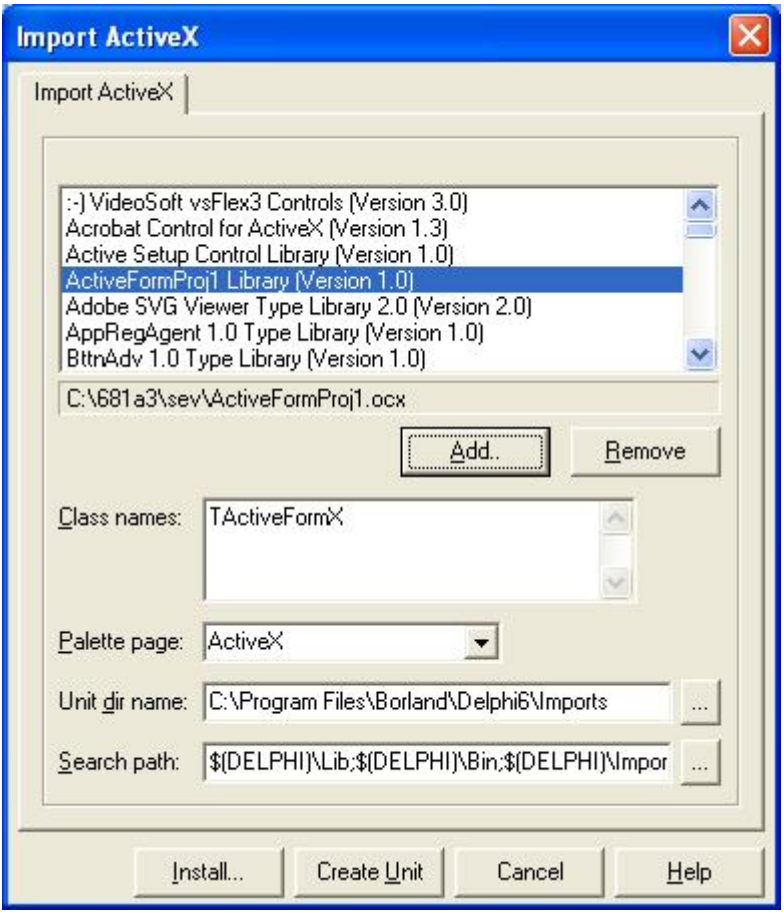
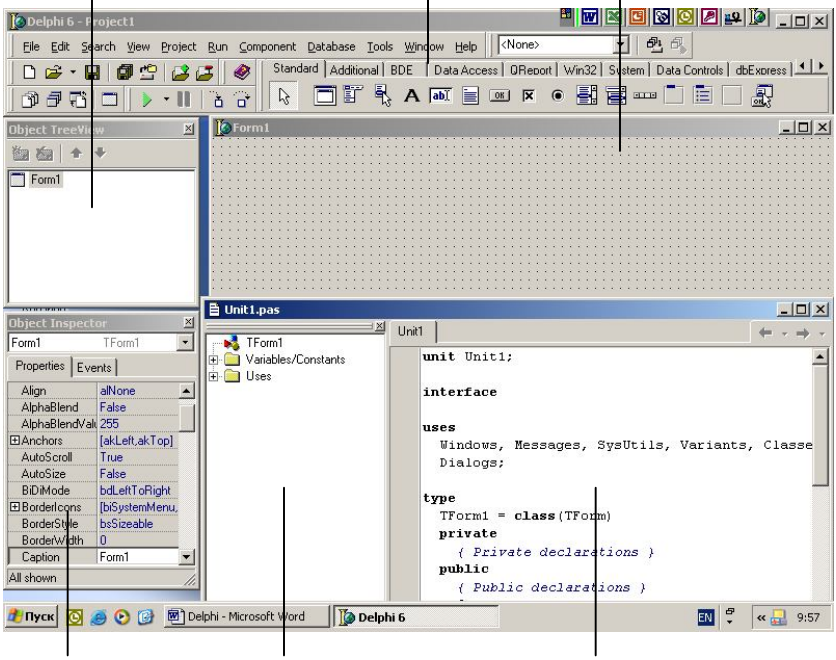


1. Delphi mühiti ile tanishliq

Delphi muhiti proqramnizin yuksek effektivli ishini temin eden murekkeb mexanizmdir. Vizual olaraq bu ekranda eyni zamanda bir neche pencerenin achilmasi vasitesi ile reallashdirilir. Bu pencereler ekran boyu hereket ede , bir-birinin bir hissesini ve ya tam orte bilirler.

Delphini ishe saldiqda ekranda shekil 1.1-dekine uygun pencere achilacaq.

Agacvari obyektler penceresi esas pencere Forma penceresi



Obyektler Inspektoru Penceresi Brauzer penceresi Proqram kodunun penceresi

Shekil 1.1 Delphi-nin en vacib pencereleri.

Shekilde Delphi-nin alti en vacib penceresi tesvir olunmushdur: esas pencere (bu pencere Delphi 6 – Project1 bashligia malikdir), forma penceresi (Form1 bashligi), agacvari obyektler penceresi(Object TreeView), brauzer penceresi(Code Explorer), obyektler inspektoru penceresi (Object Inspector) ve proqram kodunun penceresi (Unit1.Pas). Pencereleri shekilde gorunduyu kimi nizamlamaq uchun onlarin olchu ve veziyyetlerini el ile deyishmek lazimdir. Bir qayda olaraq proqram kodu penceresi tamamile forma penceresi ile ortulu olur. Pencerelerin olchu ve veziyyetleri onlarin funksionalligina hech bir tesir gostermir.

eger ekranda kod penceresi görunmurse, onu ekrana getirmek uchun sadece olaraq F12 duymesini basmaq lazimdr. F12 duymesinin tekrar basilmasi yeniden forma penceresini aktivleshdirir.

1.1. esas pencere

esas pencere yaradilacaq proqram proyektinin idaresi uchun lazim olan esas funksiyalari heyata kechirir. Bu pencere hemishe ekranda gorunur ve ekrandn en yuxari hissesini tutur. Onu butun ekran boyu yaymaq mumkun deyil. Maksimallashdirma veziyyetinde onun olchu ve veziyyeti praktiki olaraq adi haldakindan ferqlenmir.

Bu esas pencerenin funksionalligi ile baglidir: bir terefden o proqramchinin hemishe elinin altinda olmasi vacib olan elementleri ozunde saxlayir, diger terefden esas pencere Delphi-nin diger pencerelerinden ekran hissesini göturmemelidir. esas pencerede  duymesinin vurulmasi ekrandan Delphi-nin diger pencerelerinin yox olmasina, sistemin ise tapshiriqlar paneline atilmasina getirir. Onun bashlanmasi ise proqramchinin sistem proqramlashdirma ile ishinin qurtardigini gosterir.

Esas pencerede Delphi-nin esas menyusu, piktoqrafik emr duymelerinin mecmuusu ve komponentler palitrasi yerleshir.

1.1.1. esas menyu

esas menyu proyektin idaresi uchun butun zeruri imkanlara malikdir. esas menyunun butun rejimleri ikinci seviyyeli menyulara kechmeye imkan veren bashliqlardan ibaretdir.

1.1.1.1. File menyusu

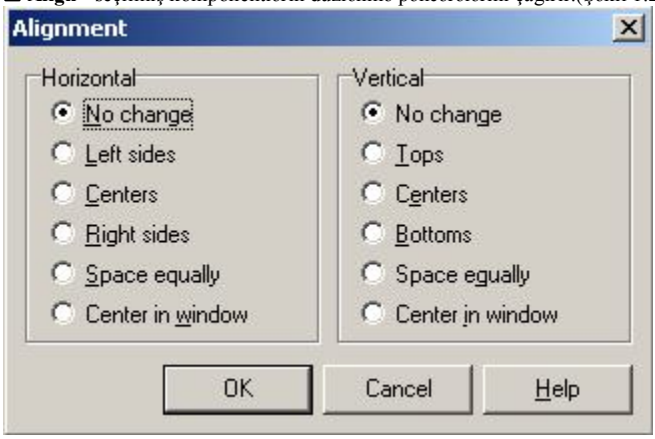
- **New**- altmenyusu ashagidaki emrlerden ibaretdir:
  - **Application** - **Windows** uchun yeni proqram yaradir;
  - **CLX Application** - **Windows** ve **Linux** uchun yeni proqram yaradir;
  - **Data Module** - yeni verilenler modulu yaradir;
  - **Form** - yeni forma yaradir ve onu proyekte qoshur;
  - **Frame** - yeni cherchive yaradir.
  - **Other** - obyektler repozitoriyasi(arxivi) penceresini achir.
- **Open** - evvelceden yaranmesh formani achir.
- **Open Project** - evvelceden yaranmesh proyekti achir.
- **Reopen** - əvvəl yükləmiş forma və proyektlərin siyahısını təkrar yüklənmək üçün çağırır.
- **Save** - cari formanı yadda saxlayır.
- **Save As** - cari formanı başqa adla yadda saxlayır.
- **Save Project As** - proyekt faylını və bütün fəal modulları yadda saxlayır.
- **Close** - cari formanı bağlayır.
- **Close All** - bütün fəal faylları bağlayır.
- **Use Unit** - cari formaya başqa modula müraciət yerləşdirir.
- **Print** - fəal forma və ya modulu çapa verir
- **Exit** - Delphinin işini sona çatdırır.

1.1.1.2. Edit menyusu.

Edit menyusunun əmrləri aşağıdakılardır.

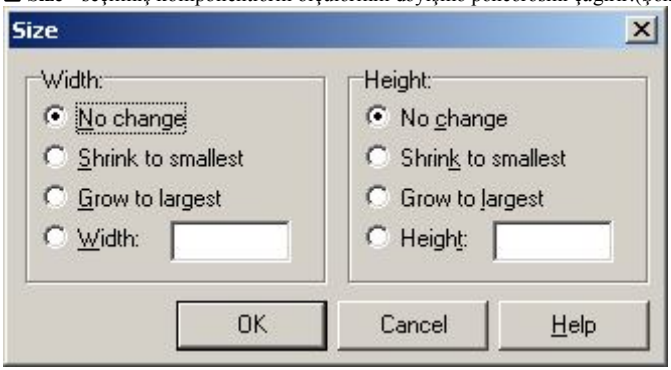
- **Undo** - proyektdə axırıncı dəyişikliyi ləğv edir.
- **Redo** - proyektdə axırıncı edilmiş dəyişikliyi bərpa edir.

- **Cut** - formanın seçilmiş komponentini və ya mətn fraqmentini kəsib götürərək, mübadilə buferinə (**Clipboard**) yerləşdirir.
- **Copy** - formanın seçilmiş komponentinin və ya modulun mətn fraqmentinin surətini mübadilələr buferinə yerləşdirir.
- **Paste** - mübadilə buferindən komponenti formaya və ya mətn fraqmentini modula qoyur.
- **Delete** - seçilmiş komponenti və ya mətn fraqmentini ləğv edir.
- **Select All** - formanın bütün komponentlərini və ya modulun bütün mətnini seçir.
- **Align To Grid** - seçilmiş komponentləri miqyas şəbəkəsində onların sol yuxarı küncünü şəbəkənin yaxın nöqtələrinə yerləşdirməklə qoşur.
- **Bring To Front** - seçilmiş komponentləri ön plana keçirir.
- **Send To Back** - seçilmiş komponentləri arxa plana keçirir.
- **Align** - seçilmiş komponentlərin düzlənmə pəncərələrini çağırır.(Şəkil 1.2).



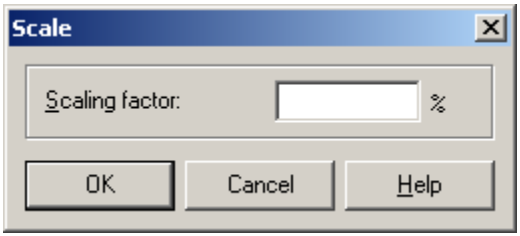
Şəkil 1.2 Komponentlərin düzlənmə pəncərəsi

- **Size** - seçilmiş komponentlərin ölçülərinin dəyişmə pəncərəsini çağırır.(Şəkil 1.3).



Şəkil 1.3 Ölçüləri dəyişmə pəncərəsi.

- **Scale** - seçilmiş komponenti miqyaslaşdırır.(Şəkil1.4).



