CloseFile(f);  
AssignFile(f,"c:\data.txt");  
Raname(f,"c:\datason.txt");
```

Yuxarıda verilən təlimatlar faylın **data.txt** kimi saxlanan adını **datason.txt** adına dəyişdi.

Erase() prosedurası

Hər hansı bir faylın diskdən (yaddaşdan) silinməsi üçün istifadə olunur. Məs.:

```
AssignFile(f,"c:\data.txt");  
Erase(f);
```

Yuxarıda verilən təlimatlar **data.txt** faylını yaddaşdan silir.

EOF funksiyası

End of file – faylın sonu deməkdir. Faylın sonunu təyin etmək üçün istifadə olunan məntiqi funksiyadır. Məsələn, əgər biz data.txt faylından bütün verilənləri *a* dəyişəninə mənimsətməklə dövr operatorunun köməyiylə oxumaq istəyiriksə, proqram faylın sonuna çatanda belə dayanır:

```
AssignFile(f,"c:\data.txt");  
While NOT EOF(f) DO  
Read(f, a);  
Proseduralar və funksiyalar
```

1. Prosedurlar və funksiyaların strukturu.

Proseduraların tərtib olunmasını asanlaşdırmaq məqsədilə bütün alqoritmik dillərdə olduğu kimi, Pascalda da bir sıra prosedur və funksiyalar nəzərdə tutulmuşdur. Belə prosedur və funksiyalar standart prosedur və funksiyalar adlanır.

Prosedur və funksiyaları istifadəçi – proqramçı özü yaradırsa onlar istifadəçi, prosedur və funksiyaları adlanırlar.

Prosedur və funksiyaların strukturları proqramın strukturundan başlıqlarına görə fərqlənərək, aşağıdakı kimidir:

```
Procedure Proseduru_adı (formal parametrlər);  
label  
const  
type  
var  
    Procedure { daxili prosedur-lar }  
    -----  
    Function { daxili funksiya-r }  
    -----  
begin
```

operatorlar;
end.

Function funksiyanın _adı (formal parametrlər): nəticənin tipii;

label

const

type

var

Procedure { daxili prosedur-lar }

function

begin

operatorlar;

funk_adı := nəticə;

end.

Funksiya və prosedurlar bir-birindən adlarına və operatorlara görə fərqlənirlər.

Prosedurdan fərqli olaraq, funksiyanın adı proqramlarda həm də dəyişən rolunu oynayır. Alınan nəticə isə funksiyanın adına mənimsədir.

Proqramla prosedur və funksiyalar arasında verilənlərin mübadiləsi iki üsulla aparılır:

1. Qlobal identifikatorlar vasitəsilə;
2. Prosedur və funksiyaların başlıqlarındaki parametrlər vasitəsilə.

Qlobal və lokal identifikatorlar.

Prosedur və funksiyalarda istifadə olunan identifikatorlar lokal və qlobal olmaqla iki növə ayrılırlar.

Lokal identifikatorlar – təsir dairəsi daxil olduqları prosedur və ya funksiyaların kənara çıxmayan identifikatorlardır. Qlobal identifikatorlar isə həm daxil olduğu prosedur və funksiyalarda, həm də onlardan kənarında öz məzmununu saxlayır. Bunu aşağıdakı sxemlə vermək olar:

```
Proqram P1;  
var a1, b1, c1 : integer;  
  Procedure P2;  
    var a2, b2, c2 : integer;  
    Procedure P3;  
      var a3, b3, c3 : integer;  
  
begin
```

```

        { a1, b1, c1, a2, b2, c2 qlobal,
end,

begin
    { a1, b1, c1 qlobal, a2, b2, c2 -lokal parametrlərdir }
-----
end;
begin
    { ancaq a1, b2, c1 parametrlərindən istifadə }
end.

```

Sxemdən görüldüyü kimi $a1, b1, c1$ identifikatorları bütün proqrama daxil olan prosedurlar üçün qlobal, $a2, b2, c2$ parametrləri P2 proseduru üçün lokal, P3 – üçün isə qlobaldır, $a3, b3, c3$ isə P3 –də lokal parametrlərlərdir.

Lakin parametrlərin lokal və ya qloballığı nisbi anlayışdır. Belə ki, bir prosedurda qlobal olan identifikator başqa birisində lokal ola bilər. Bəzən lokal və qlobal parametrlər üst-üstə düşə bilər. Məsələn:

```

Program Parametr;
var x, y, z: real;
  Procedure P;
var x, y, z: integer;
begin
    { x, y, z - integer tipli olacaq }
-----
end;

begin
    { burada x, y, z - real tipli olacaq }
-----
end.

```

Parametrlərin mübadiləsi üsulları

Prosedur və funksiyaları altproqram hesab etsək, proqramla altproqramlar arasında verilənlər mübadiləsi parametrlər vasitəsilə aparılır. Proqramın verilənlər bölməsində altproqramların adını parametrlərlə birlikdə yazmaqla proqramla altproqramlar arasında verilənlər mübadiləsi yaradılır.

Altproqram tərtib olunarkən onların adlarında göstərilən parametrlər formal, müraciət vaxtı göstərilən parametrlər isə faktiki parametrlər adlanır. Formal parametrlərdə tiplər göstərilir, faktiki parametrlərdə isə göstərilmir. Formal və faktiki parametrlər adlarına görə fərqlənə bilərlər, lakin məzmunları eyni olur.

Məlumatlar mübadiləsinə görə parametrlər aşağıdakı növlərə bölünürlər:

1. Parametr – məzmun;
2. Parametr – dəyişən;
3. Parametr – sabit.

Parametr – məzmun.

Bu parametrlər vasitəsilə məlumatlar proqramdan altproqramlara ötürülür və geri proqramma qaytarılmır. Ona görə də bu parametrlər altproqram dilində dəyişilsədə, altproqram xaricində əvvəlki məzmunlarında qalırlar. Belə parametrlərə malik prosedurun başlığı aşağıdakı kimi verilə bilər:

Prosedur Ad (P1, P2, P3 : type 1; P4 : type 2);

Bu prosedur başlığındakı P1, P2, P3 və P4 parametrləri parametr-məzmundurlar. Göründüyü kimi bu parametrlərin öqarşısında heç bir xidmət sözü (məs. var) yoxdur.

Parametr – məzmun parametrləri file tipindən başqa istənilən tiptə ola bilərlər.

Parametr – dəyişən

Bu parametrlər vasitəsilə informasiya altproqramdan proqramma ötürülə bilər. Ona görə də, adətən alınmış nəticələr bu parametrlərə mənimsədir.

Altproqram başlığındakı formal parametrlərin «parametr-dəyişən» olduğunu göstərmək üçün bu parametrlərin qarşısına var sözü əlavə olunur. Məsələn:

Prosedure Ad (var P1, P2, P3 : type 1; var P4 : type 2);

Parametr – dəyişən file tipi də daxil olmaqla istənilən tipli ola bilərlər. Təkcə, parametr-dəyişən konstant ola bilməz, parametr-məzmun isə konstant da ola bilər.

Parametr – konstant

Bu parametrlər vasitəsilə konstantlar altproqramlara göndərilir. Belə parametrli prosedurun başlığı:

Prosedure Ad (const P1, P2, P3 : type 1; const P4 : type 2);

Parametr – konstant kimi file tipindən başqa istənilən tipli sabit və dəyişənlərdən istifadə etmək olar.

Rekursiv funksiya və prosedurlar.

Məsələlərin həlli mərhələlərindən məlumdur ki, həllin proqramından əvvəl, onun alqoritmi tərtib olunur. Alqoritmin kompakt tərtib olunması üçün üsullardan biri də rekursivlikdən istifadə etməkdir. Rekursiv ifadə elə ifadələr ki, onun təyininə özündən istifadə olunur. Məsələn,

$$Y_{n+1} = f(y_n, x)$$

Doğrudan da y -nin $n+1$ –ci addımdakı qiymətinin hesablanması onun n -ci addımdakı qiymətindən asılıdır. Yəni ifadə özü-özünü təyin edir. Məsələn, faktorialın hesablanması misalına baxaq:

$$n! = n(n-1)!$$

Bu ifadənin hesablanmasının Pascal – proqramı aşağıdakı kimi olacaq:

```

Proqram Factorial;
var
  n, i : integer;
function Fact ( n: integer) : longint;
begin Fact: = 1
  for i: = 1 to n do
    Fact: = Fact ( i-1) * i;
  end;
begin
  ('n ədədinin daxil edin n = ');
  Readln (n);
  Writeln ('n ! = ', Fact (n));
  readln (n);
end

```

Standart modullar.

SYSTEM modulu. Bu modula digər modullara daxil olmayan qurulan (vstroenniy) funksiya və prosedurlar, həmçinin standart Pascalın bütün funksiya və prosedurlar daxildir, məs. INC(artırır), DEC(azaldır), GETDIR(kataloqu verir) və s. Uses-də elan olunub-olunmamasından asılı olmayaraq, SYSTEM modulu istənilən proqrama qoşula bilər, ona görə də onun qlobal sabitləri, dəyişənləri və altproqramları Turbo Pascalda qoyulan (vstroennimi) sayılırlar. Ümumiyyətlə SYSTEM modulu əsas proqram kitabxanası sayılır.

DOS modulu. Bu modulda MS-DOS disk əməliyyat sisteminin proqram vasitələrinə təması təmin edən prosedur və funksiyalar toplanmışdır. DOS modulunun heç bir proqramı standart Pascalda təyin edilməyib, ona görə də onlar öz modulunda yerləşdirilmişdir. Bunlardan zaman və tarix, faylların emalı, diskin statusu, kəsilmələrə xidmət və s. prosedur və funksiyalarını göstərmək olar.

WinDOS modulu. ASCIIZ – sətirlərini nəzərə almaqla MS-DOS-un imkanlarından istifadə üçün nəzərdə tutulmuşdur.

Crt Modulu. Burada, mətn rejimində ekranın işinin idarə olunmasını təmin edən funksiya və prosedurlar toplanmışdır. Bum odula daxil olan altproqramların köməyilə kursoru ekranda istənilən mövqeyə gətirmək, xaric edilən simvolların rəngini dəyişmək, pəncərə yaratmaq olar. Buraya həm də klaviaturaya baxmadan işləmək və səsin idarə olunması prosedurları da daxil edilmişdir.

Craph modulu. Bum odula ekranda qrafiki rejimdə işləmək üçün idarəetmə prosedur və funksiyaları, sabitlər və verilənlər tipləri daxildir. Buraya daxil olan altproqramların köməyilə müxtəlif qrafiki təsvirlər yaratmaq və onların izahını istənilən şriftlə vermək olar.

String modulu. Bu modul ASCIIZ sətirlərinin emalı üçün nəzər tutulmuş prosedur və funksiyaları saxlayır.

Overlay modulu. Overlay əməli yaddaşdan elə istifadə qaydasıdır ki, yaddaşın overlay buferi adlanan eyni bir hissəsində növbə ilə müxtəlif lazımı modullar yüklənə bilər. Bütün overlay modulları işə hazır şəkildə diskdə saxlanırlar, əməli yaddaşda isə hər an yalnız bir aktiv modul olur. Qeyd edək ki, bütün standart modullardan yalnız DOS modulu overlay ola bilər.

Printer modulu. Bu modul mətnlərin matrisli printerlərə verilməsini təmin edir. Burada Text tipli LST fayl dəyişəni təyin olunur ki, bu da PRN məntiqi qurğusu ilə əlaqə yaradır. Modul qoşulduqdan sonra, aşağıdakı proqram yerinə yetirilə bilər:

```
Uses Printer;  
begin  
writeln (LST, 'Informatika')  
end.
```

ISTIFADƏÇİ MODULLARI

Modul anlayışı ilk dəfə Ada proqramlaşdırma dilinə daxil edilərək, paket adlanırdı. Standart Pascalda modul olmayıb. Amma bir qədər sonra Ada dilində abstrakt tiplərin və paketlərin inkişafı ilə əlaqədar olaraq, modul Turbo Pascal dilinə daxil edilmişdir.

Modul informasiyanın gizlədilməsi (information hiding) prinsipini əsas götürərək, proqramların yaradılmasında istifadə olunur. Turbo Pascal dilində modullar prosedur,

funksiya və obyekt kitabxanaların yaradılmasında istifadə olunur. Modulun köməyi ilə böyük proqramlar nisbətən kiçik proqram fraqmentlərinə parçalanır.

Modullar proqramlar kimi kompilyasiya olunduğu halda, proqramlardan fərqli olaraq sərbəst icra olunmur. Modullar iki qrupa bölünür:

- Standart modullar;
- istifadəçi modulları.

İstifadəçi modulları – proqramçı tərəfindən yaradılan modullardır. Bu modullar kompilyasiya olunub, təsbih olunduqdan sonra proqramlarda istifadə oluna bilər.

Bu modullar yaradılarkən, aşağıdakılar nəzərə alınmalıdır.

- eyni vaxtda istifadə edilən modulların adları eyni ola bilməz;
- sərlovhədə (başlıq) göstərilən modulun identifikatoru ilkin (.pas) və obyekt fayllarının (.tpu, .tpp, .tpw) adı ilə üst-üstə düşməməlidir;

- modulun identifikatorunun uzunluğu 8 simvoldan çox olarsa, onda o faylın adındakı ilk 8 simvol ilə üst-üstə düşməlidir.

Modullar aşağıdakı hissələrdən ibarətdir:

- modulun başlığı;
- modulun interfeysi;
- realizasiya bölməsi;
- inisiallaşdırma bölməsi.

Modulun başlığı – unit xidmət sözündən və identifikatordan ibarətdir. Məsələn,

Unit Modul 1

Modul yerləşən faylın genişlənməsi .pas olmaqla, adı modulun ilə eyni olmalıdır.

Modulun interfeysi – burada modulun digər istifadəçi və standart modullarla, həmçinin əsas proqramla qarşılıqlı əlaqəsi təsvir olunur.

Modulun interfeysi interface sözü ilə başlayır və aşağıdakı hissələrdən ibarət ola bilər:

- İstifadə olunan modulların təsviri bölməsi;
- Sabitlərin təsviri bölməsi;
- Tiplərin təsviri bölməsi;
- Dəyişənlərin təsviri bölməsi;
- Prosedur və funksiyaların təsviri bölməsi

Realizasiya bölməsi – bu bölmədə cari modulun reallaşdırılması təsvir olunur.

Realizasiya bölməsi implementation xidmət sözü ilə başlayır və inisiallaşdırma bölməsinin başlanğıcı yaxud «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

Inisiallaşdırma bölməsi. Bir çox hallarda modula müraciətdən əvvəl onun inisiallaşdırması həyata keçirilməlidir. Məsələn, Assign prosedurunun köməyi ilə bəzi fayllarla əlaqə, hər hansı dəyişənin adlandırılması və s. Bütün bu əməliyyatlar inisiallaşdırma bölməsi həyata keçirir. Bölm begin və end sözləri arasındakı icra olunan operatorlardan təşkil olunur. Inisiallaşdırma operatorları tələb olunursa begin sözü buraxılır.

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

Turbo Paskal – Modullar

TP-də modullar (UNIT) adlandırılan tamamlanmış proqramlardan istifadə etmək olar. Modullar TP-nin xüsusi toplusuna daxil olan standart modullar və proqramçı tərəfindən yaradılan və toplularda yerləşdirilən ola bilər.

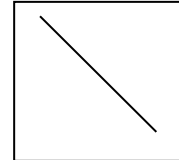
TP-nin standart modulları bunlardır: *Crt, Graph, System, Printer, Dos, WinDos, Strings, Overlay, Graph3, Turbo3*.

1. **CRT** (catode ray tube) modulu **USES Crt**; proqram əmri ilə aktivləşir və monitorla iş üçün proseduraları saxlayır:
 - **GotoXY(x,y)**; - kursuru koordinatları (x,y) olan nöqtəyə keçirir;
 - **ClrScr**; - ekranı təmizləyir;
 - **ClrEol**; - kursurun yerləşdiyi yerdən sağdakı sətiri silir;
2. **Graph** modulu **uses Graph**; proqram əmri ilə aktivləşir və bu modulda çoxlu qrafiki prosedura və funksiyalar vardır. Biz onlardan bəzilərindən istifadə edəcəyik.

Düz xətlərin, çevrələrin və s. təsviri üçün kompüter ekranını qrafik rejimə keçirmək lazım gəlir. Bunun üçün **InitGraph** prosedurasından istifadə olunur.

Sadə bir proqram yazaq. **M i s a l 1.**

```
PROGRAM Misal_1;
uses Graph;
var Gd,Gm: Integer;
BEGIN
  Gd:=VGA; { Qrafik adapter - VGA      }
  Gm:=VGAHi; { Qrafik rejim VGAHi (640x480) }
  InitGraph (Gd,Gm,'.\bgi'); { Qrafik rejimə keçid }
  If GraphResult=grOk
  then { Qrafik rejimə keçilmişsə }
    { Düz xətt parsasını qurmalı }
    Line (0,0,639,479);
    ReadLn
  END.
```



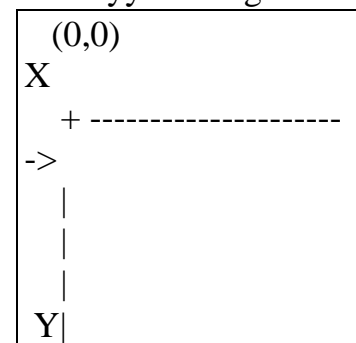
InitGraph prosedurunun üç parametri vardır. İlk iki parametr tam (**integer**) tipli dəyişənlərdir. Birinci parametr qrafik adapterin (yəni informasiyanın ekrana çıxarılmasını idarə edən elektron sxemin) kodudur. Bu ondan ötrüdür ki, IBM tipli kompüterlərdə CGA, EGA, VGA adlanan standart qrafik adapterlərdən istifadə olunur. Yuxarıda baxdığımız proqramda VGA adapterindən istifadə olunur və kompüter VGA sözünü özünə lazım olan tam kodla əvəz edir. Amma bunun bəzə dəxli yoxdur.

Hər qrafik adapter bir neçə qrafik rejimdən istifadə etməyə imkan verir. Bunlar bir-birindən rəqəmlərin sayı və həlledicilik xüsusiyyətlərinə görə fərqlənir (bunlarla aşağıda məşğul olacağıq). İkinci parametr qrafik rejimlərdən hansını işə salmaq lazım gəldiyini göstərir. Biz hələlik ancaq bir qrafik rejimlə – VGAHi rejimi ilə kifayətlənəcəik.

Üçüncü parametr fayla (EGAVGA.BGI adlanan) yolu saxlayan sətirdir. Bu faylda EGA və VGA adapterləri ilə iş üçün drayver saxlanılır. Baxdığımız misalda bu faylın TPBGI altkataloqunda yerləşdiyini görürük.

Ekranda təsvir üçün həmin təsvirin ekrandakı vəziyyətini göstərməyi bacarmalıyıq. Bunun üçün ekranda şəkildə göstərilən koordinat sistemi təsəvvür olunur.

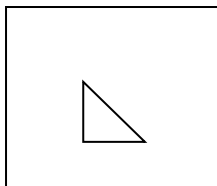
Əslində ekranda nöqtələr çox kiçik düzbucaqlı şəkildə görünür (bu əsl nöqtə olmadığı üçün əsasən *piksel* adlandırılır). Ekranda şaquli və üfiqi istiqamətdə yerləşən nöqtələrin sayı *həlledicilik səviyyəsi* adlanır. VGAHi rejimində ekranın həlledicilik səviyyəsi 640x480 olur. Bu göstərir ki, ekranda üfiqi istiqamətdə 640, şaquli istiqamətdə isə 480 nöqtə yerləşir. Sol yuxarı küncün koordinatları (0,0)-dır. Ekranın ixtifari nöqtəsinin X koordinatı 0 ilə 639 arasında, Y koordinatı isə 0 ilə 479 arasındadır.



Yəqin artıq aydındır ki, **Line** (x1,y1,x2,y2) proseduru ekranda (x1,y1) və (x2,y2) nöqtələrini birləşdirən düz xətt parçasını təsvir edir.

M i s a l 2. Təpə nöqtələri (320, 10), (120, 210), (520, 210) olan üçbucağı təsvir etməli.