Şəkil 1.4 Komponentlərin miqyaslaşdırma pəncərəsi

- **Tab Order Tab -** düyməsinin basılması ilə komponentlər üzərində gəzişmə qaydasını dəyişir.
- **Creation Order** - qeyri vizual komponentlər yaradılması qaydasını dəyişir.
- **Flip Children**-bütün və ya seçilmiş komponentlər üçün **BiDiMode** xassəsini dəyişdirir.
- **Lock Controls** - Komponentin üzərində hərəkətini qadağan edir.
- **Add To Interface** - **ActiveX** komponenti üçün yeni xassə, metod və hadisələr təyin edir.

- **No change** - komponentlər düzləndirilmir.
- **Left sides** – komponentlər sol sərhəddə görə nizamlanırlar.
- **Centers** - komponentlər etalonun sərhəddinə nəzərən mərkəzə doğru düzlənirlər. Etalon birinci seçilmiş komponentdir.
- **Right sides** - komponentlər etalonun sağ tərəfinə görə düzlənirlər.
- **Space equally** - bütün komponentlər üfqi və şaquli olaraq eyni məsafəyə düzləndirilir.
- **Center in Window** - bütün komponentlər pəncərənin sərhədindən mərkəzə düzləndirilir.
- **Tops** - komponentlər yuxarı kənara görə düzləndirilir.
- **Bottoms** - komponentlər aşağı kənara görə düzləndirilir.

Ölçüləri dəyişmə pəncərəsində **Width** seçilmiş komponentləri eninə, **Height**-hündürlüyə görə nizamlayır.

- Shrink to smallest - komponentlərin ölçüləri onların ən kiçiyinin ölçüsünə qədər kiçilir.
- **Grow to largest** - komponentlərin ölçüləri onların ən böyüyünün ölçüsünə qədər böyüyür.

Width - komponentlərin enini verməyə imkan verir.
Height-komponentlərin hündürlüyünü verməyə imkan verir.
Scallinq factor pəncərəsi komponentlərin miqyaslaşdırma əmsalını cari ölçülərinə nəzərən faizlə verməyə imkan verir.
.

1.1.1.3. Search menyusu.

Search menyusunun əmrləri aşağıdakılardır.

- **Find** - mətn fraqmentini axtarır, tapdıqda onu işıqlandırır.
- **Find In Files** - Bütün proyekt fayllarında, yalnız açılmış fayllarda və ya cari kataloqun bütün fayllarında mətn fraqmentini axtarır.
- **Replace** - mətn fraqmentini axtarır və əvəz edir.
- **Search Again** - axtarışı və ya axtarış və əvəzetməni təkrar edir.
- **Incremental Search** - Mətni onun daxil etmə ardıcıllığına görə əvvəlcə birinci hərfi, sonra ikincihərfi və s. axtarır.
- **Go to Line Number** - kursoru faylın başlanğıcından göstərilən sətrə hərəkət etdirir.
- **Find Error** - proqram yerinə yetirilərkən səhvin ünvanına görə səhvlə bağlı kod fraqmentini axtarır.

🖶 **Browse Symbol** - proqramda simvolun təyin olunduğu yeri göstərir.(əmr proqram ancaq müvəffəqiyyətlə yerinə yetirildikdə mümkündür). Proyektin istənilən qlobal identifikatoru simvol adlanır.

1.1.1.4. View menyusu.

View menyusunun əmrləri aşağıdakılardır:
🖶 **Project Manager** - proyekt menecerinin pəncərəsini göstərir.
🖶 **Translation Manager**-translyasiya menecerini açmaq imkanı verir.
🖶 **Object Inspector -** Obyektlər inspektoru pəncərəsini göstərir.
🖶 **Object TreeView** - ağacvari obyektlər pəncərəsini göstərir.
🖶 **To-Do-List** - Tapşırıqlar siyahısını açmağa imkan verir.
🖶 **Alignment Palette** - komponentlərin düzləndirilmə palitrasını göstərir.
🖶 **Browser** - Obyektlər brouzeri pəncərəsini göstərir.
🖶 **Code Explorer**-Kodlar brouzerinin gizlədilmiş pəncərəsini göstərir.
🖶 Component List - Komponentlərin seçilməsi pəncərəsini göstərir.
🖶 **Window List** - açılmış proyektlərin siyahısı altmenyusu.
🖶 **Debug Windows** - aşağıdakı əmrlərlə sazlanma pəncərəsi alt menyusu.
- **Breakpoints** - dayanma nöqtəsi pəncərəsini göstərir.
- **Call Stack -** qoşulma pəncərəsini göstərir;
- **Watching Expressions** - dəyişənləri/ifadələri müşahidə pəncərəsini göstərir;
- **Local Variables**- sazlanma rejimində lokal dəyişənlərin dəyişməsini müşahidə etməyə imkan verir;
- **Threads -** əmrlər axınının statusu pəncərəsini göstərir;
- **Modules** - proyektlər modulu pəncərəsini göstərir;
- **Event Log** - hadisələrin qeydiyyat kitabını göstərir;
- **CPU** - mərkəzi prosessorun reqistrlərinin vəziyyətini göstərir;
- **FPU - hesabı köməkçi prosessorun reqistrlərinin vəziyyətini göstərir;**

🖶 **Desktops** - əsas pəncərələrin konfiqurasiyasının idarə altmeyusu aşağıdakı əmrlərdən ibarətdir:
- **Save Desktop** - cari konfiqurasiyasını yadda saxlayır;
- **Delete** - əvvəl yadda saxlanmış konfiqurasiyanın sazlanmasını təyin edir;
- **Save Debug Desktop – cari konfiqurasiyanı sazlanmış təyin edir.**

🖶 **Toggle Form** və ya **Toggle Unit** - forma pəncərəsindən kod pəncərəsinin aktivliyini təmin edir və əksinə.
🖶 **Units** - modullar pəncərəsini göstərir;
🖶 **Forms** - forma pəncərəsini göstərir;
🖶 **Type Library** - tiplər kitabxanası pəncərəsini göstərir;
🖶 **New Edit Window** - cari modulun kodu ilə yeni pəncərə açır.
🖶 **Toolbars** - alətlər panelinin nizamlanma pəncərəsini göstərir.
🖶 **View as Form** və ya **View as Text** - forma pəncərəsini adi və ya mətni təsvirdə göstərir.
🖶 **Next Window** - proyektə qoşulmuş növbəti modulu göstərir.

1.1.1.5. Project menyusu

Project menyusunun əmrləri aşağıdakılardır.
🖶 **Add To Project**-proyektə fayl əlavə edir.
🖶 **Remove From Project**-proyektlərdən faylı ləğv edir.
🖶 **Import Type Library - ActiveX** elementlərinin tiplərini proyektlər kitabxanasına göndərir.
🖶 **Add To Repository** - proyekti repozitorlar ambarına yerləşdirir.
🖶 **View Source**- proyekt kodunu pəncərəsi ilə göstərir.
🖶 **Languages** - mövcud lokallaşdırma dillərindən yenisini əlavə etmək, ləğv etmək, və ya əsas etməyə imkan verir, aşağıdakı altmenyulardan ibarətdir.

- **Add -** yeni lokallaşdırma dili əlavə edir.
- **Remove** - mövcud lokallaşdırma dilini ləğv edir.
- **Set Active** - lokallaşdırma dilini fəallaşdırır.
- **Update Resources DLL** – proqramı lokallaşdırmağa imkan verən **DLL** resurs kitabxanasını yenidən yaradır.

🖶 **Add New Project** - cari proyekt qrupuna proqram, **DLL** və ya paket əlavə edir.
🖶 **Add Exists Project** - proyekti açır və onu cari proyekt qrupuna əlavə edir.
🖶 **Compile Proyektin_adı** - proyektin əvvəlki kompilyasiyadan sonra dəyişiklik olunmuş modullarını kompilyasiya edir.
🖶 **Build Proyektin_adı** - proyektin bütün modullarını kompilyasiya edir və yerinə yetirilməyə hazır proqram yaradır.
🖶 **Syntax Check Proyektin_adı** - proqramın sintaksis olaraq düzgünlüyünü yoxlayır.
🖶 **Information**-proqram haqqında informasiyasını göstərir.
🖶 **Compile All Projects**- proyektin əvvəlki kompilyasiyadan sonra dəyişiklik olunmuş, proyektlər qrupunun bütün fayllarını kompilyasiya edir.
🖶 **Build All Projects**-axırıncı kompilyasiya olunmuş proyektdə dəyişiklik olunub-olunmamasından asılı olmayaraq bütün fayllar qrupunu kompilyasiya edir.
🖶 **Web Deployment Options**-sizin **Web** serverə **ActiveX** və ya **ActiveForm** komponentlərini qoşur.Proyektin kompilyasiyasından əvvəl çağırılır.
🖶 **Web Deploy**-sizin **Web** serverə **ActiveX** və ya **ActiveForm** komponentlərini qoşur. Proyektin kompilyasiyasından sonra çağırılır.
🖶 **Options**-proyektin parametrlərinin nizamlanması üçün dialoq pəncərəsini göstərir.

1.1.1.6. Run menyusu.

Run menyusunun əmrləri aşağıdakılardır:
🖶 **Run**-proqramın kompilyasiya və yerinə yetirilməsini həyata keçirir.
🖶 Attach to Process-sazlanma zamanı digər şəbəkə maşınında işə salınmış prosesə qoşulmağa imkan verir.
🖶 **Parametrs**-sizin proqramın işə salınması üçün əmrlər sətrini təyin edir.
🖶 **Register ActiveX** Servers-sizin proyekti Windows reyestrində qeyd edir. Əmr yalnız **Active**X proyektləri üçün mümkündür.
🖶 **UnRegister ActiveX Servers**-sizin proyekti **Windows** reyestirindən ləğv edir. Əmr yalnız **ActiveX** proyektləri üçün mümkündür.
🖶 **Install MTS Objects**-sizin proyektdə MTS obyektini qeyd edir.
🖶 **Step Over**-sazlanma rejimində cari sətri çağırılan altproqramlara nəzarət etmədən yerinə yetirir.
🖶 **Trace info**-sazlanma rejimində cari sətri çağırılan altproqramlara nəzarət edərək yerinə yetirir.
🖶 **Trace To Next Source Line**-proqram kursorun cari vəziyyətindən ən yaxın məsafədə olan icra olunan operatora qədər yerinə yetirilir.

🖶 **Run To Cursor**- sazlanma rejimində proqramı yerinə yetirir və proqramda mətn kursorunun kodu olan sətrə rast gəldikdə dayanır.
🖶 **Run Until Return** – sazlama rejimində cari altproqramı yerinə yetirib dayanır.
🖶 **Show Execution Point**-kod pəncərəsində, proqramın yerinə yetirilməsinin kəsildiyi operatoru göstərir.
🖶 **Program Pause**-sazlanan proqramın yerinə yetirilməsini dayandırır.

- **Program Reset**-proqramın yerinə yetirilməsini dayandırır və proqramın işlənməsi rejimini bərpa edir.
- **Inspect**-cari qiymətin yoxlanması pəncərəsini açır.
- **Evaluate/Modify**-dəyişənlərin yoxlanma/dəyişmə pəncərəsini açır.
- **Add Watch**-dəyişən və ya ifadəni müşahidə pəncərəsinə əllavə edir.
- **Add Break point**-dayanma nöqtəsini əlavə edir.

1.1.1.7. Component menyusu

Component menyusunun əmrləri aşağıdakılardır:

- **New Component** - komponentlər eksperti pəncərəsini açır.
- **Install Component**-komponenti mövcud və ya yeni paketə əlavə edir.
- **Import ActiveX Control**-proyektə ActiveX komponentləri kitabxanasını əlavə edir.
- **Create Component Template**-komponentlər palitrasına şablon yerləşdirir.
- **Install Packages**-proqramın konstruksiya və yerinə yetirilməsi mərhələsində zəruri olan paketləri göstərməyə imkan verir.
- **Configure Palette**-komponentlər palitrasının nizamlanma dialoq pancərəsini çağırır.

1.1.1.8. Database menyusu

Database menyusu aşağıdakı əmrlərdən ibarətdir.

- **Explorer - Database Explorer** və ya **SQL Explorer** (Delphi-nin versiyasından asılı olaraq) verilənlər bazası ilə işləmə alətlərini çağırır.
- **SQL Monitor - SQL Monitor** verilənlər bazasına sorğular alətini çağırır.
- **Form Wizard** - uzaq məsafəli və ya lokal verilənlər bazasından verilənlər yığımını təsvir edən forma yaratmaq üçün forma eksperti pəncərəsini çağırır.

1.1.1.9. Tools menyusu.

Tools menyusunun əmrləri aşağıdakılardır:

- **Environment Options** - Delphi mühiti parametrlərini və alətlərini nizamlamaq üçün pəncərəni çağırır.
- **Editor Options** - Delphi redaktoru parametrlərini nizamlamaq üçün pəncərəni çağırır.
- **Debugger Options** - Delphi sazlayıcı parametrlərini nizamlamaq üçün pəncərəni çağırır.
- **Repozitory** - Delphi obyektlər saxlancının idarəsi pəncərəsini göstərir.
- **Translation Repozitory**-obyektlərin translyasiya saxlancı pəncərəsini göstərir.
- **External Editor**-xarici redaktor seçməyə imkan verir.
- **Web App Debugger** – **web** - əlavələr üçün sazlayıcı seçməyə imkan verir.
- **Regemrate CORBA IDL Files - CORBA** texnologiyası ilə maşınlar arası qarşılıqlı əlaqə yaratmaq üçün yenidən **IDL** faylı yaradır.
- **Configure Tools - Tools** menyusunun redaktəetmə dialoq pəncərəsini göstərir.
- **Database Desktop - Database Desktop** verilənlər bazasının xidmət alətlərini çagırır.
- **Image Editor**-qrafiki redaktor pəncərəsini çağırır.
- **Package Collection Editor**-paketlər yığımının redaktorunu çağırır.
- **XML Mapper**- verilənləri **XML** faylları şəklində təsvir etmək üçün utilitləri çağırır.

1.1.1.10. Help menyusu

Help menyusunun aşağıdakı əmrləri vardır:

- **Delphi Help** - Delphi-nin əsas məlumat xidmətidir.
- **Delphi Tools** - Delphi-nin alətləri haqqında məlumat xidmətidir.
- **Windows SDK - Windows API** üzrə məlumat xidmətidir.
- **Borland Home Page -** Borland kompaniyasının şəxsi səhifəsi.
- **Delphi Home Page** - Delphi-nin şəxsi səhifəsi.
- **Borland Developer Support** – istifadəçilərin işinə kömək səhifəsi .
- **Delphi Direct – istifadəçiyə INTERNET kömək pənsərəsi.**

🖶 **Customize - Open Help** xidmətçisinin çağırılması.
🖶 **About** - Delphi haqqında qısa məlumat pəncərəsi.

1.1.2. Piktoqrafik düymələr

Piktoqrafik düymələr əsas menyunun ən vacib rejimlərinə cəld keçməyi təmin edir. Aşağıdakı cədvəldə standart piktoqrafik düymələrin məcmusu ilə yerinə yetirilən əmrlər göstərilmişdir.

Düymə	Düymə ilə reallaşdırılan əməliyyatlar
	Standard qrupu
	Obyektlər saxlancını açmağa imkan verir. File ▶ New ▶ Other rejimlərinə ekvivalendir.
	Mövcud faylı açır. File ▶ Open File rejiminə ekvivalentdir.
	Faylı disk yaddaşında saxlayır. File ▶ Save File rejiminə ekvivalentdir. (Ctrl-S cəld müraciət klavişləri).
	Proyektin bütün fayllarını yaddaşda saxlayır. File ▶ Save All rejiminə ekvivalentdir.
	Mövcud proqram proyektini açmağa imkan verir. File ▶ Open Project rejiminə ekvivalentdir. (Ctrl-F11 cəld müraciət klavişləri).
	Yeni faylı proyektə əlavə edir. Project ▶Add to Project rejiminə ekvivalentdir. (Shift-F11 cəld müraciət klavişləri).
	Faylı proyektdən silir Project ▶Remove from Project rejiminə ekvivalentdir.

Project View qrupu.

 Cari proyektlə əlaqəsi olan modulları modullar siyahısından seçir. View ▶Units rejiminə ekvivalentdir. (Shift F12 cəld müraciət klavişləri).

 Cari proyektlə əlaqəsi olan formaları formalar siyahısından seçir. View ▶Forms rejiminə ekvivalentdir. (Ctrl-F12 cəld müraciət klavişləri).

 Forma pəncərəsi ilə proqramın kod pəncərəsi arasında fəallığı təmin edir. View ▶Toggle Form və View ▶Toggle Unit rejiminə ekvivalentdir. (F12 cəld müraciət klavişi).

 Yeni forma yaradır və onu proyektə əlavə edir File▶New▶Form rejiminə ekvivalentdir.

Debug qrupu

 Proqramı kompilyasiya və icra edir. Run▶Run rejiminə ekvivalentdir.(F9 cəld müraciət klavişi).

 Sazlanan proqramın işində fasiləni reallaşdırır. Run/Program Pause rejiminə ekvivalentdir.

 Alt proqramları izləməklə proqramın addımbaaddım trassirovkasını həyata keçirir. Run▶Trace Into rejiminə ekvivalentdir. (F7 cəld müraciət klavişi).

 Alt proqramları izləməklə proqramın addımbaaddım trassirovkasını həyata keçirir. Run▶Step Over rejiminə ekvivalentdir. (F8 cəld müraciət klavişi).

Custome qrupu

 Məlumat xidmətinə müraciət etməyə imkan verir Help▶Delphi Help rejiminə ekvivalentdir.

Desktops qrupu

Delphi-nin bütün pəncərələrinin sazlanmasının mümkün variantlarının seçilməsi siyahısı.

 Delphi-nin pəncərələrinin cari sazlanmasını yadda saxlayır.

 Sazlama rejiminə uyğun pəncərənin tənzimlənməsini seçir.

INTERNET qrupu

 **WebSnap** texnologiyasına uyğun yeni əlavənin yaradılmasına başlayır.

 **WebSnap** əlavəsində yeni səhifə yaradır.

 **WebSnap** əlavəsində yeni verilənlər modulu yaradır.

 Piktoqrafik düymələrin məcmuunu asanlıqla dəyişdirmək olar, nadir hallarda istifadə olunanları ləğv etmək və ya yenilərini əlavə etmək olar. Düymələrin

məcmuunu dəyişdirmək üçün onlardan ixtiyari birinin üzərində mausun sağ düyməsini vuraraq , açılacaq köməkçi menyudan **Customize** (xüsusiləşdir) rejimini seçsək, sağ ekranda düymələrin redaktoru pəncərəsi açılacaq. (şəkil 1.5).

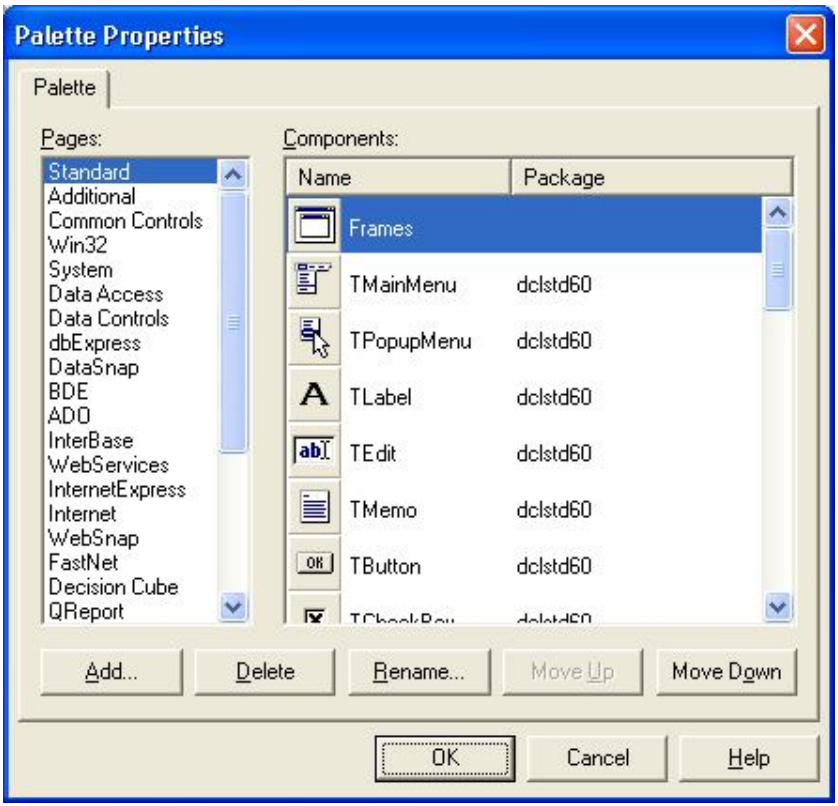
Şəkil 1.5 Düymələr redaktorunun pəncərəsi

Bu pəncərə bağlanana qədər buraya yeni düymə əlavə və ləğv etmək olar. Düyməni ləğv etmək üçün mausu onun üzərinə gətirib mausun sol düyməsini basılı vəziyyətdə saxlayaraq düyməni əsas pəncərədən kənara «sürükləmək» lazımdır. Panelə yeni düymə əlavə etmək üçün onu düymələrin redaktoru pəncərəsindən seçib, lazım olan yerə «daşımaq» lazımdır. Əgər düymənin yerləşməsi üçün yer çatmırsa, panelin ölçülərini artırmaq mümkündür. Bunun üçün mausun itiuclu göstəricisini dümələr panelini komponentlər palitrasından ayıran şaquli hissəyə gətirib, göstərici ikitərəfli oxa çevrildikdən sonra ayırıcı hissəni yeni yerə «daşımaq» lazımdır.

1.1.3. Komponentlər palitrası

Komponentlər palitrası -Delphi-nin ən böyük zənginliyidir. O, əsas pəncərənin sağ tərəfində yerləşib, lazımi komponentin tez tapılması üçün nişana malikdir. Komponent dedikdə forma pəncərəsində yerləşdirilən müəyyən xassəyə malik hər hansı funksional element

nəzərdə tutulur. Komponentlərin köməyi ilə proqramın karkası yaradılır. Hər bir halda onun ekranda görünən: pəncərə, düymə, seçmə siyahısı və s. düymələr panelində olduğu kimi komponentlər palitrasında da saxlana bilər. Bunun üçün mausun sağ düyməsini palitranın istənilən komponentinin üzərində bir dəfə vurduqda ekrana çıxan xüsusi redaktordan istifadə edilir. (Şəkil 1.6).



Şəkil 1.6 Komponentlər palitrası redaktoru pəncərəsi.

1.2. Forma pəncərəsi

Forma pəncərəsi gələcək proqramın proyekti üçün Windows pəncərədən ibarətdir. Başlanğıcda bu pəncərə boş olur. Proqramçının işinin əsas hissəsi komponentlər palitrasından lazımi komponentləri götürüb forma pəncərəsində yerləşdirməkdən ibarətdir. Forma pəncərəsinin doldurulması Vizual proqramlaşdırmanın əsas elementlərindən biridir. Proqramçı istənilən anda yaradılacaq proqram pəncərəsinə nəzarət və zəruri dəyişiklikləri edə bilir.

1.3. Obyektlər inspektoru pəncərəsi

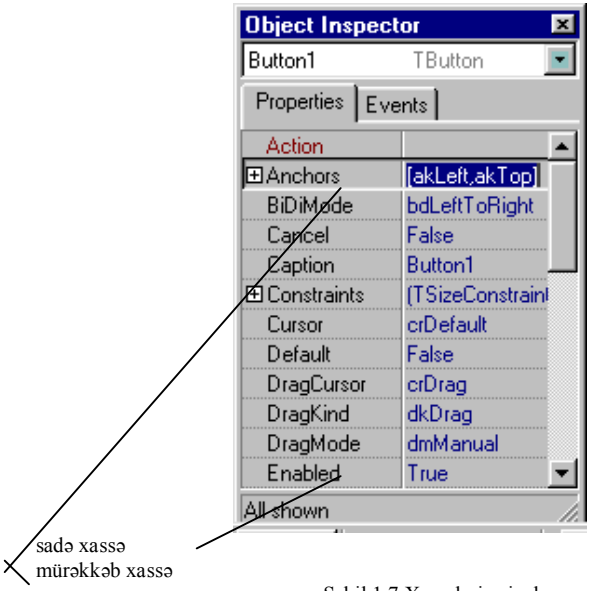
Formada yerləşdirilən istənilən komponent müəyyən parametrlərin məcmuu ilə xarakterizə olunur: vəziyyəti ölçüsü, rəngi və s.