```
PROGRAM Misal _2;
uses Graph;
var Gd,Gm: Integer;
BEGIN
  Gd:=VGA; Gm:=VGAHi;
  InitGraph (Gd,Gm,'..\bgi');
  If GraphResult=grOk
  then begin
    Line (120,210,520,210); { Üfiqi parça }
    Line (120,210,320,10); { Sol katet }
    Line (320,10,520,210); { Sağ katet }
    ReadLn
  end;
END.
```



Ekranda təsvir olunan fiquraların rəngli görünməsi üçün **SetColor** prosedurundan istifadə olunur. Rəng özünün kodu ilə bildirilir. Rənglər aşağıdakı kodlara malikdir:

Black=0-qara	DarkGray=8-tünd boz
Blue=1-göy	LightBlue=9-mavi
Green=2-yaşıl	LightGreen=10- açıq yaşıl
Cyan=3-dəniz dalğası rəngi	LightCyan=11-açıq mavi
Red=4-qırmızı	LightRed=12-rozovıy
Magenta=5-yasəməni	LightMagenta=13-açıq yasəməni
Brown=6-qəhvəyi, xurmayı	Yellow=14- yaşıl
LightGray=7-açıq boz	White=15-ağ

Proqramlarda rəngdərin identifikatorlarından da istifadə etmək olar

M i s a l 3. Misal 2-də təsvir olunan üçbucağın tərəflərini müxtəlif rənglərlə təsvir etməli

```
PROGRAM Misal _3;
uses Graph;
var Gd,Gm: Integer;
BEGIN
  Gd:=VGA; Gm:=VGAHi; InitGraph (Gd,Gm,'..\bgi');
  If GraphResult=grOk
  then begin
    SetColor (LightMagenta); { A1q yasəməni }
    Line (120,210,520,210); { Üfiqi parça }
  end;
END.
```

```

SetColor (LightCyan); { Üvet – açıq mavi}
Line (120,210,320,10); { Sol katet      }
Set Color (Green);    { Yaşıl rəng      }
Line (320,10,520,210); { Sağ katet      }
ReadLn

```

end

END.

M i s a l 4. *Konsentrik çevrələr.* Çevrələrin təsviri üçün **Circle** prosedurundan istifadə olunur. Bu prosedur üç tamqiymətli parametrdən istifadə edir. Bunlar, mərkəzi koordinatlarını və radiusu bildirir.

```

PROGRAM Misal_4;

```

```

uses Graph;

```

```

const CenterX=320;

```

```

    CenterY=240;

```

```

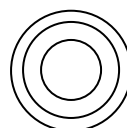
var Gd,Gm: Integer;

```

```

    i : Integer;

```



```

BEGIN

```

```

    Gd:=VGA; Gm:=VGAHi; InitGraph (Gd,Gm,'..\bgi');

```

```

If GraphResult=grOk

```

```

    then begin

```

```

        For i:=0 to 23 do

```

```

            Circle (CenterX,CenterY,i*10);

```

```

            ReadLn

```

```

        end

```

```

END.

```

Çoxlu parçalar saxlayan mürəkkəb şəkillərin təsviri zamanı tərs problem – bütün nöqtələrin koordinatlarının hesablanması qarşıya çıxır. **LineRel** prosedurundan istifadə edildikdə cari nöqtəyə nəzərən hər iki koordinat üzrə yerdəyişmələri – nisbi koordinatları vermək kifayətdir.

İz qoymadan nisbi yerdəyişmə üçün **MoveRel** prosedurundan istifadə olunur. Cari nöqtənin başlanğıc nöqtəsinin koordinatları **MoveTo** proseduru ilə verilir.

Digər qrafik prosedur və funksiyalar barədə "kömək" (Help) sistemindən məlumat almaq olar. Bunun üçün **Help** menyusundan **Standard units** bəndini seçmək lazımdır. bəzi bəndlərinin adının tərcüməsini veririk:

Color Constants

Qrafik konstantlar

Fill Pattern Constants

Rənglənmə üçün konstantlar

Graphics Drivers

Qrafik drayverlər

Graphics Modes for Each Driver

Hər drayver üçün qrafik rejimlər

Proqram modullarının yaradılması

Modul (**UNIT**) TP-dilində yazılmış hər hansı bir tamamlanmış proqramdır. UNIT kimi yaradılan proqram ayrıca kompilyasiya olunur və xüsusi modullar toplusuna əlavə olunur. Belə yolla yaradılmış modula başqa proqram mühitlərindən də standart modullara analogi müraciət olunaq istifadə edə bilərik.

Modulun strukturu:

UNIT	<i>Modulun adı;</i>
INTERFACE	<i>Elanlar bölməsinin başlanğıcı;</i>
USES	<i>İstifadə olunan modulların siyahısı;</i>
LABEL	<i>Qlobal nişanların elanı;</i>
CONST	<i>Qlobal konstantların elanı;</i>
TYPE	<i>Qlobal tiplərin elanı;</i>
VAR	<i>Qlobal dəyişənlərin elanı;</i>
PROCEDURE	<i>İstifadə olunan proseduraların adları;</i>
FUNCTION	<i>İstifadə olunan funksiyaların adları;</i>
IMPLEMENTATION	<i>Realizasiya bölməsinin başlanğıcı;</i>
USES	<i>Realizasiya vaxtı istifadə olunan modulların siyahısı;</i>
LABEL	<i>Qapalı qlobal nişanların elanı;</i>
CONST	<i>Qapalı qlobal konstantların elanı;</i>
TYPE	<i>Qapalı qlobal tiplərin elanı;</i>
VAR	<i>Qapalı qlobal dəyişənlərin elanı;</i>
PROCEDURE	<i>İstifadə olunan proseduraların adları;</i>
FUNCTION	<i>İstifadə olunan funksiyaların adları;</i>
BEGIN	
	<i>Modulun gövdəsi</i>
END.	

Turbo Pascal- da qrafika

TP-da proqram icra olunarkən kompüter mətn və ya qrafikirejimdə ola bilər.

Adi halda, yəni Turbo Pascal yüklənəndə kompüter mətn rejimində olur. Mətn rejimindən qrafiki rejimə keçmək üçün

Unit Graph (grDriver, grMode, grPath); prosedurundan istifadə olunur və əməliyyat qrafiki rejimin insializasiyası adlanır.

Burada grDriver – parametri integer tipli sabit, məsələn VGA olub, monitorun işini idarə edən adapteri göstərir.

grMode – parametri də integer tiplidir və videosistem rejimini göstərir. Bu regim təsvirin keyfiyyətini təyin edir.

grPath – parametri string tipli olub, drayverin diskdəki yolunu göstərir və bu drayverlər Turbo Pascalda BGI qovluğunda saxlanır.

Insializasiyanın normal yerinə yetirilməsi GraphResult funksiyası vasitəsilə yoxlanır. Inisializasiyadan sonra bu funksiyanın qiyməti grOk olarsa, insializasiya yenidən işlənməlidir.

Qrafik rejimdən mətn rejiminə keçmək üçün CloseGraph əmrindən istifadə olunur.

Qeyd edək ki, mətn rejimində monitoru idarə edən prosedur və funksiyalar TurboPascalın Crt modulunda, qrafiki rejimdə monitoru idarə edən prosedur və funksiyalar isə Graph modulunda saxlanır.

Turbo Pascalın əsas qrafiki prosedur və funksiyaları

Turbo Pascalda müxtəlif qrafiklərin qurulması üçün bir çox prosedur və funksiyalar vardır ki, bunlardan ən əsasları aşağıdakılardır:

1. GetMax X, GetMax Y funksiyaları ekranın işçi oblastının sağ aşağı küncünün koordinatlarını göstərir. VGA rejimində bu koordinatlar (639, 479) – dur. Sol yuxarı küncün koordinatları isə (0,0)-dır.

2. GetMax X, GetMax Y funksiyaları kursurun cari koordinatlarını göstərir.

3. Move To(X: integer, y integer) funksiyası kursuru (x, y) nöqtəsinə gətirir.

4. Set Color (rəngin _adı) ekrana çıxarılacaq elementin (nöqtə, xətt və s.) rəngini müəyyən edir. (Rənglər rəng cədvəlindən seçilir)

5. PutPixel (x,y, rəng) – (x, y) nöqtəsini çəkir.

6. Line (x₁,y₁, x₂,y₂) - (x₁, y₁) nöqtəsini birləşdirən düz xətti çəkir.

7. Line To (x, y) – kursurun cari koordinatı ilə (x, y) nöqtəsini birləşdirən düz xətt çəkir.

8. Circle (x, y, r) – mərkəzi (x, y) nöqtəsində olan r radiuslu çevrə çəkir.

9. Ellipse (x, y, başlanğıc_bucaq, son_bucaq, Radius X, radius Y)- ellips sektoru çəkir. (x, y) – ellipsin mərkəzi, Radius X – üfuqi Radius Y isə şaquli radiusları göstərir.

Əgər başlanğıc_bucaq = 0⁰, son_bucaq=360⁰ olarsa tam ellips alınar, Radius X = Radius Y olarsa, çevrə sektoru olunacaq.

10. Restangle (x₁,y₁, x₂,y₂) – diaqonalı (x₁,y₁) və (x₂,y₂) nöqtələrini birləşdirən düzbucaqlını verir.

Qrafik rejimində mətnlərin çapı

1. Write (sətir), Writeln (sətir) – prosedurları qrafiki rejimdə də mətn rejiminə olduğu kimi çap kimi edir. Qeyd edək ki, bu vaxt şriftləri, onların ölçülərini, çapın istiqamətini dəyişmək mümkün deyil.

2. OutText (sətir) – kursurun cari mövqeyindən başlayaraq bir sətiri çap edir.

3. OutText XY (x, y, sətir) – bu funksiya da bir sətir çap edir. (x,y) – nöqtəsi çap oblastının sol yuxarı küncünü göstərir.

4. SetText Style (Şrift, Çap_istiqaməti, Ölçü) – proseduru OutText və OutText XY- vasitəsilə çap olunan mətnin xüsusiyyətlərini müəyyən edir.

- Şrift parametrləri 0, 1, 2, 3, 4 qiymətlərini ala bilər.

- Çap _istiqaməti = 0 olduqda OutText və OutText XY prosedurları üfqi istiqamətdə, çap_istiqaməti =1 olduqda isə şaquli isə şaquli istiqamətdə çap edəcəkdir.

- Ölçü – şriftin ölçüsünü göstərir.

Məsələn:

```
SetTextStyle (0,1,2);
```

```
OutText XY (100, 200, `Şaquli istiqamətdə çap `);
```

```
SetTextStyle (0,0,2);
```

OutText XY (100, 200, `Üfqİ istiqamətdə çap `);

Mühazirə 15: Delphi mühiti. Layihələr

Delphi-də proqramların yaradılmasına **IDE** (*Integrated Development Environment*) - **İntegrallaşdırılmış iş mühitində** yerinə yetirilir. Onun köməyi *ilə əlavə* layihələndirilir, proqram kodları yazılır və sazlanır.

Delphi-nin **IDE** mühiti çoxpəncərəli sistemdən ibarətdir. İntegrallaşdırılmış iş mühitinin görünüşü onun sazlanmasından asılı olaraq müxtəlif ola bilər. Delphi yükləndikdə ekranda ilkin olaraq dörd pəncərə görünür :

- ❖ *Əsas pəncərə* - Delphi - Project1;
- ❖ *Obyektlər inspektor pəncərəsi* - Object Inspector;
- ❖ *Forma konstruktör pəncərəsi* - Form1;
- ❖ *Kod redaktör 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 bükükdə Delphi interfeysi və onunla birlikdə bütün açıq pəncərələr bükülür, onu bağladıqda isə Delphi ilə iş qurtarır.

Əsas pəncərə aşağıdakı hissələrdən ibarətdir:

Əsas menyü:

Alətlər paneli:

Komponentlər palitrası.

Əsas menyü Delphi-nin versiyalarından asılı olaraq bir neçə menyudan ibarət olur ki, bu menyularda Delphi-nin müxtəlif funksiyalarını icra etməyə imkan verən çoxlu əməllər toplanmışdır.

Alətlər paneli əsas menyunun altında, əsas pəncərənin sol hissəsində yerləşir. Bu paneldə yerləşən düymələr, Windows-un bütün pəncərələrində olduğu kimi, daha çox işlədilər əməlləri 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.

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 ayn-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 konfigurasiyalarında bu komponentlərin bəziləri olmaya bilər. Komponentlər palitrası 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 konstrukturu 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 konstrukturu "kadr arxasında" qalır, proqramçı isə formanın özü ilə işləyir və ona görə də Forma konstrukturu sadəcə olaraq Forma pəncərəsi və ya Forma deyilir.

Kod redaktoru pəncərəsi Forma konstrukturu 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 operatorlar 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 konstrukturu və Kod redaktoru pəncərələrinin yerini dəyişmək üçün F12 klavişini və ya lazım olan pəncərənin oblastında mausun düyməsini basmaq kifayətdir. Mausun göstəricisini sərlövhə sətirində 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 **Forml** 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, sərəlvhənin dəyişdirilməsi və s.) göstərmək olar.*

Layihənin kompilyasiyası və yerinə yetirilməsi

*Delphi layihəni kompilyasiya etdikdə susmaya görə onun fayllarını **C:\Program Files\Delphi\Bin** qovluğunda yerləşdirir. Ona görə də programı 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 əməllərini icra edir. İndi isə **File** menyusundan **SaveAll** əmrini icra etməklə layihəni yaratdığımız qovluqda yadda saxlaym. Bu məqsədlə Sizə, əvvəlcə yuniti - unitl.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 basm. Delphi layihəni icra olunan fayla kompilyasiya edəcəkdir və görəcəksiniz ki, Sizin qovluğunuza daha 2 fayl (Projectl. exe və Unitl.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, onlann ehtiyat surətləri yarana bilər: ~DP - DPR fayllar üçün, ~PA - PAS fayllar üçün. Bu fayllarm 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 <Projectl> (Layihə/Kompilyasiya <Layihəl>) əmri ilə və ya Ctrl+F9 klavişlərini basmaqla başlayır. Bu əmrdə layihənin adı yazılır (ilk dəfə Projectl 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 yerləşdirilmiş 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 <Projectl> (Layihə/Yığımaq <Layihəl>) əmri icra olunmalıdır.

Layihənin xarakteristikaları. Layihə faylı

Layihə faylı ən əsas fayldır və əslində elə proqram 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 Projectl 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 faylı) adı layihə faylının adı ilə eyni olur.

Layihə faylının üsuli 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 operatorla ibarətdir. Bu operatorlar əslində aşağıdakı metodları çağırır:

Application. Initialize - bütün Delphi əlavələrində ən birinci çağırılan 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 fayh eyni və ya oxşar koda malik olduğu üçün, biz gələcəkdə məsələlər həll etdikdə, bu faylın məzmununu göstərməyəcəyik.

Forma modulu faylı

Bu faylda formanın siniflərinin təsviri yerləşir. Susmaya görə bu fayhın adı Unit1.pas olur. Faylın birinci sətiri unit işçi sözü ilə başlayır. Bütün modulların da birinci sətiri bu işçi sözlə başlayır. Deməli, bu faylda program modulu yerləşir və ona yunit də deyirlər. Əslində modul layihədə yeganə program vahididir ki, o, məhz programçının özü tərəfindən yaradılır, başqa sözlə, layihəçinin bütün yaradıcı fəaliyyəti məhz özünü bu modulda əks etdirir. Lakin, bu o demək deyildir ki, programçı modulun bütün kodlarını özü yazır. Burada da Delphi öz köməyini əsirgəmir: modulun strukturu Delphi tərəfindən artıq hazır şəkildə programçıya təqdim olunur, hadisələrə uyğun bir neçə sətiri də Delphi özü yazır və modulda hadisə emaledicisinin yerinə yetirəcəyi əməliyyatlara aid kodları əlavə ediləcəyi yeri də göstərir. Programçı, modulun Delphi tərəfindən mətn kursoru ilə göstərilən hissəsinə müvafiq program kodları əlavə edir. Ümumiyyətlə, layihələndirmə (programlaşdırma) 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)** dialog pəncərəsi peyda olur ki, bu pəncərədə Unitl seçərək **Ok** düyməsini basmaq lazımdır. Yeri gəlmişkən qeyd edək ki, elə bu pəncərədən **Projectl** seçməklə layihə faylmm 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 yaradılır.

Forma faylı

Hər bir forma üçün layihənin tərkibində, avtomatik olaraq, formanın təsviri faylı yaranır (**.dfm** tipli). Bu faylm adı da yunitin adı ilə eyni olur. Forma fayhna 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 formanm özü haqqında informasiya təsvir olunur: o, harısı ölçüdədir. ekranda hansı vəziyyətdədir, onun üzərində hansı komponentlər vardır və bu komponentlərin xarakteristikalan necədir. Proqramçı Forma konstruktorunda obyektleri yerləşdirdikdə və Obyektler inspektorunda həmin komponentlərin xassələrini müəyyənləşdirdikdə bu məlumatlar avtomatik olaraq forma faylında yadda saxlanılır. 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 fayhnu bağladıqdan sonra, **View/Forms (Baxış/Formalar)** əmrini və ya **Shift+F12** klavişlərini basmaq lazımdır. İndi isə nümunə üçün, üzərində **Buttonl** düyməsi yerləşdirilmiş **Forml** forması üçün, forma fayhının mətninə baxaq (bu nümunədə həmin düymə üçün **OnClick** - düyməbasma hadisə emaledicisi də yazılmışdır):

```
Object Forml:TForml
  Left=192
  Top=107
  Width=54 4
  Height=375
  Caption='Forml'
```

```

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'` sətirində, yəni düymənin sərlövhasını müəyyən etmə sətirində `' 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.

Resurslar faylı

Layihəni birinci dəfə yadda saxladıqda, avtomatik olaraq, tipi `.res` olan resurslar faylı yaranır. Bu faylın adı da layihə fayllarının 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.

Layihə parametrləri faylı

Bu fayl, Project/Options... (Layihə/Parametrlər...) əmri ilə açılan dialog pəncərəsində sazlanan parametrlərin əsasında Delphi tərəfindən avtomatik olaraq yaradılır. Dialog 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, Projectl.opt olur. Nümunə üçün layihə parametrləri faylının aşağıdakı fraqmentini göstərmək olar:

[Compiler]

A=

B=0

C=

D=

E=0

Modul faylları

Yuxandakı 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 ayn-ayn fayllarda saxlanılır. 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.

Mühazirə 16 : 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, konstruksiyəmə 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 saniyə ölçəni 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 interfeysinə yaradılması prosesi

*ənənəvi proqramlaşdırmadan daha çox konstruksiyətmə işlərinə oxşayır. Ona görə də əlavənin yaradılması prosesi proqramlaşdırma yox, **konstruksiyalaşdırma** adlanır.*

Komponentlər palitrası və forma

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 adı 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 basıb 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 funksiya 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 yerləşdirilsə, onlar ardıcıl olaraq nömrələnir:

Button1: TButton; Button2: TButton; Button3: TButton;

Button düyməsi üçün təsvirlər faylına avtomatik olaraq belə kod yazıla müəyyən yazıla bilər.

```
Object Button1: TButton
  Left=88
  Top=120
  Width=75
  Height=25
  Caption='Button1'
  TabOrder=0
End
```

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 kiçiltildikdə 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. 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 üfüqi və ya şaquli ölçüləri dəyişdirdikdə tərəflər üzərindəki markerlərdən, komponentin ölçülərini mütənasib dəyişdikdə isə küncələrdəki markerlərdən istifadə etmək lazımdır. Forma üzərində komponentin yerini dəyişdirdikdə isə mausun göstəricisini komponentin daxilində yerləşdirib mausun sol düyməsini basıb 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 şəkilçəkm redaktorundakı əməliyyatları xatırladır. Eyni zamanda bir neçə komponenti seçmək üçün, *Shift* klavişini basıb 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 (*S* 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 üfüqi və şaquli şəbəkə addımdan Environment Options (Mühitin parametrləri) dialoq pəncərəsinin Preferences (Üstünlüklər) səhifəsində müəyyənləşdirilir. Bu pəncərəni çağırmaq üçün Tools/Environment Servis/Mühitin parametrləri) əmrini icra etmək lazımdır.

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 sətirində həmin komponentin adı və tipi (məsələn, Button: TButton), 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 analogi qaydada müəyyənləşdirilir. Lakin forma seçilmiş vəziyyətdə olduqda markerlərlə əhatələnmiş. Ona görə də formanı seçmək üçün onun komponentlər olmayan boş sahəsində mausun sol düyməsini basmaq kifayətdir. Bu və ya digər komponenti Obyektlər inspektorunun birinci sətirində 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 görünür. 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 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 alır. 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 seçilsə, 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 sətiri 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 Forml sərlovhəsini "Bloknot adlandırmaq üçün yunitdə

Forml.Caption: = 'Bloknot'; kodu yazmaq lazımdır. Lakin, bu çox vaxt aparır, digər tərəfdən belə təyinetmə yalnız layihə yerinə yetirildikdən sonra qüvvəyə minir, layihələndirmə zamanı isə görünür. Buna baxmayaraq, proqramda belə təyinetmələr çox tez-tez tətbiq edilir.

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 qəbul olan, Windows sistemi altında işləyə bilən, mükəmməl pəncərəli əlavə yaratmaqdan ibarətdir. Bunun üçün proqramçı komponentin yerinə yetirəcəyi funksiyaları - 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

sərlövhesini **Bağlamaq (Caption-Bağlamaq)** adlandıraraq. 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 proseduru göstərmək lazımdır. Hadisə emaletmə proseduru 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 ButtonClick (Sender: TObject);

sətiri yazılır və cursor, prosedurda **Button1** düyməsi basıldıqda yerinə yetiriləcək əməliyyatın kodlarının əlavə ediləcəyi mövqedə yerləşir. Proqramçı bu kodları özü yazır. Bu hadisə nəticəsində forma bağlanmalı olduğu üçün, cursorun 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 ButtonClick(Sender: TObject);  
  
private  
    {private declarations}  
  
public  
    {public declarations}  
  
end;
```

Forml:TForml;

*Implementation {\$R *.DFM}*

procedure TForm1.Button1Click(Sender:TObject) ; begin

Forml.Close; // Biz yalnız bu sətiri yazdıq

end;

end.

Xatırladaq ki, modulun bu təsvirində yalnız `Forml.Close;` sətiri 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 yə kompilyasiya etdikdə, həmin prosedur avtomatik olaraq bütün fayllarda pozulacaqdır.

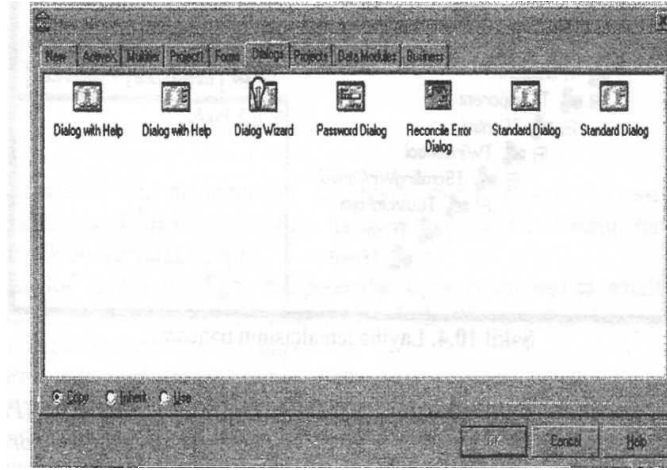
Xatırladaq ki, modulun bu təsvirində yalnız `Forml.Close;` sətiri proqramçı tərəfindən yazılmışdır, qalan bütün sətirlər Delphi tərəfindən avtomatik olaraq yaradılmışdır.

Analoji qayda ilə digər komponentlər üçün başqa hadisə emaledicibri 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 yə kompilyasiya etdikdə, həmin prosedur avtomatik olaraq bütün fayllarda pozulacaqdır. Yeni emaledici yaratmaq əvəzinə, əgər varsa, mövcud emaledicini istifadə etmək olar. Bunun üçün Obyektlər inspektorunda, hadisənin qiymətlər oblastindəki oxun üzərində, mausun düyməsini basmaqla açılan prosedurlar -ismindən onu seçmək lazımdır. Hər hadisəyə onunla eyni tipli olan emaledici təyin etmək olar. Siyahıdan lazım olan proseduru seçdikdən sonra, o, hadisə *təyin* olunur. Eyni bir proseduru müxtəlif komponentlər ilə əlaqələndirmək olar. Belə prosedur ümumi emaledici adlanır və onunla əlaqədar istənilən hadisə baş verdikdə çağrılır.

İntegrallaşdırılmış iş mühitinin vasitələri

Bu vasitələrin köməyi ilə istifadəçi integrallaşdırılmış mühitin parametrlərini, məsələn, Kod redaktorunun 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)* dialog 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 hissəyə birbaşa



keçmək və s. kimi əməliyyatlar *Layihə icmalçısının (Project Browser)* köməyi ilə yerinə yetirilir. Layihə icmalçısı *View/Browser (Baxış/Layihə icmalçı)* əmri ilə çağrılır. Layihə icmalçısının *Exploring <...> (Tədqiqat <...>)* pəncərəsində 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)* dialog pəncərəsinin *Explorer (Bələdçi)* səhifəsindən və yaxud *Layihə icmalçısının Properties (Xassələr)* kontekst menyusundan daxil olmaq olar. Delphi eyni bir obyektə şablon kimi dəfələrlə istifadə etməyə imkan verir. Belə obyektlər xüsusi yerdə (*Repository*) saxlanılır. *Obyektlərin saxlandığı yer File/New (Fayl/Yeni fayl)* əmri ilə çağrılır (şəkil 1). Burada ən müxtəlif obyektlər, məsələn, əlavə, forma, hesabat şablonları və Forma ustaları saxlanılır. Müxtəlif obyektlər qruplaşdırılaraq pəncərədə görünən səhifələrdə yerləşdirilmişdir. Layihəyə yeni obyektə əlavə etmək üçün üç üsul təklif edilir:

Copy - obyektin surəti layihəyə əlavə edilir;

Inherit - obyektədən 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. İntegrallaşdırılmış mühidə yeni əlavə yaratdıqda Delphi interfeysini bağlamadan, köhnə layihəni **File/Close All** (**Fayl/Hamısını bağlamaq**) əmri ilə bağlamaq məqsədəuyğundur.

İntegrallaşdırılmış iş mühitində ən çox istifadə edilən əmrlər

Delphi integrallaşdırılmış iş mühitində ən çox istifadə edilən əmrlər Cədvəl 1.-də göstərilmişdir.

Cədvəl 1. Delphi integrallaşdırılmış iş mühitində ən çox istifadə edilən əmrlər

Əmrin adı	Klavislər	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
ViewForms	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 Project	Ctrl+F9	Layihənin kompilyasiyası
Project/Compile All Projects		Bütün layihənin kompilyasiyası
Project/Build Project		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

<i>View/Browser</i>	<i>Shift+Ctrl+B</i>	Layihə icmalçısının ekranda təsviri
<i>Tools/Repository</i>		Obyektlərin saxlanıldığı yerin sazlanması
	<i>F12</i>	Forma konstruktoru ilə Kod redaktorunun yerlərinin dəyişdirilməsi
<i>Save Project As...</i>		Layihəni necə yadda saxlamaq
<i>Save All</i>	<i>Shift+Ctrl+S</i>	Bütün layihəni saxlamaq
<i>Close AU</i>		Bütün layihəni bağlamaq

Mühazirə 16 : 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 saniyə ölçəni və ya Table verilənlər yığımları 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 interfeysinə yaradılması prosesi ənənəvi proqramlaşdırmadan daha çox konstruksiyaetmə işlərinə oxşayır. Ona görə də əlavənin yaradılması prosesi proqramlaşdırma yox, **konstruksiyalaşdırma** adlanır.*

Komponentlər palitrası və forma

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 adı 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 basıb 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 funksiya 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 Buttonl: TButton; Editl: TEdit; kimi olacaqdır. Əgər forma üzərində bir neçə eyni komponent yerləşdirilərsə, onlar ardıcıl olaraq nömrələnir:

Buttonl: TButton; Button2: TButton; Button3: TButton;

Button düyməsi üçün təsvirlər faylma avtomatik olaraq belə kod yazıla müəyyən yazıla bilər.

```
Object Buttonl: TButton
  Left=88
  Top=120
  Width=75
  Height=25
  Caption='Buttonl'
  TabOrder=0
End
```

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 kiçiltildikdə 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. 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 üfüqi və ya şaquli ölçüləri 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 basıb 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 şəkilçəkm redaktorundakı əməliyyatları xatırladır. Eyni zamanda bir neçə komponenti seçmək üçün, Shift klavişini basıb saxlayaraq, hər bir komponent üzərində mausun sol düyməsini basmaq lazımdır.

Susmaya görə, forma üzərində komponentlər nöqtəli şəbəkə xətlərinə görə düzləndirilir. Bu integrallaşdırılmış iş mühitinin parametrlərində *Snap to Grid* (*Şəbəkəyə görə düzləndirmə*) parametri qarşısında bayraq (*S* 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 üfüqi və şaquli şəbəkə addımdan Environment Options (Mühitin parametrləri) dialoq pəncərəsinin Preferences (Üstünlüklər) səhifəsində müəyyənləşdirilir. Bu pəncərəni çağırmaq üçün Tools/Environment Servis/Mühitin parametrləri) əmrini icra etmək lazımdır.

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 sətirində həmin komponentin adı və tipi (məsələn, Button: TButton), 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 əhatələnmiş. Ona görə də formanı seçmək üçün onun komponentlər olmayan boş sahəsində mausun sol düyməsini basmaq kifayətdir. Bu və ya digər komponenti Obyektlər inspektorunun birinci sətirində 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 görünür. 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 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 alır. 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 seçilsə, 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 sətiri 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 Forml sərlovhəsini "Bloknot adlandırmaq üçün yunitdə

Forml.Caption: = 'Bloknot'; kodu yazmaq lazımdır. Lakin, bu çox vaxt aparır, digər tərəfdən belə təyinetmə yalnız layihə yerinə yetirildikdən sonra qüvvəyə minir, layihələndirmə zamanı isə görünür. Buna baxmayaraq, proqramda belə təyinetmələr çox tez-tez tətbiq edilir.

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 qəbul olan, Windows sistemi altında işləyə bilən, mükəmməl pəncərəli əlavə yaratmaqdan ibarətdir. Bunun üçün proqramçı komponentin yerinə yetirəcəyi funksiyayı - 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 sərlovhəsini **Bağlamaq (Caption-Bağlamaq)** adlandırmaq. 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 proseduru göstərmək lazımdır. Hadisə emaletmə proseduru 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 ButtonClick (Sender: TObject);

sətiri yazılır və cursor, prosedurda **Buttonl** düyməsi basıldıqda yerinə yetiriləcək əməliyyatın kodlarının əlavə ediləcəyi mövqedə yerləşir. Proqramçı bu kodları özü yazır. Bu hadisə nəticəsində forma bağlanmalı olduğu üçün, cursorun 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) Buttonl:TButton;
procedure ButtonClick(Sender: TObject);*

private

{private declarations}

Public

{public declarations}

end;

Form1:TForm1;

*Implementation {\$R *.DFM}*

procedure TForm1.ButtonClick(Sender:TObject) ; begin

Form1.Close; // Biz yalnız bu sətiri yazdıq

end;

end.

Xatırladaq ki, modulun bu təsvirində yalnız `Form1.Close`; sətiri 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 yə kompilyasiya etdikdə, həmin prosedur avtomatik olaraq bütün fayllarda pozulacaqdır.

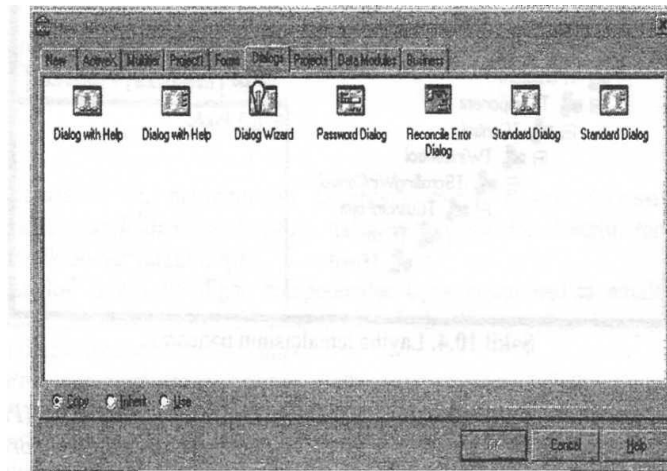
Xatırladaq ki, modulun bu təsvirində yalnız `Form1.Close`; sətiri proqramçı tərəfindən yazılmışdır, qalan bütün sətirlər Delphi tərəfindən avtomatik olaraq yaradılmışdır.

Analoji qayda ilə digər komponentlər üçün başqa hadisə emaledicibri 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 yə kompilyasiya etdikdə, həmin prosedur avtomatik olaraq bütün fayllarda pozulacaqdır. Yeni emaledici yaratmaq əvəzinə, əgər varsa, mövcud emaledicini istifadə etmək olar. Bunun üçün Obyektlər inspektorunda, hadisənin qiymətlər oblastindəki oxun üzərində, mausun düyməsini basmaqla açılan prosedurlar -ismindən onu seçmək lazımdır. Hər hadisəyə onunla eyni tipli olan emaledici təyin etmək olar. Siyahıdan lazım olan proseduru seçdikdən sonra, o, hadisə *təyin* olunur. Eyni bir proseduru müxtəlif komponentlər ilə əlaqələndirmək olar. Belə prosedur ümumi emaledici adlanır və onunla əlaqədar istənilən hadisə baş verdikdə çağrılır.

İntegrallaşdırılmış iş mühitinin vasitələri

Bu vasitələrin köməyi ilə istifadəçi inteqrallaşdırılmış mühitin parametrlərini, məsələn, Kod redaktorunun 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)* dialog 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 hissəyə birbaşa



keçmək və s. kimi əməliyyatlar *Layihə icmalçısının (Project Browser)* köməyi ilə yerinə yetirilir. Layihə icmalçısı *View/Browser (Baxış/Layihə icmalçı)* əmri ilə çağrılır. Layihə icmalçısının *Exploring <...> (Tədqiqat <...>)* pəncərəsində 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)** dialog pəncərəsinin **Explorer (Bələdçi)** səhifəsindən və yaxud *Layihə icmalçısının Properties (Xassələr)* kontekst menyusundan daxil olmaq olar. Delphi eyni bir obyektə şablon kimi dəfələrlə istifadə etməyə imkan verir. Belə obyektlər xüsusi yerdə (**Repository**) saxlanılır. **Obyektlərin saxlandığı yer File/New (Fayl/Yeni fayl)** əmri ilə çağrılır (şəkil 1). Burada ən müxtəlif obyektlər, məsələn, əlavə, forma, hesabat şablonları və Forma ustaları saxlanılır. Müxtəlif obyektlər qruplaşdırılaraq pəncərədə görünən səhifələrdə yerləşdirilmişdir. Layihəyə yeni obyektə əlavə etmək üçün üç üsul təklif edilir:

Copy - obyektin surəti layihəyə əlavə edilir;

Inherit - obyektədən 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. İntegrallaşdırılmış mühitdə yeni əlavə yaratdıqda Delphi interfeysini bağlamadan, köhnə layihəni **File/Close All** (**Fayl/Hamısını bağlamaq**) əmri ilə bağlamaq məqsədəuyğundur.

İntegrallaşdırılmış iş mühitində ən çox istifadə edilən əmrlər

Delphi integrallaşdırılmış iş mühitində ən çox istifadə edilən əmrlər Cədvəl 1.-də göstərilmişdir.

Cədvəl 1. Delphi integrallaşdırılmış iş mühitində ən çox istifadə edilən əmrlər

Əmrin adı	Klavislər	Əmrin yerinə yetirdiyi funksiya
<i>File/New</i>		Yeni obyekt (əlavə, forma, yunit və s.) yaratmaq
<i>Run/Run</i>	<i>F9</i>	Layihənin yerinə yetirilməsi
<i>View/Units...</i>	<i>Ctrl+F12</i>	Modulun ekranda təsviri
<i>ViewForms</i>	<i>Shift+F12</i>	Forma konstruktorunun ekranda təsviri
<i>Component/Configure Palette</i>		Komponentlər palitrasının sazlanması
<i>View/Code Explorer</i>		Kod bələdçisinin ekranda təsviri
<i>View/Object Inspector</i>	<i>F11</i>	Obyektlər inspektorunun ekranda təsviri
<i>Project/View Source</i>		Layihə faylının ekranda təsviri
<i>Project/Options...</i>	<i>Ctrl+Shift+F11</i>	Layihənin parametrlərinin təyini
<i>Project/Compile Project</i>	<i>Ctrl+F9</i>	Layihənin kompilyasiyası
<i>Project/Compile All Projects</i>		Bütün layihənin kompilyasiyası
<i>Project/Build Project</i>		Layihənin yığılması
<i>Project/Build All Projects</i>		Bütün layihənin yığılması
<i>Run/Program Reset</i>	<i>Ctrl+F2</i>	Sonsuz dövr etmə zamanı proqramın dayandırılması

<i>Tools/Environment Options</i>		Delphi interfeysinin parametrlərinin sazlanması
<i>View/Project Manager</i>	<i>Ctrl+Alt+F11</i>	Layihə menecerinin ekranda təsviri
<i>View/Debug Windows</i>		Layihə sazlayıcısının ekranda təsviri
<i>View/Browser</i>	<i>Shift+Ctrl+B</i>	Layihə icmalçısının ekranda təsviri
<i>Tools/Repository</i>		Obyektlərin saxlanıldığı yerin sazlanması
	<i>F12</i>	Forma konstrukturu ilə Kod redaktorunun yerlərinin dəyişdirilməsi
<i>Save Project As...</i>		Layihəni necə yadda saxlamaq
<i>Save All</i>	<i>Shift+Ctrl+S</i>	Bütün layihəni saxlamaq
<i>Close AU</i>		Bütün layihəni bağlamaq

Mühazirə 17: Object Pascal dili

Bu fəsildə biz Object Pascal dilinin əsas elementləri ilə tanış olacağıq. Object Pascal dili Delphi proqramlaşdırma dilidir və standart Pascal dilinin obyekt yönü genişlənməsidir. Kitabın birinci hissəsində Turbo Pascal dilinin əsas hissəsi ətraflı şərhləndirilmişdir. Bu fəsildə Object Pascal dilinin Turbo Pascal dilindən fərqli cəhətlərinə nəzər yetiriləcək, bəzi sahələr daha gücləndiriləcək və əsas diqqət obyekt yönü proqramlaşdırmağa yönəldiləcəkdir. Obyekt yönü proqramlaşdırmanın xüsusiyyətləri, sinif, sahə, xassə, metodlar öyrəniləcək və Object Pascal dilində proqramlaşdırmanın əsas vasitə üsullarına baxacağıq.

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:

*+ - * / . , : ; = > < `*

() { } [] # \$ A @ probel;

Xüsusi simvolların birləşməsindən aşağıdakı simvollar əmələ gəlir:

: = - mənimləmə;

<> - bərabər deyildir;

.. - qiymətlərin dəyişmə diapazonu;

<= - kiçik və ya bərabərdir;

>= - böyük və ya bərabərdir;

(* və *) - { və } fiqurlu mətərizələrin əvəzləyiciləri;

(. və .) - [və] kvadrat mətərizələrin əvəzləyiciləri.

Object Pascal dilinin lüğəti və proqramın strukturu Turbo Pascal dilinin lüğəti və strukturu ilə tamamilə eynidir.

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 sətirin əvvəlində ikiqat sleş (//) işarəsi yazılırsa, onda bütün bir sətir şərh kimi qəbul olunacaqdır. Şərhlər bir neçə sətirdə yazılırsa, onları { və } mətərizələri və ya onların ekvivalenti olan (* və *) simvol birləşmələri daxilində yazmaq lazımdır.

Misal.

// Nyuton üsulu

{ Biz Delphi dilini

öyrənirik. Delphi obyektönlü

proqramlaşdırma

dilidir }

II sum: = sum + x;

{ begin

for k: = 1 to 10 do

a[k]: = x*k;

end; }

(* Laboratoriya işi №1. Dövrü hesablama

proseslərinin proqramlaşdırılması *)

Verilənlərin tipləri

Object Pascal dilində müəyyən edilmiş tiplərin təsnifatı cədvəl 1- də göstərilmişdir.

Cədvəl 1 Object Pascal dilində müəyyən edilmiş tiplər

<i>Grup</i>	<i>Alt qrup</i>
<i>Sadə</i>	<i>Tamədədli</i> <i>Liter (simvollar)</i> <i>Məntiqi (bul)</i> <i>Həqiqi</i>
<i>Struktur</i>	<i>Sətir</i> <i>Massiv</i> <i>Çoxluq</i> <i>Yazı</i> <i>Fayl</i> <i>Sınıf</i>
<i>Göstərici</i>	<i>Tipləşdirilmiş</i> <i>Tipləşdirilməmiş</i>
<i>Variant</i>	
<i>Prosedur</i>	

Verilənlərin sadə tipləri

Bəzi sadə tiplər fiziki (əsas) və ümumi tiplərə bölünür. Fiziki tiplərin təməli dil yaradılıqda 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 programçı özü də yarada bilər, ona görə də belə tiplərə istifadəçi tipləri deyilir. İstifadəçi tiplərinə sadalanan və interval tiplər aiddir.

Tamədədli tiplər

Fiziki tamədədli tiplər və onların əsas göstəriciləri cədvəl 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 2. Tamədədli

tiplər

Tipin adı	Dəyişmə diapazonu	Yaddaşda təsviri
Byte	0-255	işarəsiz 1 bayt
vword	0-65 535	işarəsiz 2 bayt
ShortInt	(-128)-127	işarə ilə 1 bayt
Int64	(-2vv)-(2vv-1)	işarə ilə 8 bayt
SmallInt	(-32 768)-32 767	işarə ilə 2 bayt
LongVWord	0-4 294 967 295	işarəsiz 4 bayt
LongInt	(-2 147 483 648) - 2 147 483 647	işarə ilə 4 bayt

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 simvollar 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.

Həqiqi tiplər

Həqiqi tiplər də fiziki və ümumi olur. Fiziki tiplər cədvəl 2 - 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.

Tipin adı	Dəyişmə diapazonu	Yaddaşda təsviri, bayt
Real48	$2,9 \cdot 10^{-39} - 1,7 \cdot 10^{38}$	6
Single	$1,7 \cdot 10^{-45} - 3,4 \cdot 10^{38}$	4
Double	$5 \cdot 10^{-324} - 1,7 \cdot 10^{308}$	8
Extended	$3,6 \cdot 10^{-4951} - 1,1 \cdot 10^{4932}$	10
Comp	$(-2 \cdot 10^{36} + 1) - (2 \cdot 10^{63} - 1)$	8
Currency	$(-922337203685477,5808) - 922337203685477,5807$	8

Verilənlərin struktur tipləri

Struktur tiplərə aşağıdakılar aiddir:

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

Sınıf tiplər verilənlərin xüsusi tipləridir. Obyektyönlü proqramlaşdırmada sınıf tiplər xüsusi əhəmiyyət kəsb etdiyi üçün, onların izahı fəslin sonunda, ayrı bir bölmədə şərh olınacaqdır. Yerdə qalan tiplərin isə Turbo Pascal dilindən fərqli cəhətlərinə baxaq.

Sətirlər

Sətirlər üç fiziki və bir ümumi tiplə təsvir olunur. Fiziki tiplər: *ShortString* (255 simvol), *AnsiString* ($-2 \cdot 10^{31}$ simvol) *WideString* ($\sim 2 \cdot 10^{30}$ simvol) tiplərindən ibarətdir. Əslində sətir *dəyişənləri* müəyyən mənada, elementləri simvollar olan massivləri ifadə edir.

AnsiString və *WideString* tipləri dinamik massivləri təsvir Birinci tip - **ANSI**, ikinci tip isə **Unicode** kodları ilə kodlaşdırılır.

Sətir tipli verilənlərin ümumi tipi *String* tipidir ki, *ShortString* və *AnsiString* tiplərinə uyğun gəlir. Sətirlər massivlərə uyğun gəldiyindən sətirin istənilən simvoluna massiv elementi kimi müraciət etmək olar. Bunun üçün sətirin adının yanında, kvadrat mötərizə daxilində, simvolun nömrəsini göstərmək lazımdır.

Massivin sıfırıncı 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. Bu tip dəyişən sətirin başlanğıcına **göstəricidir**, **başka sözlə** maşının yaddaşında sətirin birinci simvolunu göstərir.

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 sətirin 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 təsəvvur yaratmaq üçün bir proqram fraqmentinə baxaq,

Misal.