Bu parametrlərdən bəzilərini, komponentin vəziyyəti və ölçülərini proqramçı forma pəncərəsinin özündə komponentin üzərində manipulyasiya edərək dəyişdirə bilər. Digər parametrlərin dəyişdirilməsi üçün Obyektlər Inspektoru pəncərəsi nəzərdə tutulmuşdur. Bu pəncərə iki səhifədən – Properties (xassə) və Events (hadisə) ibarətdir. Properties səhifəsi komponentə lazımi xassənin verilməsinə, Events səhifəsi isə komponentin bu və ya digər hadisəyə reaksiyasını təyin edir. Xassələrin məcmuu komponentin görünüşünü təyin edir: formanın işçi sahəsində yuxarı sol küncə görə vəziyyətini, onun ölçü və rəngini, üzərində olan mətni , şrifti və s. Hadisələrin məcmuu komponentin özünü necə aparmasını: komponentin mausun və ya klaviaturanın düyməsinin basılması zamanı ona reaksiyası və s. göstərir.

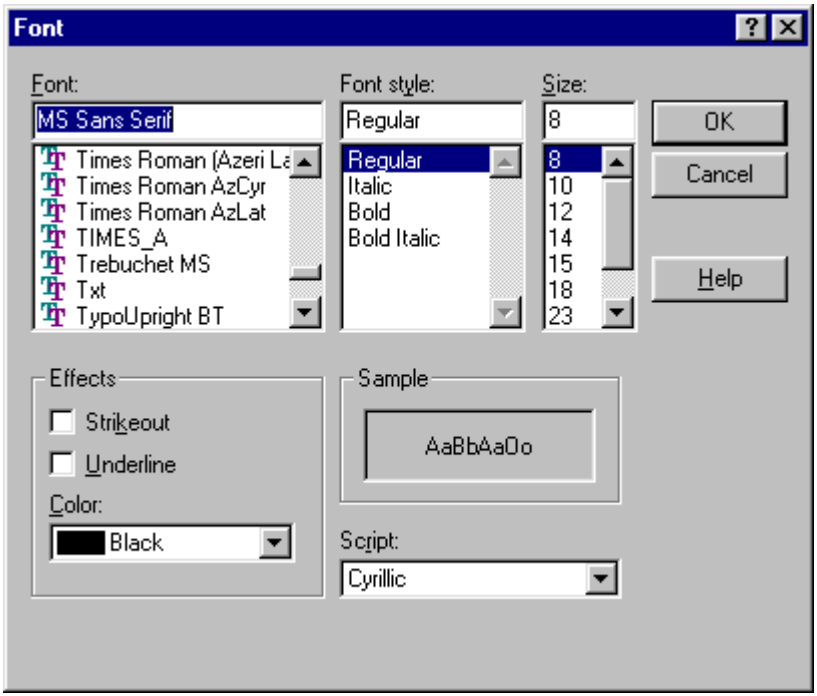
Obyektlər inspektorunun hər bir səhifəsi ikisütünlu cədvəldən ibarətdir. Sol sütun xassə və ya hadisənin adından ibarətdir. Sağ sütun isə xassənin konkret qiyməti və ya uygun hadisəni emal edəcək alt proqramlardan ibarətdir.

Cədvəlin sətri mausun düyməsinin vurulması ilə seçilir və sadə və ya mürəkkəb xassələri təsvir edə bilər. Sadə xassələrə yeganə ədədlə, işarələr sətri ilə, true(doğru) və ya false(yalan) məntiqi qiymətləri ilə və s. təyin olunan xassələr aiddir.

Məsələn, Caption(Başlıq) xassəsi işarələr sətri ilə, Height (hündürlük) və Width(en)-ədədlərlə, Enabled(mümkünlük) true və false qiymətləri ilə təyin olunur. Mürəkkəb xassələr müəyyən qiymətlərin məcmuu ilə təyin olunur.Bu cür xassələrin sol tərəfində «+» işarəsi qoyulur. Mausun sol düyməsini «+» işarəsinin üzərində vurmaqla mürəkkəb xassənin təşkiledicilərini ekrana çıxartmaq olar. Açılmış siyahını baglamaq üçün mausun sol düyməsini « – » işarəsinin üzərində vurmaq lazımdır.



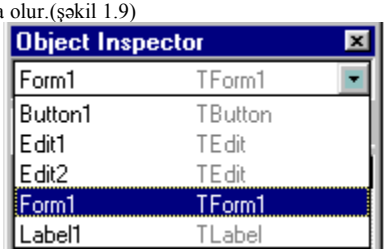
Şəkil 1.7 Xassələrin siyahısı



Şəkil 1.8 Font mürəkkəb xassəsini təşkil edən siyahının açılması

Cədvəlin sağ sütunundakı sətirdə xassənin qiyməti üzərində mausun sol düyməsnin vurulması orada göstərilən qiyməti cari etməklə onun sağ tərəfində ... və ya ▼ düymələrindən birini çıxarır, ... düyməsi üzərində mausun sol düyməsini vurduqda mürəkkəb xassənin qiymətini daxil etmək üçün dialoq ekran pəncərəsi açılacaq.(şəkil 1.8) ▼ düyməsi üzərində mausun sol düyməsinin vurulması mürəkkəb xassənin mümkün qiymətlərini ekrana çıxarır.

Obyektlər inspektoru pəncərəsinin yuxarı hissəsində forma üzərində yerləşdirilən bütün komponentlərin siyahısı verilir. Forma özü də komponent olduğundan onun adı da həmin siyahıda olur.(şəkil 1.9)



Şəkil 1.9 Forma üzərində yerləşdirilən obyektlərin siyahısı

1.4. Proqram kodu pəncərəsi

Kod pəncərəsi proqram mətninin yaradılması və redaktə olunması üçün nəzərdə tutulmuşdur. Bu mətn proqram alqoritminin işini təsvir edərək, xüsusi qaydada tərtib olunur. Mətnin yazılması qaydalarının məcmuu proqramlaşdırma dili adlanır. Delphi sistemində geniş yayılmış Borland korporasiyasının (Turbo Pascal və Borland Pascal) Pascal dilinin təkminləşdirilmiş və genişləndirilmiş versiyası olan Object Pascal dilindən istifadə olunur. Delphi vizual mühiti proqramlaşdırmanın bir sıra istiqamətlərini öz üzərinə götürməsinə baxmayaraq bu dildə proqram yazmaq bacarığı proqramçı üçün ən vacib şərtdir.

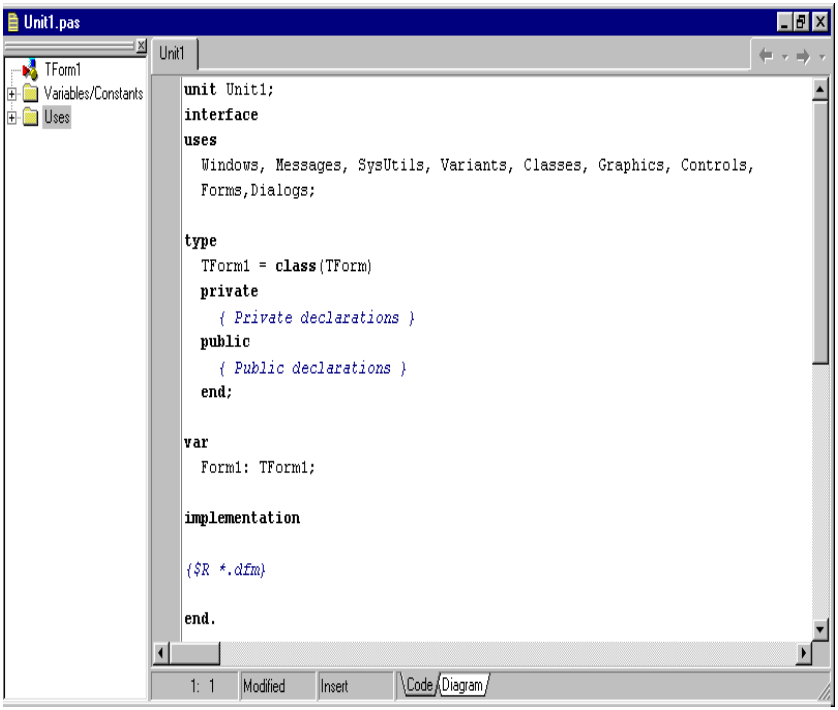
Başlanğıcda kod pəncərəsi boş forma pəncərəsinin Windows-pəncərə kimi sərbəst fəaliyyət göstərməsini təmin edən minimal ilkin mətndən ibarətdir (şəkil 1.10).Işin gedişində proqramçı proqrama lazımi funksionallıq vermək üçün proyekt üzərində zəruri dəyişikliklər aparır.

Bu sətirləri yeni forma üçün Delphi avtomatik olaraq kod pəncərəsinə qoyur. Kod pəncərəsi və forma pəncərəsi bir-biri ilə möhkəm əlaqəli olub, Delphi onların hər ikisi üzərində “nəzarət” funksiyasını həyata keçirir.

Hələlik biz bu mətn hissəsini dəyişməyəcəyik. Gələcəkdə biz lazımi sətirləri

```
{$R  *DFM}
…
end.
```

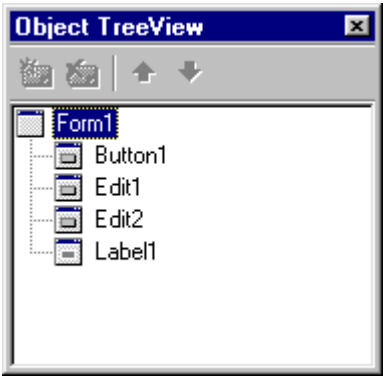
operatorları arasında yerləşdirəcəyik.



Şəkil 1.10 Proqram kodu pəncərəsi.

1.5. Ağacvari obyektlər pəncərəsi

Bu pəncərə 6-cı versiyada meydana çıxmış, fəallaşdırılmış cari modulda ayrı-ayrı komponentlər arasında əlaqəni əyani təsvir etmək üçün nəzərdə tutulmuşdur.(şəkil 1.11)

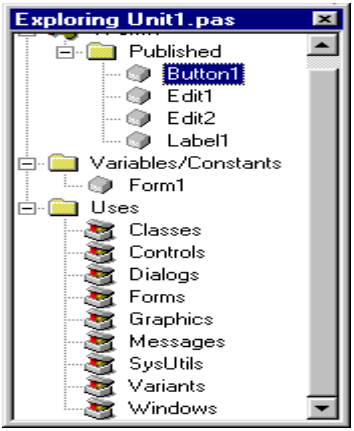


Şəkil 1.11 Ağacvari obyektlər pəncərəsi

Bu pəncərədə ixtiyari komponent üzərində mausun sol düyməsini sıxsaq, forma pəncərəsinin uyğun komponenti fəalaşdırılır və onun bütün xassələri obyektlər inspektoru pəncərəsində təsvir olunur. Sol düymənin iki dəfə vurulması Code Insight mexanizminin işə düşməsinə səbəb olur və hadisənin emaledicisi üçün hazır kod pəncərəsini açır.

1.6. Brauzer pəncərəsi

Bir qayda olaraq kod pəncərəsi ilə eyni vaxtda brauzer pəncərəsi də (Code Explorer) fəalaşdırılır. Brauzer pəncərəsi kod pəncərəsində sətirlərin sayı çox olduqda lazımi elementin tapılmasını asanlaşdırır. Brauzer pəncərəsində element üzərində mausun sol düyməsini iki dəfə vurduqda mətn göstəricisi kod pəncərəsində uyğun elementin birinci rast gəlinənin üzərinə gətirilir.



Şəkil 1.12 Obyektlər brauzeri pəncərəsi

Kiçik həcmli proqramlar işləyərkən brauzer pəncərəsini pəncərənin yuxarı sağ küncündə olan  düyməsi ilə baglamaq olar. Baglanmış pəncərəni yenidən bərpa etmək üçün kod

pəncərəsində mausun sag düyməsini vurduqdan sonra View Explorer rejimini seçmək lazımdır. Qeyd edək ki, brauzer pəncərəsi köməkçi paneldə yerləşən yuxarı hissədə idarəedici aralığa malikdir ki, onun köməyi ilə pəncərəni lazım gələn yerə «sürükləmək » olar.