```
var  
s: array [1..1000] of char;  
sl: PChar;  
begin  
sl := @s;  
end;
```

Burada, *s* dəyişəni 1000 simvoldan ibarət massiv, *sl* dəyişəni isə *PChar* tipli elan olunmuşdur və sonra *sl* dəyişəninə *s* qiyməti mənimsədilmişdir. Bununla da *sl* dəyişəni 1000 simvoldan ibarət massivin yerləşdiyi yaddaş oblastını göstərir.

Massivlər

Object Pascal dilində statik və dinamik massivlər mövcuddur.

Statik massivlərin ölçüləri elan etmə bölməsində əvvəlcədən müəyyənləşdirilir. Proqram 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 ikiölçü massiv olur; massivin elementləri *integer* - tam tiplidir. b dəyişəni 59 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, elan etmə 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 [i]:= 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 massiv in 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* - massiv in 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.

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 artırmaq.

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 sətirin 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, onlann 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 elan etmə zamanı müəyyənləşdirilmiş tipli verilənlərə istinad edə bilər. Bu zaman ünvanlaşdırılan verilənlərin tipləri qarşısında A işarəsi qoyulur. Tipləşdirilmiş göstəricilərin ümumi təsvir forması belədir:

Type göstəricinin tipi = A ünvanlaşdırılan verilənin tipi;

Tipləşdirilməmiş göstərici *Pointer* tipli olur və istənilən tip verilənlərə istinad oluna bilər.

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;

sl: string;

begin

s := @ sl;

sl := 'Mən Delphi öyrənirəm';

editl. Text := string(s^);

end;

Bu misalın birinci sətirində s göstəricisinə sl sətirinin göstəricisi mənimsədilmiş, sonra isə həmin sətirin məzmunu dəyişdirilmişdir. Sonuncu sətirdə s ünvanında yerləşən mətn Edit mətn redaktoruna çıxarılmışdır (bu redaktorla növbəti fəsillərdə tanış olacağıq). Bunun üçün aşkar şəkildə göstərməlidir ki, s ünvanında məhz string (s ^) sətiri yerləşir.

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 tamədəbli (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.

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

Object Pascal dilində hesabi ifadələr, məntiqi ifadələr və sətir ifadələr Turbo Pascal dilində olduğu kimidir. Delphi-də çoxlu standart prosedur və funksiyalardan istifadə olunur ki, onlarn ən əsasları kitabın sonundakı Əlavədə göstərilmişdir. Yadda saxlayıd ki, bu Əlavədə verilən bir sıra riyazi prosedur və funksiyalan istifadə etdikdə, modulun Uses bölməsinə **Math** modulunu qoşmaq lazımdır.

Operatorlar

Turbo Pascal dilində proqrama ilkin verilənlər daxiletmə prosedurları ilə daxil, nəticələr isə xaricetmə prosedurları ilə ekranda təsvir olunur və ya ızılırdı. Delphi sistemində isə bu əməliyyatlar xüsusi vizual komponentlərlə yerinə yetirilir.

Birinci hissədə öyrəndiyimiz mənimsətmə operatoru, keçid operatoru, boş operator, strukturlaşdırılmış operatorlar, o cümlədən, tərkibli operator, şərti operator, seçim operatoru, parametrlı, ilkin şərtli və son şərtli dövr operatorları, daxilolma operatoru Turbo Pascal dilində olduğu kimidir.

Mövzu 18: Obyektyönlü proqramlaşdırmanın xüsusiyyətləri

Obyektyönlü proqramlaşdırmanın xüsusiyyətlərini anlamaq üçün proqramlaşdırmanın keçdiyi inkişaf mərhələlərinə qısaca nəzər yetirək. İlk proqramlar bütöv mətnlərdən ibarət idi. Burada proqram əmrlər ardıcılığından 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 ardıcılığını 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 proqram 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 çağırılması 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 bilirdi. Bu isə

proqramlaşdırmanın növbəti inkişaf mərhələsi olan obyektivli proqramlaşdırmanın meydana gəlməsinə səbəb oldu. Burada proqramlar artıq "müstəvi" proqramlar deyildir və proqramçı yalnız prosedur və funksiyalarla kifayətlənmiş, 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. Obyektivli proqramlaşdırmada istifadə olunduqda isə bu hal aradan qaldırılır, belə ki, minimal itkilərlə proqram modifikasiya olunur, genişləndirilir və ya ona əlavələr edilir. Beləliklə obyektivli 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 (eyn menyular, şriftlər, pəncərələr və s.) istifadə olunur. Bu əməliyyatlar blok kimi müxtəlif əlavələrdə istifadə edilmişdir. Yeni proqramlar işlənərkən, zərurat yarandıqda, mövcud proqramlardan hazır bloklar götürülür və onlar yeni tələbatlara uyğunlaşdırılır. Lakin, bütün bu deyilənlərdən belə nəticəyə gəlmək olmaz ki, obyektivli proqramlaşdırma proqramçıları bütün bəlalardan xilas edən "dərmandır", bununla bərabər qabaqcıl texnologiya kimi onun rolu şübhəsizdir. Obyektivli proqramlaşdırmanın ideya və metodlarını öyrənmək o qədər də asan məsələ deyildir, lakin bu texnologiyanın tətbiqi mürəkkəb proqramların işlənməsini əhəmiyyətli dərəcədə sadələşdirməyə imkan verir.

Obyektlər və onların xassələri

Obyekt nədir

Hər şeydən əvvəl, obyekt hər hansı bir konkret şeydir. Biz deyə bilərik ki, o siradan başlayır və harada qurtanır. İkincisi, o hər hansı daxili quruluşa malikdir biz onu bilməyə də bilərik. Üçüncüsü, obyekt müəyyən hərəkətə malik olur 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 oyanmaq üçün onun zəng dəstəyini fırladmaq. 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 çarxm, yaym və s. olmasın bilək. Bizə yalnız onu qurmağı, vaxtı küməyi, zəngi idarə etməyi bilmək, demək olar ki, kifayətdir. Bu yanaşma məhz obyekt yanaşmam bildirir. Ətrafımızda nə qədər obyektlər var ki, biz onlann 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 maraqlandırmır. Məsələn, əgər "divar" obyektə "mismar" obyektini 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 varmışsa, 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 aparıb. Obyektin xassələri obyektin özü ilə sıx bağlıdır. Təbəssümsüz sima olmadığı kimi, xassəsiz obyekt də yoxdur.

Bəs bütün bunların proqramlaşdırmaya nə aidiyyəti vardır? Məsələn 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 obyektidir. Hər bir pəncərə, hər bir nişan, hər bir idarəetmə obyektini, hər bir menyu bir obyektidir. 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 obyektəriə 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 kərpic 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 obyektidir 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 bir 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".

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 proqram 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 quruluşdan 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.

Məsələn, ekranda görünən düymənin ölçüsü, koordinatı, adı və digər xassələri vardır. Biz yalnız özümüzlə lazım olan xassələrlə bilərik. Digər xassələri isə proqramı yazdıqda bir dəfə müəyyənləşdirib, tamamilə unuda bilərik.

Bir proqramda istifadə olunan obyekt digər proqramlarda istifadə etmək zamanı:

- yeni obyektin əlavə edilməsi proqramda mövcud olan digər obyektlərin işini pormur;
- proqramda digər obyektlərin varlığı yeni əlavə edilən obyektin özünü neçə 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 eyni bir prosedurla yazılır, hər bir tank müstəqil obyektidir və proqram hər birini fərdi idarə edir. Bu obyektlərin hər birinin aşağıdakı xassələri mövcuddur:

Xassə

Fərq

Təsvir
Bort nömrəsi
Komandanın soyadı
Texniki vəziyyəti

Döyüş dərəcəsi

Ekranda vəziyyəti

Ataş səsli

Hərəkət zamanı səsli

Hamısı eynidir
Hamısı müxtəlifdir
Hamısı müxtəlifdir
Oyunun başlanğıcında hamısı

eyni, sonra isə fərqlənir

Oyunun başlanğıcında hamısı

eyni, sonra isə müxtəlifdir

Hamısı müxtəlifdir

Hamısı eynidir

Hamısı eynidir

isə təsəvvür edərk ki, proqramçı oyuna tankdan başqa top və zirehli daxil etmək istəyir. Bu o deməkdir ki, proqramçı yeni sinif obyektlər bütün prosedurları yenidən yazmalıdır? Əsla yox: bəzi xassələri dəyişmək kifayətdir. Məsələn belə:

<i>Xassə</i>	<i>Tank</i>	<i>Top</i>	<i>Zirehli maşın</i>
Qiyməti	230	40	50
Sürəti	35	0	55
Zirehli	110	0	40
Atış sürəti	1	2	24
Təsviri	tank.bmp	art.bmp	tr.bmp
Atış səsli		art.vav	tr.vav

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" obyekt emaledicisi - metodu işə düşür. Bu alt proqramda mina öz vəziyyətini dəyişir (yox olur), tank obyekt üzərində isə "partlama" hadisəsi baş verir. Nəticədə "tank" obyekt 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. Qüllədən atış, tankın partladılması və s. kimi hadisələr üçün emal etmə 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 proqramda qayıdaq. Bu proqramda tank bütün əməliyyatları "düşünmədən" icra edir. Tank bilmir ki, onun qarşısında müqavimət var, mina var - o, tamamilə kor və kərdir. Lakin, bütün bunları onun əvəzinə proqramçı bilir və o, izləyir ki, tank mina üzərinə çıxmasın, düşməni 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 saxlamaq 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.

Sınıf və obyekt

Yuxarıda qeyd etdik ki, obyektəyönü proqramlaşdırmada proqramçı çoxlu siniflər üzərində əməliyyat aparır. Sınıf - xassə, metod və hadisələr yığıdır, başqa sözlə, sınıf - metod, xassə və hadisələrdən ibarətdir və bunlar onun mükəmməl işləməsinə 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 sınıf 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 tiptir. 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, Obyekt 1 - Düymə tiplidir. Obyektin yaradılması haqqında "proqramı" öz sözlərimizlə yazaq:

Proqramın başlanğıcı;

Dəyişənlər

Obyektl : Düymə;

Kodun başlanğıcı

Obyektl: = Düymə. Obyekti_ yaratmaq;

Obyektl. Sərlövhə: ='Delphi' ;

Obyektl. 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.

Sınıf əsasında obyekt tipli yeni dəyişənlər yaratmaq mümkündür. Aşağıdakı "proqram" iki yeni düymə yaratmağa imkan verir:

Proqramın başlanğıcı;

Dəyişənlər

Obyektl : Düymə;

Obyekt2 : Düymə;

Kodun başlanğıcı

Obyekt1:=Düymə. Obyekti_ yaratmaq;

Obyekt2: =Düymə. Obyekti_ yaratmaq;

Obyektl.Sərlövhə: = 'Pascal';

Obyekt2. Sərlövhə:='Delphi'

Obyektl.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 obyektədən müxtəlif sərlövhəli iki obyekt

aldıq. Hər iki düymə sərbəst işləyir, biri digərinə mane mimrır, çünki, onlar üçün ayrı yaddaş sahələri ayrılmışdır.

Yaradılmış obyekt Free metodu ilə məhv edilməlidir. Əgər obyekt daha lazım deyildirsə, onda onu pozmaq lazımdır:

Obyektl. 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 yaşderləşdirmək 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.

Mühazirə 19: Mətnlərin təsviri

Mətn (yarlıq) sərlovhəyə malik olmayan idarəedici elementləri işarə etmək üçün tətbiq edilir. Mətnə ən sadə misal olaraq Windows sistemində Pusk (Start) düyməsini basdıqda açılan Əsas menyunun bəndlərini misal göstərmək olar. Hər bir bəndin adı elə mətndir. Bu mətnə yazı, nişan, və ya yarlıq 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 Labell 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*) dialog 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 TAlignment 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 sığmazsa, onu *sətirdən-sətrə* keçirmək olar. Bunun üçün `WordWrap` xassəsinə `True` qiyməti vermək lazımdır.

Yazı şəffaf və ya rəngli ola bilər. Bu Boolean tipli `Trar.szä` xassəsi ilə müəyyənləşdirilir. Yazının rəngi `Color` xassəsi ilə təyin Yazını şəffaf etmək üçün `Transparent` xassəsinə `True` qiyməti lazımdır. Şəffaf yazı adətən şəkil üzərində, məsələn, xəritə üzərində yazıldıqda lazım gəlir ki, məta 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 `FocusControl` tipli `FocusCol` 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:

```
Labell.FocusControl:= Editl;
```

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 müəyyənləşdirilir. 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 yerləşdir: komponentini seçib, `OnClick` hadisəsi qarşısında mausun düyməsini ilk dəfə basaraq bu kodları yazın:

```
procedure TForm1. LabellClick (Sender : TObject);
```

```
begin
```

```
if Editl. CanFocus then Editl. SetFocus;  
end;
```

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 alınacaqdır. Forma üzərində olan bu mətnə nə düzəliş etmək, nə də onun kimi dəyişdirmək mümkündür. Sonralar yazıdan daha məqsəduyğun funksiyalar üçün istifadə edəcəyik.

İnformasiyanın daxil və redaktə edilməsi

İnformasiyanın daxil və redaktə edilməsi formanın xüsusi sahə və oblastlarında yerinə yetirilir. İnformasiyanın daxil edilməsi və zərurət (arandıqda onlara düzəlişlərin edilməsi üçün, Delphi, Edit, MaskEdit, Memo və RichEdit komponentlərini təklif edir. MaskEdit komponenti mətni şablon üzrə daxil etməyə, RichEdit komponenti isə Memo componentinin yerinə yetirdiyi funksiyalara əlavə olaraq, mətni formatlaşdırmağa imkan verən redaktorlardır. Biz Edit və Memo komponentlərini öyrənəcəyik.

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 adını 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 olunan 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 sətirin 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 sətirində 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 yığımindan 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 registrlı mətnlər daxil edin. Button1 düyməsini basdıqda Edit1 sahəsinə daxil edilən mətn kiçik hərflı mətnə, Edit2 sahəsinə daxil edilən mətn isə baş hərflərdən ibarət mətnə çevriləcəkdir.

Birsətirlı 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';
```

Misal 1. Formanın sərlövhəsinin dəyişdirilməsi.
Forma üzərinə Label, Edit və Button komponentləri endirərək onları olduğu kimi yerləşdirin. Obyektlər inspektorunda Label 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;
```

Programı işə salın. Düyməni basıldıqda formanın sərlövhəsi daxil 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 sətiri 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, mətnə sətirin 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;
```

Misal. Vurma əməliyyatı yerinə yetirən kalkulyatorun hazırlanması.