2. Redaktorla iş

Redaktorun bütün əmrlərini kursorun hərəkəti, silmək və yapışdırmaq bloklarla işləmə digər əmrlərə bölmək olar.

2.1. Kursorun hərəkəti əmrləri

Düymə	Keçid	Düymə	Keçid
←	bir işarə sola	Home	sətrin əvvəlinə
→	bir işarə sağa	End	sətrin sonuna
Ctrl ←	bir söz sola	Ctrl+PgUp	ekranın əvvəlinə
Ctrl →	bir söz sağa	Ctrl+PgDn	ekranın sonuna
↑	bir sətir yuxarı	Ctrl+Home	faylın əvvəlinə
↓	bir sətir aşağı	Ctrl+End	faylın sonuna
PgUp	bir səhifə yuxarı	Ctrl+Q+B	blokun əvvəlinə
PgDn	bir səhifə aşağı	Ctrl+Q+K	blokun sonuna

2.2. Sil və yapışdır əmrləri

Düymə	təsviri
Delete	kursordan sağdakı işarəni silmək
Enter	sətir yapışdırmaq
Ctrl+Y	sətir silmək
Backspace	kursordan soldakı işarəni silmək
Ins	işarə əlavə etmək
Ctrl+T	kursordan sağdakı sözü silmək
Ctrl+Q+Y	sətrin kursordan sağda qalan hissəsini silmək
Ctrl+Z	mətnin axırınjı dəyişməsini ləğv etmək

2.3. Bloklarla işləmə əmrləri

Proqram mətni yazılarkən müəyyən mətn fraqmentini bir yerdən başqa yerə köçürmək və ya silmək lazım gəlir. Bu növ əməliyyatları yerinə yetirmək üçün tam bir vahid kimi baxılan mətn fraqmenti-blokdan istifadə etmək əlverişlidir. Blokun uzunluğu kifayət qədər böyük ola bilər. Hər bir anda redaktor pəncərəsində yalnız bir blok təyin oluna bilər. Bloklar arası mübadilə bilavasitə mübadilə buferi vasitəsi ilə həyata keçirilə bilər. Bloklarla işləmə klavişləri aşağıdakı cədvəldə verilmişdir.

Klaviş	Təsviri
Ctrl+K+T	Kursordan soldakı sözü bloka alır.
Ctrl+K+P	Bloku çap edir.
Ctrl+K+H	Blokdan rəngi götürür;

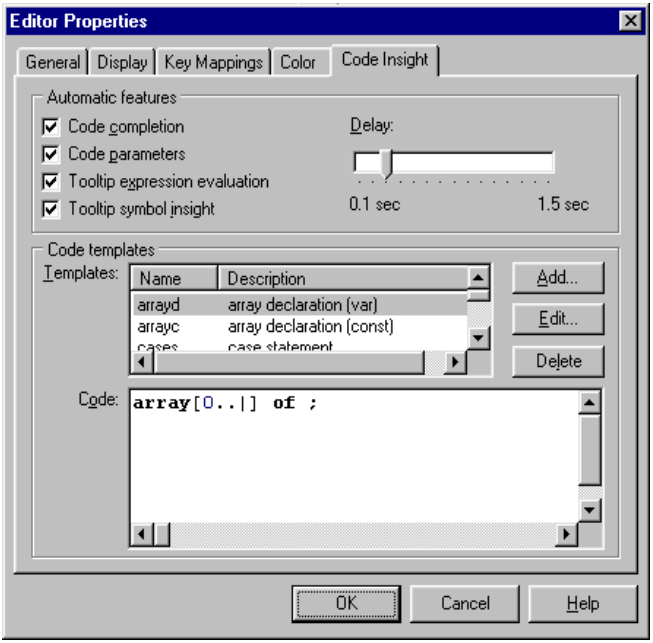
	təkrar basıldıqda bloku bərpa edir.
Ctrl+K+Y	bloku silir
Ctrl+K+W	bloku diskə yazır
Shift+Delete	bloku kəsib götürərək, mübadilə buferinə yerləşdirir
Ctrl+Insert	bloku mübadilə buferinə köçürür
Shift+Insert	bloku mübadilə buferindən götürərək,kursorla müəyyən olunan yerə yapışdırır.

2.4. Digər əmrlər

Klaviş	Təsvir
Ctrl+F	Nümunəyə görə axtarmaq
F3	Axtarışı davam etdirmək
Ctrl+R	Nümunəyə görə axtarmaq və əvəz etmək
Ctrl+K+n	Marker yerləşdirmək; n=0-9
Ctrl+Q+n	Markeri axtarmaq
Ctrl+Q+]	Mötərizələr cütünü axtarmaq
Ctrl+O+O	Kompiyatorun parametrlərini faylın başlanğıcına yapışdırmaq
Ctrl+Shift +R	Makrosun təyin olunmasını başlayır və sona çatdırır.
Ctrl+Shift +P	Əvvəlcədən təyin olunmuş makrosu yerinə yetirir.

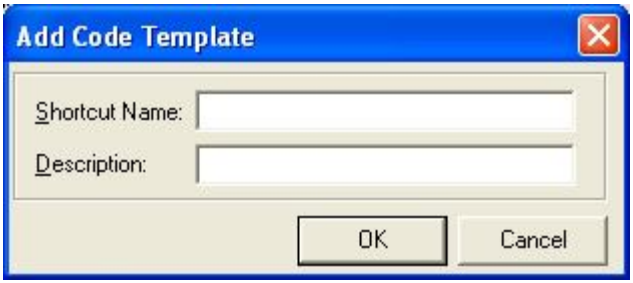
2.5. Mətn fraqmentlərinin qoyulması və Code Insight

Redaktor mətnə çoxlu sayda mətn fraqmentlərini qoymağa imkan verir ki, bu da proqram kodunun daxil edilməsi vaxtını qısaltmağa imkan verir. Hazır mətn fraqmentlərinə baxmaq və oraya yenisini əlavə etmək üçün Tools▶Editor Options komandası ilə açılan Editor Properties pəncərəsinin Code Insight nişanından istifadə etmək olar.



Şəkil 2.1 Parametrlərin sazlanması pəncərəsinin Code Insight nişanı.

Templates siyahısında nümunənin adını və qısa təsvirini vermək olar. Code sahəsində isə mətn fraqmentini görmək və redaktə etmək olar. Add düyməsi yeni nümunə daxil etməyə, Edit düyməsi onun adını və qısa təsvirini dəyişməyə, Delete düyməsi isə ləğv etməyə imkan verir.



Yeni mətn fraqmenti qoyarkən və ya mövcud mətn fraqmentini redaktə edərkən nəzərə almaq lazımdır ki, |(Şaquli xətt) mətn fraqmentinin proqram mətninə yerləşdiriləcəyi yeri göstərir. Hazırlamış mətn fraqmentini kod pəncərəsinə qoymaq üçün onun adını yazıb Ctrl+J düymələrini sıxmaq lazımdır.

Bu zaman adın yerinə mətn fraqmenti yazılacaqdır. Əgər bütün mətn fraqmentlərinin adlarını yadda saxlamaq çətinlik törədirsə, onu siyahıdan seçmək olar. Bunun

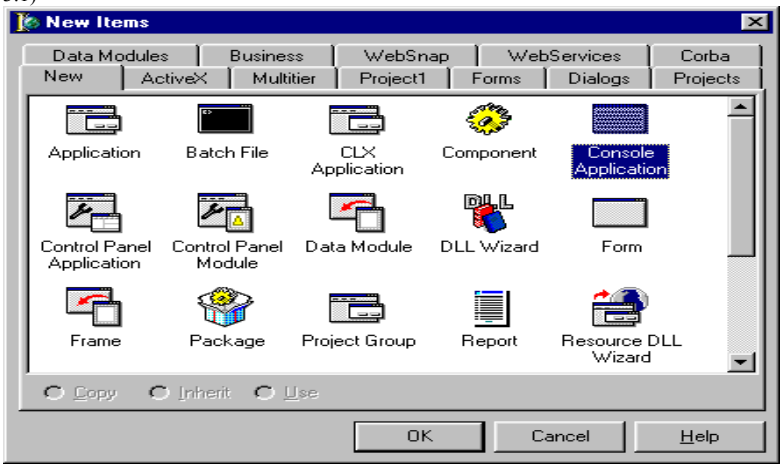
üçün mətn kursorunu fraqment qoymaq istədiyimiz yerə gətirib Ctrl +J düymələri basılsa, bütün fraqmentlərin adları və qısa təsvirlərindən ibarət pəncərə ekrana çıxacaq.

3. Sadə proqramların yaradılması

3.1. Konsol əlavələri

Delphi 6 sisteminin köməyi ilə qrafiki interfeysdən istifadə edərək qeyri-məhdud mürəkkəblikli Windows əlavələri yaratmaq olar. Pascal-ın operatorları ilə yeni işləməyə başlayanlar üçün MS-DOS üslubunda sadə proqramlar yazmaq olar. Bunlar konsol əlavələri adlanır.

Konsol əlavələri yaratmaq üçün File ▶New ▶Other komandalarını vermək və açılan New Items dialoq pəncərəsində Console Application nişanını seçmək lazımdır.(Şəkil 3.1)



Şəkil 3.1 Yaradılacaq proqramın növünün seçilməsi.

Delphi 6 sistemi avtomatik olaraq gələcək əlavənin yaradılması üçün hazır proqram mətnini verir.

```
program Project1;

{$APPTYPE CONSOLE}

uses
  SysUtils;
begin
  { TODO -oUser -cConsole Main : Insert code here }
end.
```

Proqram kodu **program** açar sözündən ibarət proqram başlığı ilə başlayır və avtomatik olaraq Project1 adını generasiya edir.

Sonra kommentariya şəklində kompilyatorun direktivi gəlir.Direktiv adi kommentariyadan { işarəsindən sonra $ işarəsinin gəlməsi ilə fərqlənir.

{$APPTYPE CONSOLE} direktivi kompilyatora verilmiş proqramın konsol əlavəsi olması barədə məlumat verir.

Sonrakı sətir SysUtils standart modulunun uses açar sözünün köməyi ilə proqrama qoşulduğunu bildirir.

Bundan sonra isə proqram kodu yazılır.

3.2. Informasiya mübadiləsi

Yaradılacaq proqram konsol əlavəsi olduğundan istifadəçi ilə qrafiki interfeysin köməyi ilə əlaqə mümkün olmur.Istifadəçi ilə informasiya mübadiləsini təmin etmək üçün daha sadə vasitələr tələb olunur.

Pascal-da konsol əlavələrində istifadə etmək üçün iki standart prosedura Readln(verilənləri daxil etmək üçün) və Writeln(verilənlərin yazılması üçün) vardır.Birinci prosedura qiymətləri klaviaturadan daxil etməklə parametr kimi götürülən dəyişənə(dəyişənlər siyahısı götürülərsə ,daxil edilən qiymətlər bir-birindən aralıq işarəsi ilə ayrılırlar) mənimsədir.

Ikinci prosedura dəyişənlərin və ya ifadələrin bir və ya bir neçə qiymətlərini ekranın cari sətrinə aralıq işarəsi ilə ayırmaqla çıxarır.

Readln prosedurası daxiletmə başa çatdıqdan sonra Enter düyməsinin basılmasını,Writeln prosedurası isə verilənlərin yazılması başa çatdıqdan sonra kursoru növbəti sətrə keçirir.Digər Write prosedurası növbəti sətrə keçməni təmin etmir,verlənləri cari sətrə çıxarmağa davam edir.

Readln prosedurunun heç bir parametri olmaya da bilər.

Bu zaman operator yerinə yetirilərkən proqram istifadəçinin Enter düyməsini basmasını gözləyir.Əgər Writeln prosedurasında heç bir parametr göstərilməmişdirsə , onda ekranda boş birsətir ötürüləcək.

Proqramın sınaq üçün ilk dəfə işləməsi üçün əsas məntiqi bloka yeganə **Readln** operatoru yerləşdirək.Proqramı işə salarkən qara rəngli konsol pəncərəsi açılacaq və **Enter** düyməsi basıldıqdan sonra bağlanacaqdır.

```
Begin
  Readln
  End.
```

Proqramın yadda saxlanması üçün alətlər panelində  (Save all) düyməsini sıxmaq lazımdır.Bu zaman Delphi6 sistemi Project1 faylının yadda saxlanması üçün kataloqu soruşacaqdır.