Labell 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;

Labell: TLabel;

procedure Button1Click (Sender: TObject);

Labell: 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 labell.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 daxili etmək olmaz. Edit1 və Edit2 mətn sahələrindən daxil edilən ədədlər sətir tiptən 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 vuruqlan bir mətn sahəsindən daxil etməklə həll edək.

Misal. Vuruqları bir mətn sahəsindən daxil edən kalkulyator. Məsələnin xüsusiyyəti ondan ibarətdir ki, biz yuxandakı proqramda

```

z1:= Edit1.Text;

```

```
z2: = Editl.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)
```

```
    Edit: 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  
  
{SR *.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;  
  
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.

Siyahılar

Siyahı mətn sətirlərindən ibarət qarşılıqlı əlaqəli, nizamlanmış elementlər yığıdır. Windows sistemində siyahılardan geniş istifadə olunur (Font dialog pəncərəsində şriftin adı, tərz, ölçüsü, rəngi və s.). Bu sistem üçün aşağıdakı siyahılar xarakterikdir:

Açılan siyahı pəncərədə bükülmüş sətirdən ibarət olur. Bu sətirdə yerli düymə üzərində mausun düyməsini basdıqda siyahı açılır və bu siyahıdan istənilən bəndi seçmək olar. Siyahı büküldükdə seçilmiş bənd bir sətirdə olunur.

Siyahıdan ibarət açılan sahə - açılan siyahıya oxşayır, lakin, ondan olaraq, siyahıya klaviaturadan yeni qiymət (bənd) əlavə etmək olar. Bu siyahıda kombinasiyalı siyahı da deyirlər. Bu iki idarəetmə elementi Delphi-nin etdiyi **ComboBox** komponenti ilə yaradılır.

Sadə siyahı - ekranda dərhal görünən bir neçə sətirdən ibarət olur. element **ListBox** komponenti ilə yaradılır.

Sadə siyahı

Sadə siyahılarda mətnlərdən ibarət sətirlər düzbucaq sahədə yerləşir. siyahıları yaratmaq üçün Standart səhifəsindəki **ListBox** komponenti istifadə olunur.

Əgər sətirlərin sayı görünmə sahəsində yerləşə biləcəyindən çoxdursa. siyahıda

fırlatma zolağı əmələ gəlir. Fırlatma zolaqları və sütunların Integer tipli Columns xassəsinin qiymətindən asılıdır. Əgər onun qiyməti olarsa, onda sətirlər bir sütunda yerləşəcək və zərurət yaranarsa, şaquli fitf zolağı avtomatik əmələ gələcək və ya itəcəkdir. Əgər Columns xassəsinin qiyməti 7-dən böyük və ya 1-ə bərabər olarsa, onda hökmən üfuqi 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. Üfuqi fırlatma zolağını yaratmaq üçün isə SendMessage metodu ilə siyahıya LERSetHorizontalExtent məlumatı 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). İkinci 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 Larametr burada lazım olmadığı üçün sıfıra bərabər edilmişdir.

Sadə siyahının üslubu TListBoxStyle tipli Style xassəsi ilə müəyyənləşdirilir: Bu xassə aşağıdakı qiymətləri ala bilər:

IbStandart -standart üslub (susmaya görə);

IbOwnerDrawFixed -ItemHeight xassəsi ilə müəyyən olunmuş eyni hündürlüklü elementlərdən ibarət siyahı;

IbOwnerDrawVariable -müxtəlif hündürlüklü elementlərdən ibarət siyahı.

Siyahı haşiyə daxilində də ola bilər. Bu TBorderStyle tipli IBorderStyle xassəsi ilə təyin olunur və bu xassə aşağıdakı qiymətləri ala bilər:

bsNone -haşiyə yoxdur;

bsSingle -haşiyə var (susmaya görə).

Kombinasiyalı siyahı

Kombinasiyalı siyahı redaktə sahəsini və siyahını birləşdirir. İstifadəçi qiyməti siyahıdan seçə və ya redaktə sahəsindən birbaşa daxil edə bilər. Kombinasiyah siyahılan 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 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. Əlifba sırası ilə düzülmüş siyahıya yeni sətir əlavə edildikdə, o ya daxil edildiyi mövqedə qalır, ya da siyahının sonuna əlavə edilir. Bu zaman siyahını ətəfni 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 sətiri seçmək olar. Bir neçə sətiri 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 sətirin seçilməsi üsulunu `ExtendedSelect` xassəsi müəyyənləşdirir. Bu xassəsi verildikdə (məsələn, `ListBox1.ExtendedSelect = True;`) siyahıda cursorla idarəetmə klavişləri (sola, sağa, aşağı və yuxarı), `Shift` və `Ctrl` nşləri ilə seçmək olar. Lakin, unutmayın ki, bu iki xassə yalnız sadə aiddir. `ComboBox` siyahısında eyni zamanda yalnız bir elementi mümkün olduğu üçün, onun `MultiSelect` və `ExtendedSelect` xassəsi voxdur.

Siyahıların `Items` xassəsi

Sadə və kombinasiyalı siyahıların bir sıra oxşar cəhətləri olduğundan onlar çoxlu ümumi xassə, metod və hadisələrə malikdir. Siyahıların ən əsas xassəsi 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 vmmq 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 mahiyyətə 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 (' İncəməlik ');
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 sətiri (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 sətiri **index** parametri ilə göstərilən nömrəli mövqeyə əlavə edir.

MyList siyahısı yaradıldıqdan sonra, **Assign** metodu ilə **ListBoxI** komponentinə mənimsədir.

Assign (source : TPersistent); proseduru - uyğun tipli bir obyektə digər obyektə mənimsədir. Baxılan nümunə **MyList** siyahısını **ListBoxI** komponentinə köçürür. Nəhayət, proqram sonunda **Create** metodu ilə yaradılan siyahının tutduğu yaddaş azad edilir (**Free**).

Siyahının ayrı-ayrı sətirlərinə **Items** massivinin nömrəsi ilə müraciət etmək olar. Sətirlərin nömrəsi sıfırdan başladığı üçün 1-ci sətirə müraciət **Items [0]**, 2-ci sətirə müraciət **Items [1]** və s. kimi yerinə yetirilir. Siyahıda olmayan sətirə müraciət etmək olmaz. Məsələn, siyahı 20 sətirdən ibarətdirsə 27-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 mənimsədir. 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 sətiri seçdikdə proqramçı bu xassəyə özü qiymət verməlidir, məsələn,

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 (CurlIndex, NewIndex : integer); proseduru - CurlIndex nömrəli elementi NewIndex nömrəli mövqeyə yerləşdirir.

IndexOf (const s : string) : integer; proseduru - siyahıda s sətirinin olmasını yoxlayır. Əgər siyahıda belə bir sətir tapılarsa, həmin sətirin nömrəsi, əks halda isə (-1) qiyməti göstərilir.

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.ButtonClick(Sender:TObject);
Var i : LongInt; // Dəyişən istiyari 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;
```

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.

Misal. Sadə siyahının redaktə komponentləri ilə əlaqəsi.

Forma üzərində göstərilən komponentləri yerləşdirin. Burada aşağıdakı komponentlər

təsvir edilmişdir: 1-Memol; 2-ListBox1; 3-Edit1; 4-Label1; 5-9- Button1- Button5;

10 - Samples səhifəsindən SpinEditl.

```
procedure TForm1.EditlExit(Sender: TObject);
Var
s: String;
begin
s:= Editl.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 verilməli və Button düyməsi qoşulmayacaqdır (Button1.Enabled:=False;). Bu onun üçün edilir ki, siyahıya boş sətir daxil edilməsin. Probel klavişini basdıqda siyahıya boş sətirin daxil edilməsinin qarşısını almaq üçün Trim funksiyasından 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 EditlExit 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. Editl komponentini seçib, həmişəki kimi, OnChange hadisəsi qarşısında mausun düyməsini iki dəfə basaraq koda əlavə edin:

```
procedure TForm1.EditlChange(Sender: TObject); begin
Edit1.OnExit(Parent); end;
```

Burada, faktiki olaraq bir hadisədə başqa bir hadisə emaledicisi çağırılmışdır. 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 SpinEditl 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 mətai ListBox1 siyahısına əlavə edir. Bu zaman siyahıya eyni sətirlər daxil edilə bilər. Bunun qarşısını almaq üçün həmin proseduru belə yaratmaq daha məqsəduyğun olar:

```
procedure TForm1.Button1Click(Sender: TObject); Var
s: String; begin
  if ListBox1.Items.Count=0 then
  begin
    ShowMessage('Siyahı boşdur, sətiri 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 düyməsinə 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 Editl 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 sətirin nömrəsi (...IntToStr(i))də siyahıya daxil ediləcəkdir.

Button3 (Del one) düyməsini iki dəfə basaraq yazm:

```
procedure TForm1.Button3Click(Sender: TObject);  
  
Var  
    Current Index: Long Int; // Dəyişənin adı sərbəst seçilmişdir  
begin  
    CurrentIndex:= ListBox1.ItemIndex;  
    // ListBox1.ItemIndex = -1 then Exit;  
    ListBox1.Items.Delete(CurrentIndex)  
end;
```

Del one düyməsini basdıqda bu prosedur siyahıda seçilmiş sətiri 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;
```

Button 5 (Count) düyməsi üçün bu kodu yazm:

```
procedure TForm1.Button5Click(Sender: TObject);  
  
Var  
    m, I: LongInt;  
begin  
  
    m:= Memo1.Lines.Count;  
  
    I:= ListBox1.Items.Count +  
    ShowMessage ('Memo Komponentində 'IntToStr(m)+'  
sətir var '+#13#10+ 'ListBox Komponentində'  
+IntToStr(I)+' sətir var ');
```

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.LabelClick(Sender: TObject);  
begin  
    Memo.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-4ə bir çox hallarda bir siyahıdan müəyyən əlamətlərə görə elementlər seçilərək digər siyahıda yerləşdirilir və ya geri qaytarılır. Bu məsələni proqramlaşdıraraq. Bunun üçün formaya iki ListBox, iki Label və iki Button düymələri yerləşdirin.

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 müəyyənləşdirir (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 latin 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 koddarı da daxil etmək olar bu halda hər bir fənn üçün prosedurdə ListBox1.Items.Add - İnformatika1) ; 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; // Düzüldürmə qadağan
  ListBox2.Sorted:=False; // edilir
  ListBox1.MultiSelect:=True; // Bir neçə elementin
  ListBox2.MultiSelect:=True; // seçilməsinə icazə verilir
  ListBox1.ExtendedSelect:=True; // Klaviaturla id elementin
  ListBox2.ExtendedSelect:=True; // seçilməsinə icazə verilir
  ListBox1.Clear;
ListBox1.DragMode:=dmAutomatic; //Mausla elementlərin
// yerlərinin dəyişdirilməsi
ListBox2.DragMode:=dmAutomatic; //əməliyyatın 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;
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 yoxlanılır.

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.ListBox2.DragOver(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.

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 prosedurlara 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 proqram 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;           // Düzendirmə qadağan
    ListBox2.Sorted:= False;          // edilir
    ListBox1.Multi Select:= True;      // Bir neçə elementin
    ListBox2.Multi Select:= True;      // seçilməsinə icazə verilir
    ListBox1.Extended Select:= True;   // Klaviatıra ilə elementin
    ListBox2.Extended Select:= True;   // seçilməsinə icazə verilir

    ListBox2.Clear;

    ListBox1.DragMode:=dmAutomatic; // Mausla elementlərin
    //yerlərinin dəyişdirilməsi
    ListBox2.DragMode:=dmAutomatic; // əməliyyatın avtomatik
    //başlamasına 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;
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;
end;

```

```

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;

```

```

[procedure TForm1.ListBox2 Drag Drop (Sender,

```

```

    Source: TObject; X, Y:Integer);

```

```

begin

```

```

    With Source as TListBox do

```

```

begin

```

```

    ListBox2.Items.Add (Items[Item Index]);

```

```

    Items.Delete (Item Index);

```

```

end;

```

```

end; ]

```

```
procedure TForm1.ListBoxDragOver(Sender, Source:
```

```
TObject; X, Y: Integer; State: TDragState;
```

```
var Accept: Boolean);
```

```
begin
```

```
if Source = ListBox2 then Accept := True
```

```
else Accept := False;
```

```
end;
```

```
[procedure TForm1.ListBoxDragDrop(Sender,
```

```
Source: TObject; X, Y: Integer);
```

```
begin
```

```
With Source as TListBox do
```

```
Begin
```

```
Listbox.Items.Add (Items [Item Index] );
```

```
Items.Delete (Item Index);
```

```
end;
```

```
end; ]
```

```
procedure TForm1.ListBox2 DragDrop (Sender,
```

```
Source:TObject; X,Y: Integer) ;
```

```
Begin
```

```
//btnRight.Click; və ya
```

```
btnRightClick (Sender);
```

```
end;
```

```
procedure TForm1.ListBoxClick (Sender : TObject;
```

```
begin
```

```
// btnLeft.Click; və ya
```

```
btnLeftClick(Sender) ;
```

```
end;
```

```
end.
```

Programın mətnində elementlərin maus və düymə vasitəsilə qayda ilə yerinə yetirilməsi kodlarının hər iki variantı prosedurlar eyni sərlövhəli, müxtəlif məzmunlu olduğdan zamanda icra etmək mümkün deyildir. Ona görə də bu prosedurlar kursivlə göstərilərək şərh simvolları ({ }) daxilinə salınmışdır.

Düymələrlə iş

Düymələr idarəedici elementlər olaraq müəyyən yetirmək üçün əmrlər vermək məqsədilə istifadə olunur. Ona görə də onları çox vaxt əmrlər düymələri də

adlandırılırlar. Delphi aşağıdakı düymələri təklif edir:

- **Button standart düyməsi;**
- **BitBin şəkilli düyməsi;**
- **SpeedButton cəld müdaxilə düyməsi.**

Bu düymələrin zahiri görünüşü və funksional imkanları çox az fərqlənir.

Standart düymə

Button standart düyməsi pəncərəli idarəetmə elementidir 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 mausu 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, raaasində 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 xətti düzbucaqlı ilə əhatə olunmuş) basılır. Əgər düymə yox, başqa pəncərəli element məsələn, *Edit* və ya *Menyu* 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 **ModalResult** tipli **ModalResult** xassəsindən istifadə etmək olar. Bu hissə aşağıdakı qiymətləri ala bilər: **mrNone**, **mrOk**, **mrCancel**, **mrAbort**, **mrYes**, **mrNo**, **mrAll**, **mrNoToAll**, **mrYesToAll** susmaya görə **mrNone** qiyməti mənimsədir. Əgər bu xassəyə **mrNone** qiymətindən fərqli istənilən qiymət mənimsədilsə, onda **Close** metodu çağırılmadan avtomatik olaraq bağlanacaqdır.

Misal. Mausdan qaçan düymə.

Biz elə proqram yazacağıq ki, mausu düyməyə yönəldikdə o, mausdan qaçacaqdır. Bunun üçün forma üzərində yeganə komponent - Buttonl I yerləşdirərək, onun üçün OnMouseMove hadisəsini yaradaq. Bu məsələnin limiti 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 Button1 MouseMove (Sender: TObject;
```

```
Shift: TShiftState; X, Y: Integer);
```

```
private
```

```
{ Private declarations } public
```

```
{ Public declarations }
```

```
end;
```

```
var
```

```
Form1: TForm1;
```

```
Implementation
```

```
{SR *.dfm}
```

```
procedure TForm1.Button1.MouseMove (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.
```

Şəkilli düymə

Şəkilli düymə Delphi-də TBitBin sinifli BitBin komponenti ilə təsvir olunur. Bu düymə TButton sinifi 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 ayn-ayn təsvirlərdən ibarət ola bilər. Düymənin üzərinə bu təsvirlərdən hansının çıxarılması düymənin aşağıdakı üç vəziyyətdən 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 mənimsənilir;

`bkYes` - düymədə yaşıl rəngli ✓ işarəsi və Yes yazısı var;

`bkNo` - düymədə qırmızı rəngli, üstündən xətt çəkilmiş çevrə (0) 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 təkrar etmə 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 nisbətə təsvirin yerləşməsini `TButtonLayout` tipli `Layout` xassəsi müəyyənləşdirir. Bu xassə aşağıdakı qiymətləri ala bilər:

`blGlyphLeft` -təsvir yazıdan solda (susmaya görə)

`blGlyphRight` -təsvir yazıdan sağda

`blGlyphTop` -təsvir yazıdan yuxarıda

`blGlyphBottom` -təsvir yazıdan aşağıda

Bunlardan başqa, `BitBtn` düyməsi üçün `Margin` və `Spacing` (hər ikisi integer tipli) xassələri var. Bu xassələr uyğun olaraq təsvir və yazılan düymənin kənarlarına görə nizamlamaq və təsvirlə yazı arasındakı məsafəni (piksəllə) müəyyənləşdirmək üçündür. Susmaya görə, hər iki xassənin qiyməti (-1)-ə bərabərdir, yəni təsvir və yazı düymənin mərkəzinə nisbətən simmetrik yerləşmişdir

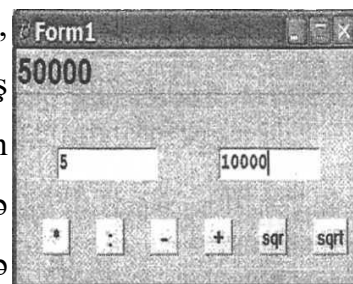
Cəld müdaxilə düyməsi

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 razılaşıdırırlar, 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 xassəsi 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 mənimsədir.



Səkil 1

SpeedButton düyməsinin səthində üç yox, dörd ayrı-ayn təsvir ola bilər. Ona görə də bu düymə üçün NumGlyph xassəsinin maksimal qiyməti 4-9 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 1).

Panell komponentini seçərək onun Aligment xassəsinə taLeftJustify, Align xassəsinə alTop qiyməti verin və sərlövhəsini pozun. Edit1 və Edit2 komponentlərinin Text xassəsinə pozun. Düymələrin sərlövhəsini şəkildəki kimi dəyişin.

Əvvəlki misalı tam təfəsilatı ilə izah etdiyimizdən, burada əlavə izahata ehtiyac görməyərək, məsələnin hazır modulunu Sizə təqdim edirik.

unit Unitl;

interface

uses
Windows, Messages, SysUtils, Classes, Graphics,

Controls, Forms, Dialogs, Buttons, StdCtrls, ExtCtrls;

type

TForm1 = class (Tform)
Panel1: Tpanel;
Edit1: TEdit;
Edit2: TEdit;
SpeedButton1: SpeedButton
SpeedButton2: SpeedButton
SpeedButton3: SpeedButton
SpeedButton4: SpeedButton
SpeedButton5: SpeedButton
SpeedButton6: SpeedButton

procedure Speed Button4 Click (Sender: TObject:
procedure Edit1 KeyPress (Sender: TObject;
var Key: Char);

procedure Speed Button1 Click (Sender: TObject);
procedure Speed Button5 Click (Sender: TObject);
procedure Speed Button3 Click (Sender: TObject);
procedure Speed Button6 Click (Sender: TObject);
procedure Speed Button2 Click (Sender: TObject);

end

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 təkliddə 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 c dəyişdiriciləri Button düyməsinin əmələ gəldiyi TButtonControl sinfindən yaranmışdır.

Müstəqil qeyd olunmuş dəyişdirici

Bu dəyişdirici CheckBox komponenti ilə yaradılır. Dəyişdirici sərlövhdən ibarət düzbucaqlı şəkildədir. 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 verilsə, 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 keçməsinə əsaslı 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-qurguların səs gücləndirici şkalasını göstərmək olar. Bu komponentin xassələrini məsələnin həlli prosesində öyrənsəyik.

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.

Şkala üzərində hərəkət edən (maus və ya idarəetmə klavişləri ilə) məkiyin mövqeyi vuruqlarını 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 xassə `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 xassənin qiymətini dəyişmək lazımdır {999, 9999 və s.).

Position (Mövqe) xassəsi - məkiyin 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 kənarında yerləşməsi üçün bu xassəyə də 2 qiyməti daxil edin.

LineStyle (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 PageDow 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 bölg digər bölgüyə atılacaqdır.

İndi isə GroupBox qrup komponentini seçin. Bu komponent də yenidir. GroupBox komponenti düzbucaqlı haşiyədən və onun sol yuxarı küncündə yerləşən sərlövhədən ibarətdir. Bu komponentin xassəsini Hasil adlandırın. Onun üzərindəki Label3 komponentinin Aligment xassəsinə taLeft Justify qiyməti verin.

Mühazirə 20: Komponentlərlə iş

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 menyü;
PopupMenu - Peyda olan menyü;
Label - Yazı;
Edit - Birsətirli redaktor;
Memo - Çoxsətirli redaktor;
Button - Standart düymə;
CheckBox - Müstəqil dəyişdirici (bayraq);
RadioButton - Dəyişdirici;
ListBox - Siyahı;
ComboBox - Siyahı sahəsi;
ScrollBar - Firlatma 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 - Həndəsi fiqurlar;

Shape - Qrafik təsvir;

Bevel - Faska;

ScrollBar - Fırlatma oblasti;
CheckBox - Dəyişdiricilər siyahısı;
Splitter - Ayrıcı;
StaticText - -Statik mətn;
ControlBar - Alətlər paneli üçün konteyner;
ApplicationEvents - Əlavə hadisəsi;
Chart – Diaqram.

PageControl -Bloknot;
ImageList –Qrafik təsvirlər siyahısı
RichEdit – tam funksiyalı mətn redaktoru;
TabControl- Əlfəcin;
TrackBar- Şkala (məkik);
ProgressBar - İşin gedisi indikatoru;
UpDown - Sayğac;
HotKey - Cəld klavişlər kombinasiyası redaktoru;
Animate - Videokliplərə baxış;
DateTimePicker - Tarixi daxiletmə sətri;
MonthCalendar - Təqvim;
TreeView - Obyektlər ağacı;
ListView - Siyahı;
HeaderControl1- Ayrıcı;
StatusBar -Vəziyyətlər sətri;
ToolBar - Alətlər paneli;
CoolBar -"Sərt" alətlər paneli;
PageScroller- Təsvirlər fırladıcısı

Pəncərəli idarəetmə elementləri müəyyən təyinatlı xüsusişdirilmiş pəncərədən ibarətdir. Belə elementlərə əmrlər düymələri, mətn sahələri, fırlatma zolaqları və s. aiddir. Pəncərəli elementlər üçün TWinControl sinfi baza sinifidir 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 xətlə 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.

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 kombinasiyasının müəyyənləşdirmək üçün uyğun simvolun qarşısında & işarəsi yazmaq lazımdır, məsələn, Buttonl düyməsinin Caption xassəsi qarşısında &Close

yazıb, proqrama Forml .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 sətiri yazın:

Forml.Caption:=' Book' ;

Yenidən Obyektər inspektoruna qayıdın. OnDbClick hadisəsini tapıb. Onun qarşısında mausun düyməsini iki dəfə basın (OnDbClick hadisəsi mausun düyməsini iki dəfə basmaq hadisəsidir) və koda aşağıdakı sətiri əlavə edin:

Forml.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. Bn prosedurların adlarına, onların komponent və hadisə ilə əlaqəsinə, type bölməsində yazılışına diqqət yetirin və implementation bölməsindəki strukturunu dərinlən 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:

Panel.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 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 dialog pəncərəsindən təklif olunan konkret rənglər seçmək olar.

Cədvəl 1. Əsas rəng sabitləri

Sabit	Rəng	Qiyməti
clAqua	Açıq mavi	\$FFFFFF00
clBlack	Qara	\$000000
clBlue	Mavi	\$FF0000

clFuchsia	<i>Yasəməni</i>	\$FF00FF
clGray	<i>Boz</i>	\$808080
clGreen	<i>Yaşıl</i>	\$008000
clLime	<i>Açıqyaşıl</i>	\$00FF00
clMaroon	<i>Tünd qırmızı</i>	\$000080
clNavy	<i>Tünd göy</i>	\$800000
clOlive	<i>Zeytımı</i>	\$008080
clPurple	<i>Bənövşəyi</i>	\$800080
clRed	<i>Qırmızı</i>	\$0000FF
clSilver	<i>Gümüşü</i>	\$C0C0C0
clTeal	<i>Göy-yaşıl</i>	\$808000
clWhite	<i>Ag</i>	\$FFFFFF
clYellow	<i>San</i>	\$00FFFF

Misal. Komponentin rənginin dəyişdirilməsi.

Forma üzərinə bir Panel paneli və bir Edit redaktoru yerləşdirin. Panell komponentini seçərək Align xassəsinə alTop qiyməti verin. Sonra Color xassəsinə clGreen qiyməti seçin. Edit redaktorunu seçin, onun Color xassəsinə clRed qiyməti verin. Nəhayət, Forml 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:

Panel.Color:=clGreen;

Editl.Color:=clRed;

Forml.Color:=clAqua;

Sabitlərin bu kitabda göstərilməyən digər hissəsindən standart Windows pəncərəsinin Svoystva:Gkran/oformlenie 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 seçilsə, 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 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 saati şəklindədir.

DragCursor xassəsi. TCursor tipli DragCursor xassəsi komp 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 edilir. 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.

Pont 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.

Labell.Font.Name:='Arial' ;

Integer tipli **Size** xassəsi şriftin punktlarla (*1 punkt=1/72 düym*) ölçüsünü müəyyənləşdirir.

Misal.

Labell.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.

Labell.Font.Height:= -11;

Label2.Font.Size:= -Labell.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.

Labell.Font.Color:= clMaroon;

Labell.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 səhi* (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 sərlövhasində mətn yazırdıq. İndi isə formanın üzərində mətn yazacaq. Bunun üçün forma üzərində Button standart düyməsi yerləşdirib, onun üzərində mausun düyməsini iki dəfə basaraq yunitdə kursorla göstərilən mövqeyə aşağıdakı kodları yazın:

```
Forml.Color:= clAqua;  
// Şriftin parametrlərinin təyini  
Forml.Canvas.Font.Name:= 'Courier';  
Forml.Canvas.Font.Style:= [fsBold]+[fsUnderline];  
Forml.Canvas.Font.Size:=28;  
Forml.Canvas.Font.Color:=clRed;  
// Forma səthinə mətnin çıxarılması  
Forml.Canvas.Text Out (50, 50, 'Salam, Delphi');
```

Bu misalda, *F9* klavişini basdıqdan sonra, Button düyməsini basdıqda formanın səthi mavi rəngli olmaqla onun üzərinə qırmızı rəngli, 28 punfa ölçülü, yanımqalın və altdan xətt çəkilmiş şriftlə Salam, Delphi məoi çıxarılır. Forma səthinə mətn çıxarmaq üçün

```
procedure TextOut(x,y : integer; const Text: string);
```

proseduru tətbiq edilmişdir. Burada, *x*, *y* mətnin sol yuxarı kuncünün koordinatlarını, *Text* isə səthə çıxarılacaq mətn sətirini müəyyənləşdirir.

Height və *Width* xassələri. Integer tipli *Height* və xassələri uyğun olaraq komponentin piksellə *hündürlüyünü* və enini müəyyənləşdirir

Misal.

```
GroupBox1.Height:=Forml.ClientHeight;  
GroupBox1.Width:= Forml.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 ləqəmlərlə 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 həcmi*ni bildirməklə, komponentin onun *görünüşünü qəbul edib-etmədiyini* müəyyənləşdirir. *True* və ya *False* qiyməti alır.

ParentFont xassəsi. Boolean tipli **ParentFont** xassəsi əcdad-obyektin şrifflərini bildirməklə, komponentin onun *şrifflərini 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 ardıcılığını* müəyyənləşdirir. Fokusalma komponentə idarəetmənin verilməsini müəyyən edir. Komponentlərin fokusalma ardıcılığı onların forma üzərində yerləşmə ardıcılığına 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 ardıcılığının başqa konteynerdə yerləşən komponentlərin fokusalma ardıcılığına heç bir aidiyyəti yoxdur. Fokusalma ardıcılığını dəyişdirmək üçün **TabOrder** xassəsinə müvafiq qiymətlər vermək lazımdır. **TabOrder** xassəsi sıfıra 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 ardıcılığını Delphi interfeysində də dəyişmək olar. Bunun üçün **EditTabOrder** (*Tabulyasiya ardıcılığına düzəliş*) dialoq pəncərəsini ekrana çıxarmaq lazımdır. Bu pəncərə **Edit** (*Düzəliş*) menyusundan **EditTabOrder** əmrini icra etməklə ekrana çağırır.

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 mətndə 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:

```
Editl.ReadOnly:=True;
```

```
Editl.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 bilər, lakin, mətnə düzəlişlər edilmir:

Delphi-də komponentlərin çoxlu xassələri mövcuddur. Biz komponentlər üçün daha ümumi və xarakterik olan əsas xassələrə Konkret komponentlərin xüsusiyyətlərini öyrəndikdə yeni xassələrlə tanış olacaq, onları və burada öyrəndiyimiz xassələri tətbiq etməklə məsələlər həll edəcəyik.

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 daxilətməfokusunu 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) ;
  // Busətrioproqramçıyazır
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 analogi) tipli, yəni sətir tipli olduğu üçün, cari vaxt sətir tipinə çevrilməlidir. Belə çevirməni isə *TimeToStr* funksiyası ("*vaxtı sətərə 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 **OnDbClick** hadisəsi baş verir. Bu zaman hadisələr aşağıdakı ardıcılıqla baş verir: **OnMouseDown**, **OnClick**, **OnMouseUp**, **OnDbClick**, **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, ss Ctrl] 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 İbdisəsi 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 sətiri yazırsınız. F9 klavişini basın. Əgər proqramda heç bir səhv olmazsa, onda mausun göstəricisini forma üzərində gəzdirdikdə Panell 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 sistemində aiddir. Bundan başqa, əgər forma üzərində digər komponentlər olarsa, mausun göstəricisini həmin komponentlərin üzərində yerləşdirdikdə göstəricinin koordinatları yenə də təsvir edilməyəcəkdir.

İndi isə yazdığımız sətiri izah edək. Sərlövhədə "soldan:" və "yuxarıdan:" sözləri, onların qarşılarında isə x və y dəyişənlərinin qiymətləri təsvir ediləcəkdir. Prosedurdan görüldüyü kimi, x və y dəyişənləri tam tiplidir, Caption xassəsi isə sətir tiplidir. Ona görə də x və y dəyişənlərinin qiymətlərini sətərə çevirmək lazımdır ki, bu məqsədlə IntToStr ("*tamı sətərə 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:=
```

```
Panel.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 mausa» digər (sağ və orta) düymələri basıldıqda forma təmizlənir (refresh metodB 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 ardıcılıqla baş verir: *OnKeyDown*, *OnKeyPress* *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 parametərə sıfır qiyməti (#0) verilərsə, bu o deməkdir ki, klavişin basılması ləğv edilir.

OnKeyPress hadisəsi idarəedicisi klavişlərə məhəl qoymur. **Caps Lock** və *Shift* klavişləri ilə müəyyən edilən registrlərin vəziyyətini isə nəzərə alır. *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  
  Panell.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 çıxarıyıq ki, paneldə - basılan klavişin simvolunu, formada isə ədədi qiymətlərini (kodlarını) görə bilərsiniz. İndi yuxanda 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+I* 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
end

```

İstənilən simvolu basdıqda o, sərlövhədə təsvir olunacaq, "!" simvolunu basdıqda isə Key parametri sifra bərabər edilir, sanki klaviş basılmamışdır və sərlövhədə Bu simvolu daxil etmək olmaz ! mətni təsvir olunacaqdır.

Pəncərəli idarəetmə elementi fokus aldıqda TNotifyEvent tipi *OnEnter* hadisəsi baş verir. Bu hadisə element üzərində mausun düyməsini və ya *Tab* klavişini basdıqda yaranır. Element fokusu itirdikdə isə *OnExit* hadisəsi baş verir.

Komponentlərin *drag-and-drop* ("*dartmaq və yerləşdirmək*") metodu ilə yerlərini dəyişdikdə iki element istifadə edilir: mənbə və qəbuledici. *Mənbə*-hərəkət etdirilən obyektədən, *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 təsdiqəldimə 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 = Labell 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  
    Labell.Left = x; Labell.Top: = y;  
end;
```

F9 klavişini basdıqdan sonra, mausla *Labell* komponentinin yerini dəyişdirə bilərsiniz.

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 fokus* verir. Əgər idarəetmə elementi həmin vaxt fokus ala bilmirsə, səhv baş verir. Ona görə də fokus verməzdən əvvəl, həmişə 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 Editl.CanFocus then Editl.SetFocus;
```

Burada, *Editl* birsətirli mətn redaktoruna fokus verməzdən (*SetFocus*) əvvəl onun fokus ala biləcəyi (*CanFocus*) *if* operatoru ilə yoxlamır.

Clear metodu. **Clear** metodu komponentin mətndən ibarət məzmununu *pozmaq* üçündür.

Misal. *Edit* mətn sahəsinin təmizlənməsi.

```
Editl.Clear;
```

Ref resh metodu. **Ref resh** metodu idarəetmə elementini *yeniləşdirmək* üçündür. Yeniləşmə elementin təsviri və onun konturlarını 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.
```

Mühazirə 21: Mətnlərin təsviri

Mətn (yarlıq) sərlövhəyə malik olmayan idarəedici elementləri işarə etmək üçün tətbiq edilir. Mətnə ən sadə misal olaraq Windows sistemində *Pusk (Start)* düyməsini basdıqda açılan *Əsas menyunun* bəndlərini misal göstərmək olar. Hər bir bəndin adı elə mətndir. Bu mətnə yazı, nişan, və ya yarlıq 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 *Labell* 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)* dialog 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.

Yazı şəffaf və ya rəngli ola bilər. Bu Boolean tipli *Trar.szä* xassəsi ilə müəyyənləşdirilir. Yazının rəngi *Color* xassəsi ilə təyin. Yazını şəffaf etmək üçün *Transparent* xassəsinə *True* qiyməti ı lazımdır. Şəffaf yazı adətən şəkil üzərində, məsələn, xəritə üzərində yazıldıqda lazım gəlir ki, məta 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 *FocusControl* tipli *FocusCol* 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:

```
Labell.FocusControl:= Editl;
```

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 müəyyənləşdirilir. 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 yerləşdir: komponentini seçib, OnClick hadisəsi qarşısında mausun düyməsini ilk dəfə basaraq bu kodları yazın:

```
procedure TForm1. LabelClick (Sender : TObject);  
  
begin  
if Edit1. CanFocus then Edit1. SetFocus;  
end;
```

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 alınacaqdır. Forma üzərində olan bu mətnə nə düzəliş etmək, nə də onun kimi dəyişdirmək mümkündür. Sonralar yazıdan daha məqsədəuyğun funksiyalar üçün istifadə edəcəyik.

İnformasiyanın daxil və redaktə edilməsi

İnformasiyanın daxil və redaktə edilməsi formanın xüsusi sahə və oblastlarında yerinə yetirilir. İnformasiyanın daxil edilməsi və zərurət (arandıqda onlara düzəlişlərin edilməsi üçün, Delphi, Edit, MaskEdit, Memo və RichEdit komponentlərini təklif edir. MaskEdit komponenti mətni şablon üzrə daxil etməyə, RichEdit komponenti isə Memo componentinin yerinə yetirdiyi funksiyalara əlavə olaraq, mətni formatlaşdırmağa imkan verən redaktorlardır. Biz Edit və Memo komponentlərini öyrənəcəyik.

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 adını 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 olunan 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 sətirin 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 sətirində 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;

`ecNo rma 1 -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 hərflili 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';
```

Misal I. Formanın sərlövhəsinin dəyişdirilməsi.

Forma üzərinə `Label`, `Edit` və `Button` komponentləri endirərək onları 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. Buttonl düyməsinin OnClick hadisəsi üçün aşağıdakı kodları yazın:

```
procedure TForm1.Button1Click(Sender: TObject);  
begin  
    Form1.Caption:= Edit1.Text;  
end;
```

Programı işə salın. Düyməni basıldıqda formanın sərlövhəsi daxil 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 sətiri belə də yazmaq olar:

```
if (Key < '0') or (Key > '9') then
```

Programı işə buraxın. Bu programın 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, mətdə sətirin sonu işarəsi (#13 kodu) olmur və ona görə də bu komponent Enter klavişinə məhəl qoymur. Edit komponentinin Enter klavişinə reaksiya verməsi üçün kodları programçı 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;
```

Misal. Vurma əməliyyatı yerinə yetirən kalkulyatorun hazırlanması.

Labell komponentinin sərlövhəsini pozun, Button1 düyməsinin sərlövhəsini Vurma adlandırın, Edit komponentlərinin isə Text xassələrini pozun. Button1 düyməsi üçün OnClick hadisə emaledicisi yaradın. Bu məsələnin programının 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;
Labell: TLabel;
procedure Button1Click (Sender: TObject);
Labell: 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 labell.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 daxili etmək olmaz. Edit1 və Edit2 mətn sahələrindən daxil edilən ədədlər sətir tiptən 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 vuruqlan 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:= Edit2.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;
  z: LongInt; // Qlobaldəyişən

implementation
{$SR *.DFM}
// Klaviaturanın Enter klavişinə reaksiyası
procedure TForm1.Edit1KeyPress(Sender: TObject);
    var Key: Char;
begin
  if Key=#13 then
  begin
    Key:=#0;
    Edit1.SetFocus;
    z:= 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;
  end;
  Style:= [fsBold];
end;
  Label1.Caption:=s;
end;

end.

```

Burada, OnKeyPress hadisə emaledicisində, Edit1 komponentinə daxil etmə 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.

Siyahılar

Siyahı mətn sətirlərindən ibarət qarşılıqlı əlaqəli, nizamlanmış elementlər yığıdır. Windows sistemində siyahılardan geniş istifadə olunur (Font dialog pəncərəsində şriftin adı, tərz, ölçüsü, rəngi və s.). Bu sistem üçün aşağıdakı siyahılar xarakterikdir:

Açılan siyahı pəncərədə bükülmüş sətirdən ibarət olur. Bu sətirdə yerli düymə

üzərində mausun düyməsini basdıqda siyahı açılır və bu siyahıdan istənilən bəndi seçmək olar. Siyahı büküldükdə seçilmiş bənd bir sətirdə olunur.

Siyahıdan ibarət açılan sahə - açılan siyahıya oxşayır, lakin, ondan olaraq, siyahıya klaviaturadan yeni qiymət (bənd) əlavə etmək olar. Bu siyahıda kombinasiyalı siyahı da deyirlər. Bu iki idarəetmə elementi Delphi-nin etdiyi **ComboBox** komponenti ilə yaradılır.

Sadə siyahı - ekranda dərhal görünən bir neçə sətirdən ibarət olur. element **ListBox** komponenti ilə yaradılır.

Sadə siyahı

Sadə siyahılarda mətnlərdən ibarət sətirlər düzbucaq sahədə yerləşir. siyahıları yaratmaq üçün Standart səhifəsindəki **ListBox** kompon istifadə olunur.