Proqramı kompilyasiya edib işə salmaq üçün F9 düyməsini basmaq kifayətdir.Proqramın ilkin mətni yerləşən kataloqda yerinə yetirilməyə hazır Project1.exe faylı yaranacaqdır.Həmin fayl Delphi6 mühiti tərəfindən həmin anda işə salınacaqdır.Ekranda konsol əlavəsinin pəncərəsi açılacaqdır. **Enter düyməsini basdıqdan sonra idarə yenidən** Delphi6 mühitinə ötürüləcək.

Delphi6 və Pascal-ın imkanlarını praktiki olaraq nümayiş etdirmək üçün klaviaturadan iki ədədin daxil edilməsi və onların cəmini hesablayan proqram tərtib edək.

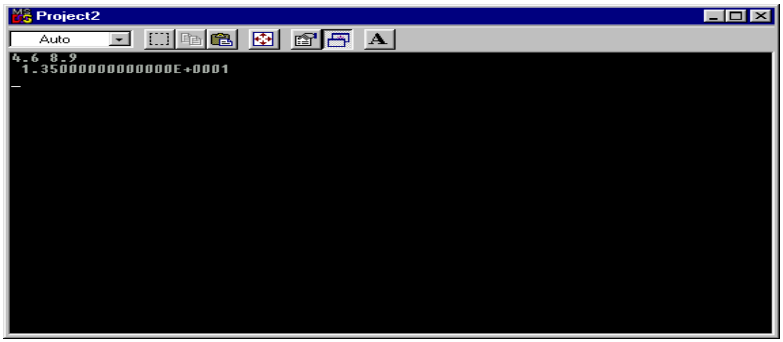
Proqramın ilkin mətni aşağıdakı kimi olacaqdır:

```
program Project1;
```

```
{$APPTYPE CONSOLE}

uses
  SysUtils;
var
   x,y,z:real;
begin
  readln(x,y);
  z:=x+y;
  writeln(z);
  readln
end.
```

F9 düyməsini basdıqdan sonra ekrana yaradılmış proqramın pəncərəsi çıxacaqdır.Həmin pəncərədən iki ədədi ,məsələn

4.6 5.9 daxil edərək Enter düyməsini vursaq ,nəticə ekrana çıxacaqdır və öz işini yekunlaşdırmaq üçün Enter düyməsinin basılmasını gözləyəcəkdir.(Şəkil 3.2)



Şəkil 3.2 Yaradılmış konsol əlavəsi iş zamanı

4. Delphi mesaj pəncərələri

Delphi istifadəçiyə hər hansı bir məlumatı çatdırmaq,müəyyən işlərin görülməsi zamanı istifadəçidən göstəriş almaq,bəzi məsələlərin həlli zamanı istifadəçinin diqqətini cəlb etmək üçün mesaj pəncərələri açır.Bunların bir hissəsi Delphi tərəfindən,digər hissəsi isə Windows tərəfindən açılan standart mesaj pəncərələridir.

4.1.ShowMessage proseduru

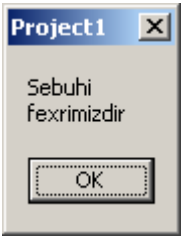
ShowMessage proseduru istifadəçiyə hər hansı bir anda məlumat vermək üçün istifadə olunur.

```
ShowMessage(Mesaj);
```

ShowMessage proseduru sadəcə Ok düyməsini daxilinə alan bir mesaj pəncərəsi çıxarır.

Bir sətirdən artıq mesaj pəncərəsi açmaq üçün mətnlər arasında #13 işarələri qoyulur (#13 işarəsi Enter düyməsinə uyğundur).

ShowMessage('Sebuhi'#13'fexrimizdir');



4.2. ShowMessagePos proseduru

Formatı: ShowMessagePos(Mesaj,x,y);

ShowMessagePos prosedurunun ShowMessage prosedurundan fərqi mesaj pəncərəsinin ekranın ortasında deyil, verilən x,y koordinatlarında görünməsidir.

Proqramdan çıxarkən Salamat qalin sözlərini ekrana çıxaran bir mesaj pəncərəsi düzəldək.Proqramdan çıxarkən OnClose hadisəsi meydana çıxdığı üçün proqram kodunu bu hadisənin emaledicisinə yazaq

```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
ShowMessagepos('Salamat qalin',100,100);
end;
end.
```

4.3. InputBox() funksiyası

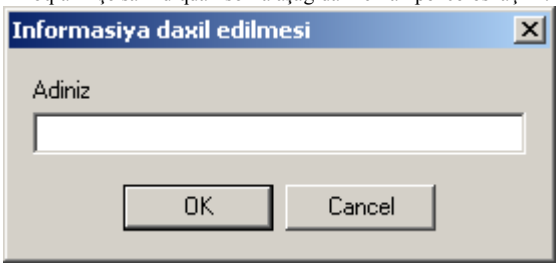
InputBox() funksiyası Delphi proyektinin aktiv forma və ya pəncərəsindən müstəqil bir dialoq qutusu içərisindən istifadəçinin verilənləri daxil etməsinə imkan verir.Bu funksiya kənardan üç parametr alır.

InputBox(Dialoq qutusu başlığı,Mətn qutusu başlığı,Daxil edilən qiymət)

Ilk parametr ilə dialoq qutusunun başlığı təyin edilir.Ikinci parametr ilə verilənin daxil edilməsi üçün dialoq qutusunda hazırlanan mətn qutusunun başlığı ,üçüncü parametr ilə isə mətn qutusuna daxil etmək istədiyimiz qiymət verilir.

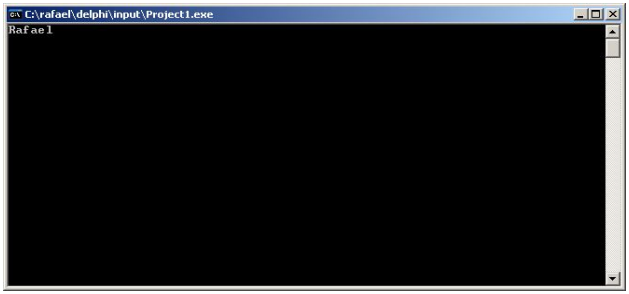
```
    program Project2;

{$APPTYPE CONSOLE}

uses
  SysUtils, dialogs;
var
   ad:string;
begin
ad:=inputbox('Informasiya daxil edilmesi','Adiniz','');
showmessage(ad);
writeln(ad);
readln;
  { TODO -oUser -cConsole Main : Insert code here }
end.
```

Proqram işə salındıqdan sonra aşağıdakı ekran pəncərəsi açılır.



Adiniz səhifəsindən adı daxil edib,OK düyməsini sıxdıqda aşağıdakı mesaj pəncərəsi açılır.

OK düyməsi üzərində mausun sol düyməsini vurduqda aşağıdakı ekran pəncərəsi görüntüyə gəlir.



4.4. MessageBox ( ) funksiyası

MessageBox ( ) funksiyası vasitəsi ilə həm ekrana mesaj vermək və həm də istifadəçiyə seçim imkanı verilir.

Application obyekti TApplication sinfinə məxsus olmaqla proqramın ən vacib xassə və metodlarını özündə cəmləşdirir. Application obyekti komponentlər palitrasında olmadığından onun xassələrinə layihələndirmə mərhələsində müraciət mümkün deyildir. Proqram yerinə yetirilərkən onun xassələrinə müraciət mümkündür.

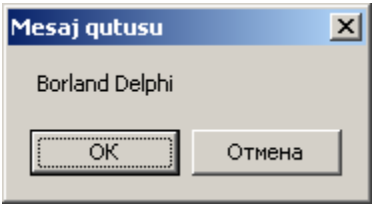
Application obyekti proqram və Windows əməliyyat sistemi arasında vasitəçi rolunu oynayır. Xüsusi halda onun köməyi ilə kontekst informasiya xidməti reallaşdırılır, müstəsna hallar emal edilir və s.

MessageBox ( ) funksiyası haqqında məlumat vermək üçün Fili▶New▶Application əmrlərini verməklə yeni bir fayl yaradaq. Proqram işləyərkən forma üzərində mausun düyməsi vurularkən MessageBox funksiyasının işləməsini istəyiriksə, proyekyin Unit1.pas moduluna FormClick adında proqram yazaq.

```
Procedure Tform1.FormClick(Sender:TObject);
Begin
  Application. MessageBox('Borland Delphi',
'Mesaj qutusu',mb_OKCancel+mb_DefButton1);
End;
```

MessageBox funksiyası kənardan 3 parametr alır. Birinci parametr mesaj qutusuna yazılacaq mesajı təyin edir. İkinci parametr ilə diaıoq qutusunun başlığı verilir. Dialoq qutusuna qoyulacaq düymələr və onlardan hansının seçildiyi üçüncü parametrin köməyi ilə verilir.

Verilən misalda dialoq qutusuna gətiriləcək düymələr 'mb_OKCancel' parametri ilə təyin olunur. Ona görə də dialiq qutusuna OK və Otmena(Delphi-nin rus versiyası olduğuna görə) düymələri qoyulur. Üçüncü parametrin davamı olaraq yazılan 'mb_DefButton1' sabiti isə dialoq qutusu ekrana gəldikdə birinci düymənin seçili olmasını göstərir.

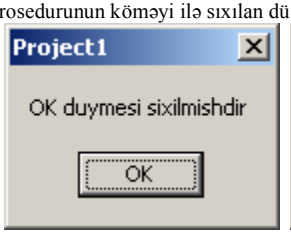


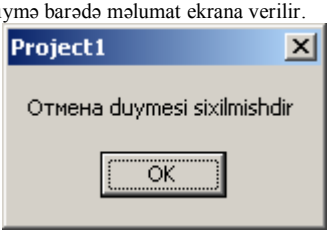
Dialoq qutusu ekranda olarkən OK düyməsi üzərində mausun düyməsi vurularkən MessageBox( ) funksiyası 1 qiymətini, Otmena düyməsi üzərində vurularkən isə 2 qiymətini qaytaracaqdır.

Proqram işləyərkən hansı düymənin seçildiyini öyrənmək üçün MessageBox( ) funksiyasının geriyə qaytardığı qiyməti bir dəyişənə mənimsədək və yuxarıdakı FormClick proqramında aşağıdakı dəyişikliyi aparaq.

```
Procedure Tform1.FormClick(Sender:Tobject)
   Var
      n: integer;
    begin
   n:= application. MessageBox('Borland Delphi',
    'Mesaj qutusu', mb_OKCancel+mb_DefButton1);
            If  n:=1 then
               ShowMessage('OK düyməsi sıxılıb');
            If  n:=2 then
               ShowMessage('Cancel düyməsi sıxılıb');
     End;
```

FormClick proqramı işlədikdən sonra OK düyməsi üzərində mausun sol düyməsini sıxsaq, MessageBox( ) mesaj pəncərəsi ekrandan götürülür, sıxılan düyməni təmsil edən ədədi qiymət (1) təsvir olunan dəyişənə (n-ə) mənimsədilir. Misala uyğun olaraq, if operatorunun və ShowMessage prosedurunun köməyi ilə sıxılan düymə barədə məlumat ekrana verilir.





MessageBox( ) finksiyası ilə ekrana gətirilən mesaj pəncərəsinə istənilən düyməni qoymaq olmaz.

Da, Net və Otmena düymələrini mesaj pəncərəsinə gətirmək üçün MessageBox funksiyasında üçüncü parametr olaraq, "YesNoCancel" sabitini veririk.

MessageBox( ) funksiyası ilə ekrana gətirilən mesaj pəncərəsində düymənin sıxılması funksiyanın müəyyən konkret tam qiyməti qaytarmasına gətirir. Bu tam qiymətlərin hər birinə aşağıdakı cədvəldə göstərilənlərə uyğun işarə sabiti var.

düymənin adı	ədədi qiymət	işarə sabiti

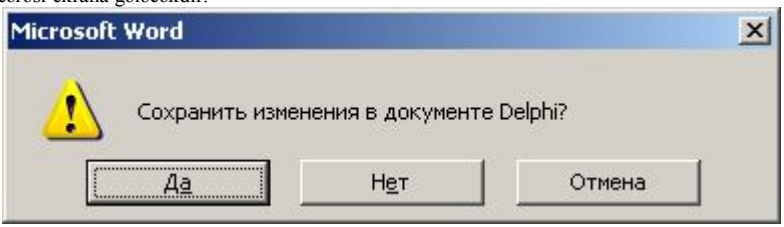
OK	1	İDOK
Cancel	2	İDCancel
Abort	3	İDAbort
Retry	4	İDRetry
İgnore	5	İDİgnore
Yes	6	İDYes
No	7	İDNo

Bu qiymətlərdən nə cür istifadə olunduğunu göstərmək üçün yuxarıda göstərdiyimiz MessageBox( ) funksiyasından istifadə proqramında aşağıdakı dəyişiklikləri aparaq.

```
procedure TForm1.FormClick(Sender: TObject);
var
  n:integer;
begin
n:=Application.MessageBox('Fayl diske yazilsinmi?',
'BorlandDelphi',mb_YesNoCancel+mb_defButton2);
if n=IDYes then
     ShowMessage('Da duymesi sixilmishdir');
if n=IDCancel then
     ShowMessage('Otmena duymesi sixilmishdir');
if n=IDNo then
ShowMessage('Net  duymesi sixilmishdir');
end;

end.
```

4.5. MessageDlg( ) funksiyası

MessageDlg( ) funksiyası ilə tanış olmamışdan əvvəl, Microsoft Word-də işləyərkən faylda axırıncı etdiyimiz dəyişikliyi diskə yazmadan pəncərəni bağlamaq istəsək, aşağıdakı dialoq pəncərəsi ekrana gələcəkdir.

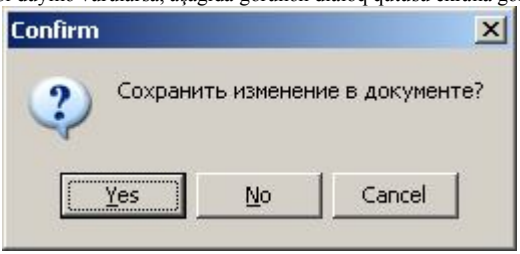


Bu dialoq pəncərəsinə diqqətlə baxdıqda sol tərəfdə  işarəsini görürük. Bunun mənası istifadəçiyə xəbərdarlıq olunduğu və seçim imkanı verilməsidir. Indi bu dialoq qutusunun bənzərini Delphi- də yaratmağa çalışaq.

Bu məqsədlə üzərində işlədiyimiz proyektdəki formaya aid Unit1.pas adlı modulun FormClick proqramını aşağıdakı kimi yazaq.

```
procedure TForm1.FormClick(Sender: TObject);
begin
 MessageDlg('Soxranitğ izmenenie v
dokumente?',mtConfirmation,[mbYES,mbNO,mbCancel],0);
    end;

    end.
```

Buradan görünür ki, MessageDlg( ) funksiyası kənardan 4 qiymət alır. Birinci parametrlə dialoq qutusuna yazılacaq mesaj verilir. Bu proqramı işlədikdən sonra forma üzərində sol düymə vurularsa, aşağıda görünən dialoq qutusu ekrana gələcəkdir.



MessageDlg( ) funksiyasında ikinci parametr ilə dialoq qutusunun başlığı və mesajın önünə qoyulan kiçik şəkil verilir.

MessageDlg( ) funksiyasına mtConfirmation sabit qiymətindən başqa mtİnformation, mtError, mtCustom və mtWarning sabit qiymətləri verilə bilər.

MessageDlg( ) funksiyasında üçüncü parametr ilə çoxluqlar konstruktoru şəklində dialoq qutusuna yerləşdiriləcək düymələr verilir.

- mbYes- Yes düyməsi (bu düyməyə vurularkən qaytarılan qiymət- mrYes)
- mbNo- No düyməsi (mrNo)
- mbCancel- Cancel düyməsi (mrCancel)
- mbHelp- Help düyməsi (bu düyməni sıxdıqda pəncərə bağlanmır)
- mbAbort- Abort düyməsi (mrAbort)
- mbİgnore- İgnore düyməsi (mrİfnore)
- mbAll- All düyməsi (mrAll).

MessageDlg( ) funksiyasında dördüncü parametrin tipi LongİInt olub, bu parametrin köməyi ilə yaradılan pəncərəni məlumat xidməti ilə əlaqələndirmək olar.

MessageBox ( ) funksiyası ilə yaradılacaq pəncərədən fərqli olaraq, MessageDlg( ) funksiyasında pəncərənin başlıq hissəsini dəyişmək olmaz.

5. Proqramın sazlanması

Səhvsiz proqram demək olar ki, olmur. Əmrlərin düzgün daxil edilməməsi, identifikatorların düzgün yazılmaması və digər qeyri-korrekt əməliyyatları ilkin mətni analiz etməklə aşkara çıxarmaq olar və Delphi 6-nın kompilyatoru bunu həmişə təyin edir.

Bəzi hallarda xəbərdarlıq və ya köməkçi məlumat çıxır. Onların mənbəyini aradan qaldırmaq məqsədəuyğundur, bununla birlikdə alqoritmin düzgün reallaşdırılmaması (məsələn, > işarəsinin əvəzinə < işarəsi yazılmışdırsa) proqram işləyərkən səhvin meydana çıxmasına səbəb ola bilər. Bundan başqa ilkin alqoritmin düzgün reallaşdırılmaması əlavənin işinin pozulmasına gətirmir, lakin qeyri-dəqiq nəticənin alınmasına və ya səhv əməliyyatın yerinə yetirilməsinə gətirir.

Seb massivinə qiymət mənimsədilməsi ilə bağlı aşağıdakı proqram fraqmentinə baxaq.

```
Var
    i,n:integer;
    Seb:array[1..10] of integer;
    n:=11;
    for i:=1 to n do
      Seb(i):=0;
```

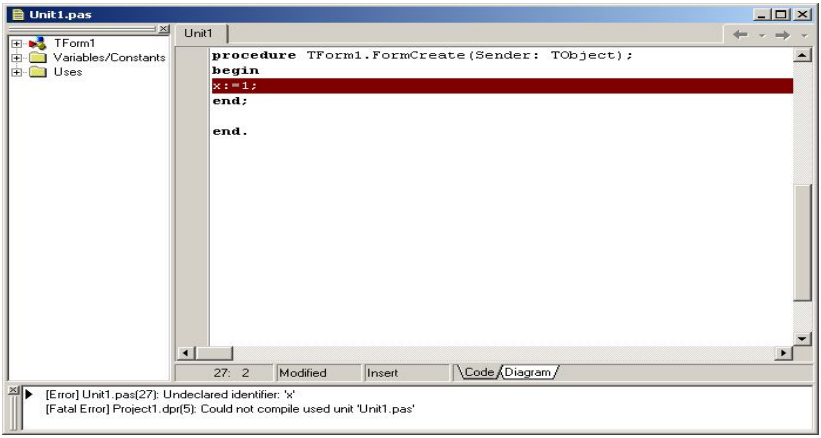
Bu proqram fraqmentində səhv vardır. Çünki dövrün sonunda massivin 11-ci elementinə müraciət vardır ki, bu da massivin elan olunmuş təsvirindəki ölçüsünü aşır və səhvə gətirir. Bu cür səhvləri kompilyator işləyib aşkara çıxara bilmir və bu cür səhvlərin tapılması və aradan qaldırılması prosesinin yerinə yetirilməsi proqramçıya həvalə olunur.

5.1. Səhvlərin səbəbləri

Səhvlər yeni proqram yazmağa başlayan proqramçıda da, peşəkar proqramçıda da olur. Lakin bu səhvlərin səbəbləri müxtəlif olur. Yeni proqram yazmağa başlayan proqramçı böyük proyektlərə girişmir, onun səhvləri adətən Pascal dilinin operatorlarını zəif bilməsi ilə əlaqədardır. Peşəkar proqramçı bir qayda olaraq kodlaşdırma mərhələsində səhvə yol vermir. Lakin alqoritmin bütün detalları aydın olmadıqda gələcək proqram layihələndirilərkən işin başlanğıc mərhələsində səhvə yol verə bilər. Digər tərəfdən yüz minlərlə operatorlardan ibarət olan proqramın ayrı-ayrı modulları arasındakı qarşılıqlı əlaqəni əvvəlcədən nəzərə almaq çətin olur. Ona görə də müxtəlif proqramçı fəndlərindən istifadə edərək proqramın etibarlılığın artırmaq və səhvlərin tez aradan qaldırılmasına çalışmaq lazımdır. Bu da əməktutumunun kifayət qədər azalmasına səbəb olur. Delphi 6 sistemi istənilən kvalifikasiyalı proqramçılara orientasiya olunmuş güclü sazlama vasitələrinə malikdir.

5.2. Sintaksis səhvlər

Sintaksis səhvlər kompilyator tərəfindən avtomatik olaraq aşkar olunur. Aşkar olunmuş səhvlər barədə məlumat redaktorun aşağı hissəsində kiçik pəncərədə təsvir olunur.



Şəkil 5.1. Kompilyator tərəfindən aşkar olunan sintaksis səhv barədə informasiya

Səhvlər barədə məlumat olan sətirdə mausun sol düyməsini iki dəfə vursaq, Delphi 6 sistemi səhv olan sətri işıqlandırmaqla redaktora qoşulacaqdır. Məlumat (ingilis dilində) səhvi kifayət qədər ətraflı təsvir edir, bu da onun səbəbini aşkar etməyə imkan verir.

Məsələn:

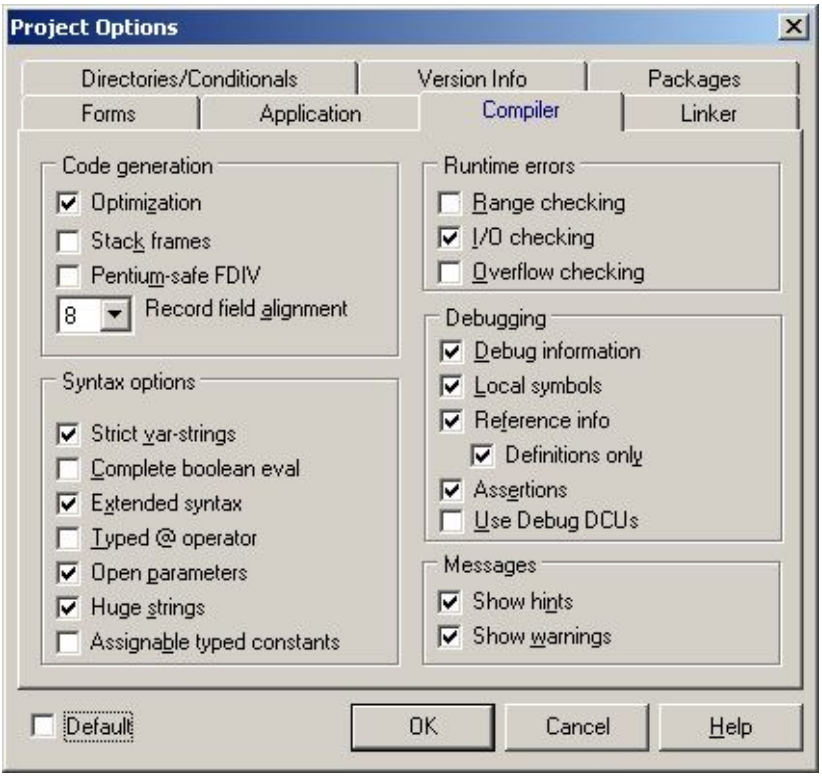
Undeclared identifier: x

Bu halda x identifikatoru elan olunmamışdır.

5.3. Məntiqi səhvlər

Məntiqi səhvlərin qarşısının alınmasının, aşkara çıxarılması və aradan qaldırılmasının bir neçə üsulu vardır. Onların hamısından bir-biri ilə kombinasiyalı şəkildə istifadə olunur.

Tez-tez rast gəlinən səhvləri izləməyə proqramın özünü məcbur etmək olar. Bunun üçün Project ▶ Options əmri ilə çağrılan dialoq pəncərəsinin Compiler səhifəsində aşağıdakı işləri görmək lazımdır.