Əgər sətirlərin sayı görünmə sahəsində yerləşə biləcəyindən çoxdursa. siyahıda fırlatma zolağı əmələ gəlir. Fırlatma zolaqları və sütunların Integer tipli **Columns** xassəsinin qiymətindən asılıdır. Əgər onun qiyməti olarsa, onda sətirlər bir sütunda yerləşəcək və zərurət yaranarsa, şaquli fitf zolağı avtomatik əmələ gələcək və ya itəcəkdir. Əgər **Columns** xassəsinin qiyməti 7-dən böyük və ya 1-ə bərabər olarsa, onda hökmən üfuqi 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. Üfuqi fırlatma zolağını yaratmaq üçün isə **SendMessage** metodu ilə siyahıya **LESetHorizontalExtent** məlumatı 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. Üfuqi fırlatma zolağı isə **SendMessage** metodu ilə yaradılır. Bu metodda birinci parametr **ListBox1** komponenti ilə əlaqə yaradır (**Handle**). İkinci parametr üfuqi

fırlatma zolağını yaradır, üçüncü parametrlə üfqı fırlatma zolağının həmişə təsvir edilməsi müəyyənləşdirilir: əgər bu parametr siyahının ölçüsündən böyük olarsa, üfqı fırlatma zolağı həmişə görünəcəkdir. Dördüncü Larametr burada lazım olmadığı üçün sıfıra bərabər edilmişdir.

Sadə siyahının üslubu TListBoxStyle tipli Style xassəsi ilə müəyyənləşdirilir: Bu xassə aşağıdakı qiymətləri ala bilər:

IbStandart -standart üslub (susmaya görə);

IbOwnerDrawFixed -ItemHeight xassəsi ilə müəyyən olunmuş eyni hündürlüklü elementlərdən ibarət siyahı;

IbOwnerDrawVariable -müxtəlif hündürlüklü elementlərdən ibarət siyahı.

Siyahı haşiyə daxilində də ola bilər. Bu TBorderStyle tipli IBorderStyle xassəsi ilə təyin olunur və bu xassə aşağıdakı qiymətləri ala bilər:

bsNone -haşiyə yoxdur;

bsSingle -haşiyə var (susmaya görə).

Kombinasiyalı siyahı

Kombinasiyalı siyahı redaktə sahəsini və siyahını birləşdirir. İstifadəçi qiyməti siyahıdan seçə və ya redaktə sahəsindən birbaşa daxil edə bilər. Kombinasiyah siyahıyanı 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 üfqı fırlatma zolağı olmur. Kombinasiyalı siyahının xarici görünüşünü və onun 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 daxil etmə 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 qrafikçəkme 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. Əlifba sırası ilə düzülmüş siyahıya yeni sətir əlavə edildikdə, o ya daxil edildiyi mövqedə qalır, ya da siyahının sonuna əlavə edilir. Bu zaman siyahını ətofnı 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 sətiri seçmək olar. Bir neçə sətiri 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 sətirin seçilməsi üsulunu **ExtendedSelect** xassəsi müəyyənləşdirir. Bu xassəsi verildikdə (məsələn, **ListBox1.ExtendedSelect := True;**) siyahıda kursorla idarəetmə klavişləri (sola, sağa, aşağı və yuxarı), **Shift** və **Ctrl** nşləri ilə seçmək olar. Lakin, unutmayın ki, bu iki xassə yalnız sadə aiddir. **ComboBox** siyahısında eyni zamanda yalnız bir elementi mümkün olduğu üçün, onun **MultiSelect** və **ExtendedSelect** xassəsi voxdur.

Siyahıların **Items** xassəsi

Sadə və kombinasiyalı siyahıların bir sıra oxşar cəhətləri olduğundan onlar çoxlu ümumi xassə, metod və hadisələrə malikdir. Siyahıların ən əsas xassəsi 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 vmmq 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 mahiyyətə 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: TStringList; // MyList adı sərbəst seçilmişdir  
  
  begin  
  
    MyList:= TStringList.Create;  
  
    try  
  
      With MyList do begin Add (' Riyaziyyat ');  
  
        Add (' İncəsənət ');  
  
        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 sətiri (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 sətiri index parametri ilə göstərilən nömrəli mövqeyə əlavə edir.

MyList siyahısı yaradıldıqdan sonra, Assign metodu ilə ListBox1 komponentinə mənimsədir.

Assign (source : TPersistent); proseduru - uyğun tipli bir obyekt digər obyektə mənimsədir. Baxılan nümunə MyList siyahısını ListBox1 komponentinə köçürür. Nəhayət, program sonunda Create metodu ilə yaradılan siyahının tutduğu yaddaş azad edilir (Free).

Siyahının ayrı-ayrı sətirlərinə Items massivinin nömrəsi ilə müraciət etmək olar. Sətirlərin nömrəsi sıfırdan başladığı üçün 1-ci sətirə müraciət Items [0], 2-ci sətirə müraciət Items [1] və s. kimi yerinə yetirilir. Siyahıda olmayan sətirə müraciət etmək olmaz. Məsələn, siyahı 20 sətirdən ibarətdirsə 27-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. Program yolu ilə sətiri seçdikdə programçı bu xassəyə özü qiymət verməlidir, məsələn,

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 : TString) : 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 sətirinin olmasını yoxlayır. Əgər siyahıda belə bir sətir tapılarsa, həmin sətirin nömrəsi, əks halda isə (-1) qiyməti göstərilir.

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 TForm.ButtonClick(Sender:TObject);
Var i : LongInt; // Dəyşən istiyari 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;
```

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.

Misal. Sadə siyahının redaktə komponentləri ilə əlaqəsi.

Forma üzərində göstərilən komponentləri yerləşdirin. Burada aşağıdakı komponentlər təsvir edilmişdir: 1-Memol; 2-ListBox1; 3-Edit1; 4-Label1; 5- Button1- Button5; 10 - Samples səhifəsindən SpinEdit1.

```
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 sətirin daxil edilməsinin qarşısını almaq üçün Trim funksiyasından 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 mətai 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 düyməsinə 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 sətirin nömrəsi (...IntToStr(i))də siyahıya daxil ediləcəkdir.

Button3 (Del one) düyməsini iki dəfə basaraq yazm:

```

procedure TForm1.Button3Click(Sender: TObject);
var
  Current Index: Long Int; // Dəyişənin adı sərbəst seçilməlidir
begin
  CurrentIndex:= ListBox1.ItemIndex;
  // ListBox1.ItemIndex = -1 then Exit;
  ListBox1.Items.Delete(CurrentIndex)
end;

```

Del one düyməsini basdıqda bu prosedur siyahıda seçilmiş sətiri 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;
```

Button 5 (Count) düyməsi üçün bu kodu yazm:

```
procedure TForm1.Button5Click(Sender: TObject);
Var
  m, l: LongInt;
begin
  m:= Memo1.Lines.Count;

  l:= ListBox1.Items.Count; +
  ShowMessage ('Memo Komponentində 'IntToStr(m)'+
  sətir var +'#13#10'+ ListBox Komponentində'
  +IntToStr(l)+' sətir var ');
```

Kodun təsvirindən onun yerinə yetirdiyi funksiya aydın olduğu üçün əlavə izahə 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-4ə bir çox hallarda bir siyahıdan müəyyən əlamətlərə görə elementlər seçilərək digər siyahıda yerləşdirilir və ya geri qaytarılır. Bu məsələni proqramlaşdıraraq. Bunun üçün formaya iki ListBox, iki Label və iki Button düymələri yerləşdirin.

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 şəkllə uyğun müəyyənləşdirir (Form -

Fənn və imtahanlar, Button1-Sağa, Button2-Sclə 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 sərlövəsində 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 koddarı da daxil etmək olar bu halda hər bir fənn üçün prosedurda ListBox1.Items.Add - İnformatika1) ; 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; // Düzəldirmə qadağan
  ListBox2.Sorted:=False; // edilir
  ListBox1.MultiSelect:=True; // Bir neçə elementin
  ListBox2.MultiSelect:=True; // seçilməsinə icazə verilir
  ListBox1.ExtendedSelect:=True; // Klaviatıra ilə elementin
  ListBox2.ExtendedSelect:=True; // seçilməsinə icazə verilir
  ListBox2.Clear;
  ListBox1.DragMode :=dmAutomatic; //Maussa elementlərin
  // yerlərinin dəyişdirilməsi
  ListBox2.DragMode :=dmAutomatic; //əməliyyatın 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;
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,

dövrlerin sayı dəyişməyəcəkdir. Bu isə səhvə gətirəcəkdir. Elementin seçilməsi Selected xassəsi ilə yoxlanır.

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.ListBox2.DragOver(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.

Bu prosedur icra olunduqda həmişə sonuncu seçilmiş elemem 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 prosedurlana 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.ListBox2.DragDrop(Sender,
  Source:TObject; X, Y: Integer);
begin
  btnRight.Click; // O biri düymə üçün btnLeft.Click;
end;
```

və ya

```
procedure TForm1.ListBox2.DragDrop(Sender, Source:TObject; X, Y: Integer);
begin
  btnRight.Click(Sender);
end;
```

Beləliklə, siyahılar arasında element mübadiləsini icra edən proqram tam mətni aşağıdakı kimi olacaqdır:

```

unit Unit;

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 ListBoxDragOver (Sender, Source: TObject;
      X,Y: Integer; State: TDragState; var Accept: Boolean);
    procedure ListBoxDragDrop(Sender,Source: TObject;
      X, Y:Integer);
    procedure ListBoxClick (Sender :TObject);
    //procedure ListBoxDragDrop (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; // Dürbündürme qadağan
  ListBox2.Sorted:= False; // edilir
  ListBox1.Multi Select:= True; // Bir neçe elementin
  ListBox2.Multi Select:=True; // seçilməsinə icazə verilir
  ListBox1.Extended Select:= True; // Klaviatura ilə elementin
  ListBox2.Extended Select:= True; // seçilməsinə icazə verilir

  ListBox2.Clear;

  ListBox1.DragMode:=dmAutomatic; // Mausla elementlərin
  //yerlərinin dəyişdirilməsi
  ListBox2.DragMode:=dmAutomatic; // əməliyyatın 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;

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;

[procedure TForm1.ListBox2 Drag Drop (Sender,

        Source: TObject; X, Y:Integer);

begin

    With Source as TListBox do

begin

        ListBox2.Items.Add (Items[Index]);

        Items.Delete (Item Index);

    end;

end; ]

procedure TForm1.ListBox Drag Over (Sender, Source:

        TObject; X,Y : Integer; State: TDragState;

        var Accept: Boolean);

begin

    if Source= ListBox2 then Accept: = True

    else Accept: = False;

end;

[procedure TForm1.ListBoxDragDrop (Sender,

        Source: TObject; X, Y: Integer);

begin

    With Source as TListBox do

Begin

        ListBox.Items.Add (Items [Item Index]) ;

        Items.Delete (Item Index);

    end;

end; ]

procedure TForm1.ListBox2 DragDrop (Sender,

```



```

Source:TObject; XY: Integer);
begin
    //btnRight.Click; və ya
    btnRightClick (Sender);
end;

procedure TForm1.ListBoxClick (Sender : TObject;
begin
    //btnLeft.Click; və ya
    btnLeftClick(Sender);
end;
end.

```

Programın mətnində elementlərin maus və düymə vasitəsilə qayda ilə yerinə yetirilməsi kodlarının hər iki variantı prosedurlar eyni sərlövhəli, müxtəlif məzmunlu olduğdan zamanda icra etmək mümkün deyildir. Ona görə də bu prosedurlar kursivlə göstərilərək şərh simvolları ({ }) daxilinə salınmışdır.

Düymələrlə iş

Düymələr idarəedici elementlər olaraq müəyyən 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;
- BitBin şəkilli düyməsi;
- SpeedButton cəld müdaxilə düyməsi.

Bu düymələrin zahiri görünüşü və funksional imkanları çox az fərqlənir.

Standart düymə

Button standart düyməsi pəncərəli idarəetmə elementidir 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 mausu 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, raaasində 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 xətti düzbucaqlı ilə əhatə olunmuş) basılır. Əgər düymə yox, başqa pəncərəli element məsələn, **Edit** və ya **Menyu** 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 **ModalResult** tipli **ModalResult** xassəsindən istifadə etmək olar. Bu hissə aşağıdakı qiymətləri ala bilər: **mrNone**, **mrOk**, **mrCancel**, **mrAbort**, **mrYes**, **mrNo**, **mrAll**, **mrNoToAll**, **mrYesToAll** susmaya görə **mrNone** qiyməti mənimsədir. Əgər bu xassəyə **mrNone** qiymətindən fərqli istənilən qiymət mənimsədilsə, onda **Close** metodu çağrılmadan avtomatik olaraq bağlanacaqdır.

Misal. Mausdan qaçan düymə.

Biz elə proqram yazacağıq ki, mausu düyməyə yönəldikdə o, mausdan qaçacaqdır. Bunun üçün forma üzərində yeganə komponent - **Button1** yerləşdirərək, onun üçün **OnMouseMove** hadisəsini yaradaq. Bu məsələnin limiti 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 Button1 MouseMove (Sender: TObject;
```

```
Shift: TShiftState; X, Y: Integer);
```

```
private
```

```

{ Private declarations } public
{ Public declarations }
end;

```

```

var
  Form1: TForm1;

Implementation
{$R *.dfm}
procedure TForm1.ButtonMouse Move (Sender: TObject;
  Shift: TShiftState; X,Y: Integer);
var
  index: integer;
begin
  index := random(4);
  case index of
    0: Button.Left := Button.Left + Button.Width;
    1: Button.Left := Button.Left - Button.Width;
    2: Button.Top := Button.Top + Button.Height;
    3: Button.Top := Button.Top - Button.Height;
  end;
  if Button.Left < 0 then Button.Left := 0;
  if Button.Left + Button.Width > Form1.Width then
    Button.Left := Form1.Width - Button.Width;
  if Button.Top < 0 then Button.Top := 0;
  if Button.Top + Button.Height > Form1.Height then
    Button.Top := Form1.Height - Button.Height;
  end;
end.

```

Şəkilli düymə

Şəkilli düymə Delphi-də TBitBin sinifli BitBin komponenti ilə təsvir olunur. Bu düymə TButton sinifi 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 ayn-ayn 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 mənimsədir;

`bkYes` - düymədə yaşıl rəngli ✓ işarəsi və `Yes` yazısı var;

`bkNo` - düymədə qırmızı rəngli, üstündən xətt çəkilmiş çevrə (0) 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 təkrar etmə 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` xassəsinə dəyişmək məsləhət görülmür. Çünki, bu halda düymə onun üçün nəzərdə tutulmuş funksiyanı yerinə yetirməyəcəkdir. Düymənin səthində yazıya nisbətə təsvirin yerləşməsinə `TButtonLayout` tipli `Layout` xassəsi müəyyənləşdirir. Bu xassə aşağıdakı qiymətləri ala bilər:

`blGlyphLeft` -təsvir yazıdan solda (susmaya görə)

`blGlyphRight` -təsvir yazıdan sağda

`blGlyphTop` -təsvir yazıdan yuxarıda

`blGlyphBottom` -təsvir yazıdan aşağıda

Bunlardan başqa, `BitBtn` düyməsi üçün `Margin` və `Spacing` (hər ikisi integer tipli) xassələri var. Bu xassələr uyğun olaraq təsvir və yazılan düymənin kənarlarına görə nizamlamaq və təsvirlə yazı arasındakı məsafəni (piksəllə) müəyyənləşdirmək üçündür. Susmaya görə, hər iki xassənin qiyməti (-1)-ə bərabərdir, yəni təsvir və yazı düymənin mərkəzinə nisbətən simmetrik yerləşmişdir

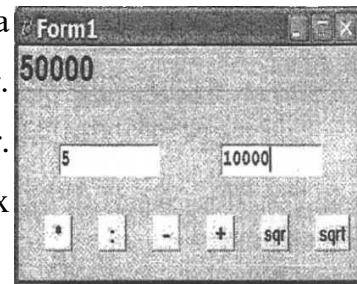
Cəld müdaxilə düyməsi

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 razılaşıdırırlar, yəni bir düymənin seçilməsi o birinin seçilməsini ləğv edir. Düymənin qrupa mənsub olması Integer tipli GroupIndex xassəsi ilə müəyyən edilir.

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 xassəsi 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 mənimsənilir.



SpeedButton düyməsinin səthində üç yox, dörd ayrı-ayn təsvir ola bilər. *Səkil 1*
Ona görə də bu düymə üçün NumGlyph xassəsinin maksimal qiyməti 4-9 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 1).

Panell komponentini seçərək onun Aligment xassəsinə taLeftJustify, Align xassəsinə alTop qiyməti verin və sərlövhəsini pozun. Edit1 və Edit2 komponentlərinin Text xassəsinə pozun. Düymələrin sərlövhəsini şəkildəki kimi dəyişin.

Əvvəlki misalı tam təfəsilatı ilə izah etdiyimizdən, burada əlavə izahata ehtiyac görməyərək, məsələnin hazır modulunu Sizə təqdim edirik.

```
unit Unit1;  
  
interface  
  
uses  
    Windows, Messages, SysUtils, Classes, Graphics,  
  
    Controls, Forms, Dialogs, Buttons, StdCtrls, ExtCtrls;  
  
type  
    TForm1 = class (Tform)  
        Panell: Tpanel;  
        Edit1: TEdit;  
        Edit2: TEdit;  
        SpeedButton1: SpeedButton  
        SpeedButton2: SpeedButton  
        SpeedButton3: SpeedButton  
        SpeedButton4: SpeedButton  
        SpeedButton5: SpeedButton  
        SpeedButton6: SpeedButton  
  
        procedure Speed Button4 Click (Sender: TObject;  
        procedure Edit1 KeyPress (Sender: TObject;  
            var Key: Char);  
  
        procedure Speed Button1 Click (Sender: TObject);  
        procedure Speed Button5 Click (Sender: TObject);  
        procedure Speed Button3 Click (Sender: TObject);  
        procedure Speed Button6 Click (Sender: TObject);  
        procedure Speed Button2 Click (Sender: TObject);  
  
    end
```

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 təkliddə 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 c dəyişdiriciləri Button düyməsinin əmələ gəldiyi TButtonControl sinfindən yaranmışdır.

Müstəqil qeyd olunmuş dəyişdirici

Bu dəyişdirici CheckBox komponenti ilə yaradılır. Dəyişdirici sərlövhdən ibarət düzbucaqlı şəkildədir. 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 verilsə, 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-qurguların səs gücləndirici şkalasını göstərmək olar. Bu komponentin xassələrini məsələnin həlli prosesində öyrənsəyik.

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.

Şkala üzərində hərəkət edən (maus və ya idarəetmə klavişləri ilə) məkiyin mövqeyi vuruqlarını 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 xassə `ttHorizontal` qiyməti verir.

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 xassənin qiymətini dəyişmək lazımdır {999, 9999 və s.).

Position (Mövqe) xassəsi - məkiyin 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 kənarında 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 PageDow 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 bölg digər bölgüyə atılacaqdır.

İndi isə GroupBox qrup komponentini seçin. Bu komponent də yenidir. GroupBox komponenti düzbucaqlı haşiyədən və onun sol yuxarı küncündə yerləşən sərlövhədən ibarətdir. Bu komponentin xassəsini Hasil adlandırın. Onun üzərindəki Label3 komponentinin Aligment xassəsinə taLeft Justify qiyməti verin.

Mühazirə 22: 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.

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 olunur əlavə faylının adından, pəncərənin özü isə *Msg* məlumat sətiri və *Ok* düyməsindən ibarət olur. Biz bu proseduru həl 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 məlumardan ibarət məndir. *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 sıfıra 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 sətiri 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 sətiri bildirir; əgər istifadəçi *Cancel* düyməsini və ya *Esc* klavişini basarsa, funksiyanın nəticəsi bu sətirdən ibarət olur. Məsələn, əgər proqramda

```
InputBox ('İstifadəçi', 'Soyadı', 'Abbasov');
```

yazılırsa, 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 sətirin) yerində *Value* dəyişəni istifadə olunur. Bu parametrdə 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);

yazılırsa, InputBox funksiyasının ekrana çıxardığı məlumat pəncərəsi ilə eyni olan sorğu ekrana çıxarılır.

Dialog 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:

<i>OpenDialog</i>	- açılacaq faylın seçilməsi;
<i>SaveDialog</i>	- yadda saxlanılacaq faylın seçilməsi;
<i>OpenPictureDialog</i>	- açılacaq qrafik faylın seçilməsi;
<i>SavePictureDialog</i>	- yadda saxlanılacaq qrafik faylın seçilməsi;
<i>FontDialog</i>	- şrift parametrlərinin təyini;
<i>ColorDialog</i>	- rəngin seçilməsi;
<i>PrintDialog</i>	- printerdə çap etmə;
<i>PrinterSetupDialog</i>	- printerin seçilməsi və onun parametrlərinin təyini;
<i>FindDialog</i>	- axtarılacaq mətn sətirinin daxil edilməsi;
<i>ReplaceDialog</i>	- axtarılacaq və əvəz olunacaq mətn sətirinin 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 əlaqələndiriləcəkdir. Belə hadisə kimi adətən ya menyuların bəndlərinin seçilməsi, ya da düymələr basılması hadisələri istifadə edilir.

İstənilən standart dialoq Execute metodu ilə çağrılır. Bu 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 *OpenDialog*.

FileName və Color yazılmışdırsa, onda istifadəçinin seçdiyi fayl və ya rəng yüklənəcəkdir.

Fayllarm açılması və yadda saxlanması

Yuxarıda qeyd etdiyimiz kimi, bu əməliyyatlar uyğun olaraq penDialog və SaveDialog komponentləri ilə həyata keçirilir. Bu komponentlərin əsas xassələri aşağıdakılardır:

FileName - faylın adını və ona tam yolu göstərir;

Title - dialoq pəncərəsinin sərlovhəsini müəyyən edir, əgər bu xassəyə qiymət verilməzsə, susmaya görə pəncərənin sərlovhəsi Open (vəya Save) File olur;

InitialDir –dialoq pəncərəsi açıldıqda təsvir olunan qovluq 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 iyirmiye qədər parametrləri vardır və hər bir parametirin 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) müəyyələşdirilir. Filtr bir-birindən "|" işarəsi ilə ayrılan qiymətlərdən iir. Hər bir qiymət təsvir və örtükdən

ibarət olur. Təsvir örtüyü izah edən mətndir (məsələn, "mətn faylları") örtük isə faylın adı və tipindən ibarət olur (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:

*Open Dialogl.Filter: ='Mətn faylları
I*.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. Bu redaktor Filter Name (Filtrin adı) və Filter (Filtr) sütunlarından ibarətdir. Birinci sütunda filtrin təsviri, ikinci sütunda isə uyğun örtük göstərilir.

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.

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.

MaxFont Size -şriftin maksimal ölçüsünü müəyyən edir;

MinFont Size -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;

Bo kodla dialoq pəncərəsindən istifadəçinin seçdiyi şrift yazı üçün tətbiq olunur.

Mühazirə 23: 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 əməlləri 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 yazılırsa, 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 aktividir.

Visible xassəsi - Boolean tipli Visible xassəsi ekranda menyu bəndinin görünməsinə 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: TShift State) : TShortcut;

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 - TMenuItem tipli Break xassəsi menyunun sütünlara bölünməsinə təyin edir. Bu xassə aşağıdakı qiymətlərdən birini ala bilər:

mbNone -menyu sütünlara bölünmür (susmaya görə);

mbBreak -cari bənddən başlayaraq menyu yeni sütun əmələ gətirir;

mbtBreakBar-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 verilsə, onda menyu bəndinin sərlövhəsində xüsusi qeydetmə nişanı əmələ gəlir. Susmaya görə Checked xassəsinə False qiyməti verilmişdir, ona görə də menyu bəndi seçilmir.

Radioltem xassəsi - Boolean tipli Radioltem 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 - TMenuItem tipli Items xassəsi menyu bəndlərindən ibarət massivdir. Bu xassə ilə menyunun ayrı-ayrı bəndlərinə Items(O).

Count xassəsi - Integer tipli Count xassəsi menyuda bəndlərin sayını bildirir. Əgər menyuda bənd yoxdursa, həmin menyu üçün Count xassəsi sıfır barabər 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.

Alignment xassəsi - TPopupMenuAlignment tipli Alignment 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 menyunuda əmələ gəlməsi üçün, onun PopupMenu xassəsinə qiymət kimi, tələb olunan kontekst menyunun adı mənimsədilməlidir. Məsələn, Label komponentinə aid kontekst menyunun yaradılması üçün proqramda

```
Label.PopupMenu:=PopupMenu1;
```

yazılmalıdır.

Maus və ya klaviatura ilə menyu bəndini seçdikdə baş verən əsas OnClick hadisəsidir. Əksər hallarda, əlavələrdə eyni bir əməliyyat 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.

Misal. Menyu bəndinin seçilməsini imitasiyası.

```
procedure TForm1.Button1Click (Sender: TObject);
```

```
begin
```

```
mnuOpen.Click;
```

```
end;
```

Burada, Button1 düyməsi basıldıqda, mnuOpen (ad şərtidir) bəndinin icra etdiyi əməliyyat yerinə yetiriləcəkdir.

Layihələndirmə zamanı menyuların yaradılması xüsusi *konstruktorunda* yerinə yetirilir. Menyuları dinamik olaraq, proqramlaşdırma yolu ilə də yaratmaq mümkündür.

Menyu konstruktoru

Əlavələrin layihələndirilməsi prosesində menyuları yaratmaq və ya dəyişdirmək üçün Delphi-də Menyü konstruktorundan (*Menu* istifadə olunur. Bu konstruktoru çağırmaq üçün forma üzərində *MainMta* ya *PopupMenu* komponentləri yerləşdirərək kontekst menyudar. *Designer...* əmrini icra etmək və ya bu komponentlər üzərində düyməsini iki dəfə basmaq lazımdır. Bu redaktorla yaradılan menyü yerinə yetirildikdən sonra necə görünəcəkdirsə, elə o cür də görünür.

Menyü konstruktoru ilə işlədikdə aşağıdakı kontekst menyulardast etməklə menyuların yaradılması və dəyişdirilməsi prosesini sürətləndirmək olar:

Insert -menyü bəndini *əlavə* etmək;

Delete-menyü bəndini *pozmaq*;

Create Submenu - *alt menyü yaratmaq*;

Select Menu - menyünü *seçmək*,

Save Template - menyünü *şablon* kimi saxlamaq;

Insert From Template - menyü *şablonlarını pozmaq*;

Delete Template - menyünü *şablondan yükləmək*;

Insert From Resource... - menyünü *resurslardan yükləmək*.

Redaktorla işləyərkən, Obyektlər inspektorundan istifadə etməklə menyü bəndlərinin xassələrinə qiymətlər verilir.

Menyular yaradıldıqda *drag-and-drop* texnologiyası ilə menyü bəndlərinin yerini dəyişdirmək olar.

Menyular yaradıldıqdan sonra, onlar üçün prosedurlar yaradıldıqda (bənd üzərində mausun düyməsini basmaqla), prosedurun sərlovhəsində bu bəndin nömrəsi göstərilir, məsələn:

Procedure Tforml. N3Click (Sender:TObject);

Əgər Obyektlər inspektorunda menyü bəndi üçün Name xassəsinə ad verilərsə, onda prosedurun sərlovhəsində həmin ad göstərilir, məsələn:

Procedure Tforml.mnu CloseClick (Sender:TObject);

Burada, *mnuClose* menyü bəndinin adıdır və tamamilə şərti seçilmiş addır. Lakin, unutmayın ki, bu ad yalnız latin hərflərindən və rəqəmlərdən ibarət ola bilər.

Menyu sətirlərindən ibarət mətn redaktorunun yaradılması

Dialoglar bölməsində yaratdığımız, düymələrlə idarə olunan mətn redaktorunu yenidən yaradaq. Bu dəfə düymələrin icra etdiyi əməlləri menyular vasitəsilə icra edək. Eyni zamanda menyu bəndlərini proqram yolu ilə deyil, Menyu konstrukturu ilə yaradaq.

Bu redaktorda menyulan 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 sətiri üç menyudan ibarət olacaqdır.

Kontekst menyunu isə aşağıdakı bəndlərdən ibarət tərtib edək:

- Exit - *çıxmaq*;
- Reset – *bərpa etmək*;
- Color- *rəng seçmək*;
- Font-*şrift seçmək*..

Bu dəfə redaktorda heç bir düymə istifadə etməyəcəyik.

Yeni layihə üçün forma üzərinə Memo, MainMenu, PopupMBUh, OpenFileDialog, SaveDialog, FontDialog və CoicrZialof komponentləri yerləşdirin. MainMenu komponentini seçərək, onun mausun sağ düyməsini basıb, kontekst menyudan Menu Designer.... konstrukturu 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 ilə menyuya avtomatik olaraq NI 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 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 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, 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 sətiri yaradılmış olacaqdır.

İndi Memol 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 (Memol) pozun. Bunun üçün Lines xassəsi qarşısındakı üç nöqtə təsvirli düyməni iki dəfə basaraq açılan pəncərədən Memol sözünü pozun.

OpenDialogl komponentini seçib Filter xassəsi qarşısında mausun düyməsini basaraq *Filter Editor* redaktorunu çağırın. Bu redaktorun birinci sütununun birinci sətirinə *Mətn faylları *.txt, *.doc*, ikinci sütununun həmin sətirində **.txt, *.doc* yazın. İkinci sətirin birinci sütununda *Bütün fayllar *.**, ikinci sütununda isə **.** yazıb *Ok* düyməsini basın. Obyektlər inspektorunda CefaultExt xassəsinə 'txt' qiyməti daxil edin. Bütün bu əməliyyatları SaveDialogl 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 menyü bəndlərini imitasiya kodu yazılacaqdır.

Beləliklə, menyularla idarə olunan məm 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;  
        MainMenu: 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;
OpenDialogI: TOpenDialog;
SaveDialogI: TSaveDialog;
FontDialogI: TFontDialog;
ColorDialogI: 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 mnuSelectAHClick (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

FormI: TFormI;
FormColorYad, MemoColorYad: LongInt;
FAYL: String;
implementation
{SR *.DEM}
procedure TFormI.FormActivate(Sender: TObject);
begin
OpenDialogI.Title:= ' Mətn faylları ';
OpenDialogI.Filter:= 'Mətn faylları[*.TXT;*.DOC]
                    |*.TXT;*.DOC|Bütün fayllar*.*|I*.*';
OpenDialogI.DefaultExt:= 'TXT';
SaveDialogI.Title:= ' Mətn faylları ';
SaveDialogI.Filter:= ' Mətn faylları*.TXT;*.DOC/

```

```

        *.TXT;*.DOC|Bütün fayllar*. * I *.*';
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 OpenFileDialog1.Execute then
begin
    FAYL:= OpenFileDialog1.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
Memol.Lines.SaveToFile(FAYL);
Close;
end;
end
else Close;
end;

procedure TForm1.mnuUndoClick(Sender: TObject);
begin
SendMessage(Memol.Handle,EM_UNDO,0,0);
end;

procedure TForm1.mnuSelectAllClick (Sender: TObject) ;
begin
MEMol.HideSelection:=False;
Memol.SelectAll;
end;

procedure TForm1.mnuResetClick (Sender: TObject);
begin
Memol.Color:= MemoColorYad;
Form1.Color:= FormColorYad;
end;

procedure TForm1.mnuColorClick(Sender: TObject);
begin
if ColorDialog1.Execute then
Memol.Color:= ColorDialog1.Color;
end;

```

```

procedure TForm1.FormClose(Sender:TObject;
    var Action:TCloseAction);
begin
    if Memol.Modified then
        if MessageDlgC (Fayl dəyişmişdir! '+#13#10+
            ' Çıxırsınız mı? ,mtConfirmation,
            [mbYes, mbNo], 0) = mrYes then
                Begin
                    if Memol.Modified then
                        begin
                            Memol.Lines.SaveToFile (FAYL);
                            Action: = caFree;
                        end;
                    end
                else Action: = caNone;
            end;
end;

```

Mühazirə 23: Menyularla iş

Menyu Windows sistemində və onun əlavələrində ən vacib elementdir və demək olar ki, elə bir pəncərə yoxdur ki, orada menyu sətiri olmasın. Menyu müəyyən funksional əlamətlərə görə birləşdirilmiş bəndlər yığımından ibarətdir və hər bir bənd müəyyən əmri icra edir. Windows sistemindən bilir ki, menyular əsas menyu və kontekst (peyda olan) menyulardan ibarətdir. *Əsas menyu* menyular sətiri 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 yazılırsa, 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 aktividir.

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 funksiyaları 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; Shif t: TShif tState) : 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 sütünlara bölünməsini təyin edir. Bu xassə aşağıdakı qiymətlərdən birini ala bilər:

mbNone -menyu sütünlara bölünmür (susmaya görə);

mbBreak -cari bənddən başlayaraq menyu yeni sütun əmələ gətirir;

mbtBreakBar-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.

Radioltem xassəsi - Boolean tipli Radioltem 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 - TMenuItem tipli Items xassəsi menyu bəndlərindən ibarət massivdir. Bu xassə ilə menyunun ayrı-ayrı bəndlərinə Items(O).

Count xassəsi - Integer tipli Count xassəsi menyuda bəndlərin sayını bildirir. Əgər menyuda bənd yoxdursa, həmin menyu üçün Count xassəsi sıfıra 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 - TPopupMenuAlignment 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 menyunuda əmələ gəlməsi üçün, onun PopupMenu xassəsinə qiymət kimi, tələb olunan kontekst menyunun adı mənimsədilməlidir. Məsələn, Label komponentinə aid kontekst menyunun yaradılması üçün proqramda

Label.PopupMenu:=PopupMenu1;

yazılmalıdır.

Maus və ya klaviatura ilə menyu bəndini seçdikdə baş verən əsas OnClick hadisəsidir. Əksər hallarda, əlavələrdə eyni bir əməliyyat 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.

Misal. Menyu bəndinin seçilməsini imitasiyası.

procedure TForm1.Button1Click (Sender: TObject);

begin

mnuOpen.Click;

end;

Burada, Button1 düyməsi basıldıqda, mnuOpen (ad şərtidir) bəndinin icra etdiyi əməliyyat yerinə yetiriləcəkdir.

Layihələndirmə zamanı menyuların yaradılması xüsusi *konstruktorunda* yerinə yetirilir. Menyuları dinamik olaraq, proqramlaşdırma yolu ilə də yaratmaq mümkündür.

Menyu konstrukturu

Əlavələrin layihələndirilməsi prosesində menyuları yaratmaq və ya dəyişdirmək üçün Delphi-də Menyu konstrukturdan (*Menu* istifadə olunur. Bu konstrukturu çağırmaq üçün forma üzərində MainMta ya PopupMenu komponentləri yerləşdirərək kontekst menyudar. *Designer...* əmrini icra etmək və ya bu komponentlər üzərində düyməsini iki dəfə basmaq lazımdır. Bu redaktorla yaradılan menyu yerinə yetirildikdən sonra necə görünəcəkdirsə, elə o cür də görünür.

Menyu konstrukturu ilə işlədikdə aşağıdakı kontekst menyulardast 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 - menyu *şablonlarını* pozmaq;

Delete Template - menyunu *şablondan yükləmək*;

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 sərlovhəsində bu bəndin nömrəsi göstərilir, məsələn:

Procedure TForm1.N3Click (Sender:TObject);

Əgər Obyektlər inspektorunda menyu bəndi üçün Name xassəsinə ad verilərsə, onda prosedurun sərlovhəsində həmin ad göstərilir, məsələn:

Procedure TForm1.mnu CloseClick (Sender:TObject);

Burada, mnuClose menyu bəndinin adıdır və tamamilə şərti seçilmiş addır. Lakin, unutmayın ki, bu ad yalnız latin hərflərindən və rəqəmlərdən ibarət ola bilər.

Menyu sətirlərindən ibarət mətn redaktorunun yaradılması

Dialoglar bölməsində yaratdığımız, düymələrlə idarə olunan mətn redaktorunu yenidən yaradaq. Bu dəfə düymələrin icra etdiyi əməlləri menyular vasitəsilə icra edək. Eyni zamanda menyu bəndlərini proqram yolu ilə deyil, Menyu konstrukturu ilə yaradaq.

Bu redaktorda menyulan belə qruplaşdırmaq:

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 menyusu sətri üç menyudan ibarət olacaqdır. Kontekst menyunu isə aşağıdakı bəndlərdən ibarət tərtib edək:

- Exit - çıxmaq;
- Reset – bərpa etmək;
- Color- rəng seçmək;
- Font-şrift seçmək..

Bu dəfə redaktorda heç bir düymə istifadə etməyəcəyik.

Yeni layihə üçün forma üzərinə Memo, MainMenu, PopupMBU, OpenFileDialog, SaveDialog, FontDialog və CoicrZialof komponentləri yerləşdirin. MainMenu komponentini seçərək, onun 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 ilə 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 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 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... menyusu bəndini yaratdıqdan sonra, növbəti təklif olunan bəndin Caption xassəsinə ad deyil, 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 sətiri yaradılmış olacaqdır.

İndi Memol 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 (Memol) pozun. Bunun üçün Lines xassəsi qarşısındakı üç nöqtə təsvirli düyməni iki dəfə basaraq açılan pəncərədən Hemol sözünü pozun.

OpenDialogl komponentini seçib Filter xassəsi qarşısında mausun düyməsini basaraq *Filter Editor* redaktorunu çağırın. Bu redaktorun birinci sütununun birinci sətirinə *Mətn faylları *.txt, *.doc*, ikinci sütununun həmin sətirində **.txt, *.doc* yazın. İkinci sətirin birinci sütununda *Bütün fayllar *.**, ikinci sütununda isə **.** yazıb *Ok* düyməsini basın. Obyektlər inspektorunda CefaultExt xassəsinə 'txt' qiyməti daxil edin. Bütün bu əməliyyatları SaveDialogl 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 məm redaktorunun modulunun tam mətni belə olacaqdır:

```
unit Unitl;  
  
interface  
  
uses  
    Windows, Messages, SysUtils, Classes,  
    Graphics, Controls, Forms, Dialogs,  
    Menus, StdCtrls;  
  
type  
    Tforml = 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;
    PopupMenu: TPopupMenu;
    mkExit: TMenuItem;
    mkReset: TMenuItem;
    mkColor: TMenuItem;
    mkFont: TMenuItem;
    OpenDialog: TOpenDialog;
    SaveDialog: TSaveDialog;
    FontDialog: TFontDialog;
    ColorDialog: 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
{SR *.DEM}
procedure TForm1.FormActivate(Sender: TObject);
begin
  OpenFileDialog.Title:= ' Mətn faylları ';
  OpenFileDialog.Filter:= "Mətn faylları[*].TXT, *.DOC]
                          [*].TXT;*.DOC/Bütün fayllar*.*|I*.*";
  OpenFileDialog.DefaultExt:= 'TXT';
  SaveDialog1.Title:= ' Mətn faylları ';
  SaveDialog1.Filter:= ' Mətn faylları*.TXT;*.DOC]
                      *.TXT;*.DOC/Bütün fayllar*.*|I*.*';
  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 OpenFileDialog.Execute then
          begin
            FAYL:= OpenFileDialog.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;
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;

```

```

Forml.Color:= FormColorYad;
end;
procedure TForml.mnuColorClick(Sender: TObject);
begin
if ColorDialogl.Execute then
Memol.Color:= ColorDialogl.Color;
end;

procedure TForml.FormClose(Sender:TObject;
var Action:TCloseAction);
begin
if Memol.Modified then
if MessageDlgC (Fayl dəyişmişdir! '+#13#10+
' Çıxırsınız mı? ,mtConfirmation,
[mbYes, mbNo], 0) = mrYes then
Begin
if Memol.Modified then
begin
Memol.Lines.SaveToFile (FAYL);
Action:= caFree;
end;
end
else Action:=caNone;
end;

```