Şəkil 5.2 Səhvlərin maksimal nəzarətdə saxlanması üçün kompilyatorun sazlanması

O Code generation panelində Optimization qoşulmuşdursa, bayraqcığı götürək. Bu zaman kompilyator optimizasiya olunmuş kod yaradarkən, Pascal-da reallaşdırılmış alqoritmik detallarının yaxşılaşdırılmasına əsaslı dərəcədə təsir edir. Məsələn, əgər proqramçı prosedurada aralıq nəticənin yadda saxlanılması üçün lokal x dəyişənini daxil edirsə:

```
function Sum:integer;
        var x:integer;
begin
x:=strToInt(Edit1.Text)+StrTo Int(Edit2.Text);
Result:=x
end;
```

Proqram kodu optimizasiya olunduqdan sonra aralıq dəyişəndən istifadə etməyə bilər və hesablamanın nəticəsi aşağıdakı kimi olar:

```
Result:=StrToInt(Edit1.Text)+StrToInt(Edit2.Text)
```

Bu isə o, deməkdir ki, proqramçı proqramın işi zamanı x lokal dəyişəninin qiymətini bilə bilməz, çünki real olaraq əməli yaddaşda bu dəyişənə yer ayrılmır.

○ Runtime errors panelində aşağıdakı bayraqçıqlar qoyulmalıdır. Range checking (massivin indeksinin kənara çıxmasına nəzarət), I/O checking(daxil/xaric etmə səhvlərinə nəzarət) və Overflow checking (tam tipli dəyişənlərlə əməliyyatlarda daşmaya nəzarət). Belə ki, Pascal-da bütün tiplər ciddi məhdudlaşdırılmış diapazona malikdir. Məsələn, iki ədədin toplanmasının nəticəsi bu diapazondan kənara çıxırsa, axırıncı bayraqcıq bu tip səhvlərə nəzarət edəcəkdir.

○Debugging(sazlama) panelində Debug Information (Sazlama informasiyasının əlavə olunması), Local Symbols(lokal dəyişənlərin qiymətinə baxılması), Reference info (kodun strukturuna baxılması), Assertions(Assert prosedurunun maşın koduna qoşulması) və Use Debug DCUs (VCL standart modullar kitabxanası komponentlərinin sazlama versiyasından istifadə olunması) bayraqcıqlarını qoşaq.

Delphi 6-da sazlama informasiyası olmadan sazlama prosesini həyata keçirmək qeyri-mümkündür. Assert prosedurası sazlama funksiyasını yerinə yetirir. Bir qayda olaraq proqramın axırıncı versiyasında o, artıq lazım olmur və onun çağrılması ilkin mətndən onların sayı çox olduğundan əlverişli deyil. Bu prosedura üçün maşın kodunun generasiyasını Assertions bayraqcığının köməyi ilə yerinə yetirmək olar.

Standart moddulların sazlama versiyası Delphi 6 komponentləri ilə işin korrektliyini yoxlamaq üçün əlavə rejimlərə malikdir.

Delphi 6 mühitindən işə salınmış proqramda səhvə rast gəlinərsə (məsələn, indeksin mümkün sərhəddi aşması) proqramın yerinə yetirilməsi dayanacaq və səhv olan sətir (bizim halda-Seb (i):=0;) işıqlandırılacaq.

Bununla yanaşı Delphi 6 səhvin səbəbini tez aşkara çıxarmağa imkan verir.

Məsələn, mausun göstəricisini müxtəlif dəyişənlər üzərində gəzdirməklə onların qiymətini parlayan köməkçi məlumatda görmək olar (i=11).

Əksəriyyət hallarda Delphi 6 mühitindən işə salınmış proqram səhv səbəbindən dayanarsa, istənilən dəyişən və sabitlərin qiymətinə baxmaq olar.

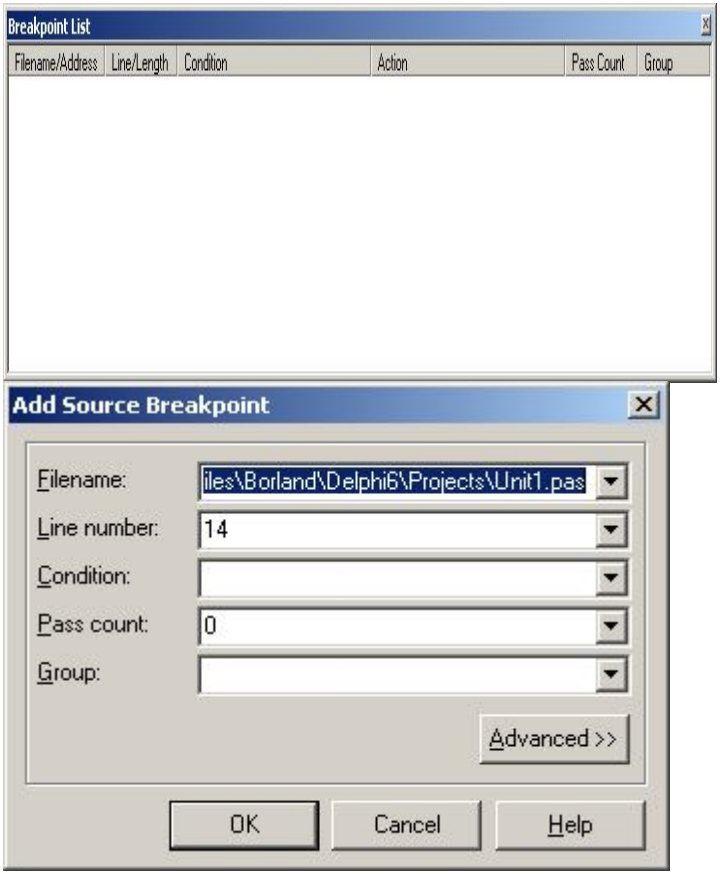
Daha mürəkkəb səhvləri proqramçı Delphi 6 mühitindən müstəqil olaraq izləyə bilər. Bunun üçün bir sıra standart üsullardan istifadə olunur ki, lakin bu zaman sazlanan proqramın Delphi 6 mühitindən işə salınması vacibdir. Yalnız bu halda işləmə mühiti lazımi səviyyədə proqramın yerinə yetirilməsinə və müxtəlif dəyişənlərin qiymətlərinin dəyişməsinə nəzarət edə bilir.

5.4. Yoxlamaya görə dayanma nöqtəsi

Yoxlamaya görə dayanma nöqtəsi proqramın öz işini dayandırdığı operatoru təyin edir və idarəni Delphi mühitinə verir. Yoxlamağa görə dayanma nöqtəsi

View▶Debug Windows▶Breakpoints əmrlərinin köməyi ilə verilir.

Dayanma nöqtəsi pəncərəsində yerinə yetirilməsindən sonra proqramın öz işini dayandırdığı bütün nöqtələrin siyahısı verilir.



Şəkil 5.3 Boş yoxlamaya görə dayanma nöqtəsi(yuxarıda)və yeni nöqtənin əlavə olunması pəncərəsi(aşağıda)

Yeni nöqtənin əlavə olunması üçün boş pəncərədə (yuxarıdakı şəkildə) siçanın sağ düyməsini basıb açılan kontekst menyudan Add əmrini seçmək lazımdır. Bu halda əlavə olunan nöqtənin parametrlərini daxil etmək üçün pəncərə (aşağıdakı) açılacaqdır.

File name-faylın adı və yolu

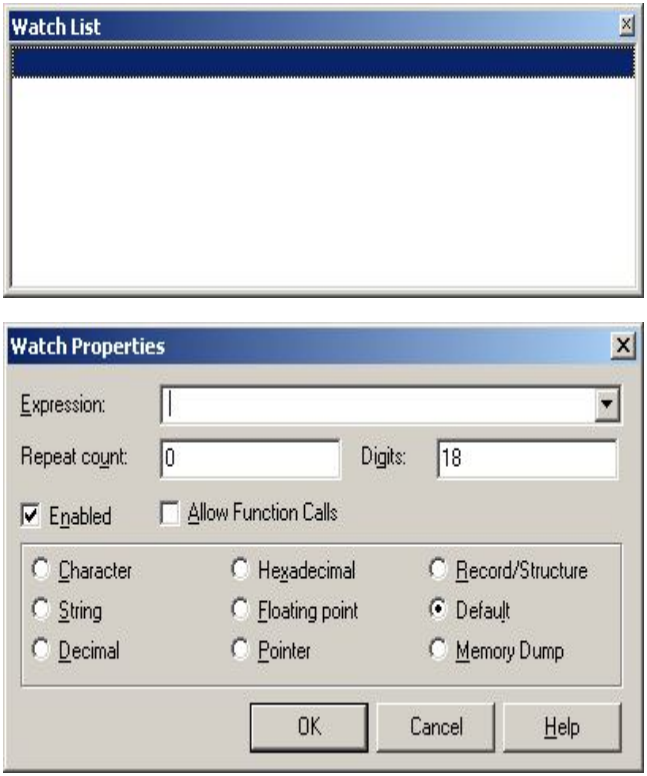
Line number-faylın başlanğıcından sətrin nömrəsi;

Condition-bu sahədə dayanma şərtini məntiqi ifadə şəklində vermək olar (məsələn, MyValue=MaxValue-12);

Pass Count - bu sahədə yerinə yetirilən proqramın yoxlamaya görə dayanma nöqtəsindən dayanmadan keçidlərinin sayını vermək olar.

5.5. Müşahidə pəncərəsi

Dəyişənlərin və ifadələrin vəziyyətinə nəzarət etmək üçün View▶Debug Windows▶Watches əmri ilə çağırılan pəncərədən istifadə etmək olar.



Şəkil 5.4 Boş müşahidə pəncərəsi(yuxarıda)
və oraya yeni ifadənin daxil edilməsi pənsərəsi(aşağıda)

Müşahidə pəncərəsi sazlama rejimində bu pəncərədə yerləşdirilən ifadənin qiymətinin dəyişməsini müşahidə etmək üçün nəzərdə tutulmuşdur. Yeni ifadənin daxil edilməsi üçün boş müşahidə pəncərəsində (yuxarıdakı) siçanın sağ düyməsini basıb kontekst menyudan New Watch əmrini seçirik.

Açılan pəncərədə Expression sahəsinə ifadəni daxil edirik. Repeat count sahəsinə verilənlər massivinin göstərilən elementlərinin sayını; Digits sahəsinə-həqiqi verilənlərin təsviri üçün qiymətli rəqəmlərin sayını; Enabled bayraqcığı ifadənin qiymətinin

hesablanmasına icazə və ya qadağa qoyur. Qalan elementlər qiymətlərin təsvir üsullarını təyin edirlər. Qeyd edək ki, Delphi-nin axırıncı versiyalarında sazlama rejimində siçanın göstəricisini dəyişənin üzərində saxlamaqla onun cari qiymətinə baxmaq olar.

5.6. Proqramın işinin məcburi dayandırılması

Əgər proqram Delphi mühitindən işə salınmışdırsa, onun işini dayandırmaq üçün:

1. Ctrl +F2- düymələrinin kombinasiyasından;
2.  düyməsindən;
3. **Run ▶ Program pause** əmrindən;
4. Proqramın yerinə yetirilən və ya gələcəkdə yerinə yetiriləcək hissəsinə yoxlamaya görə dayanma nöqtəsi qoymaqla.

5.7. Proqramın trassirovkası

Yoxlamaya görə dayanma nöqtəsi təyin olunmuş operator yerinə yetirilməmişdən əvvəl proqramın işi dayandırılacaq,idarə Delphi mühitinə ötürülməklə,müşahidə pəncərəsində müşahidə olunan dəyişən və ya ifadənini qiyməti veriləcək.Proqramçı artıq F7 və F8 və ya  və  düymələrinin köməyi ilə proqramın işinə addım-addım nəzarət edə bilər.

F8 düyməsinin sıxılması ilə cari sətirdə proqramlaşdırılmış əməliyyat yerinə yetiriləcək, proqramın işi növbəti sətrin yerinə yetirilməsindən əvvəl dayandırılacaq. Qeyd edək ki, kəsilməyə görə dayanma nöqtəsi qırmızı rənglə, növbəti işlənəcək sətir isə göy rənglə seçilir. Əgər proqram nəzarət nöqtəsində öz işini dayandırmışdırsa, onda cari sətir dayanma nöqtəsi ilə üst-üstə düşür və qırmızı rənglə seçilir. Cari sətir redaktor pəncərəsinin xidməti sahəsində xüsusi işarə ilə təyin olunur.

Yoxlamaya görə dayanma nöqtəsi təyin etmək və ləğv etmək üçün lazımı sətrin sol tərəfində xidməti sahədə mausun düyməsini vurmaq və ya kursoru həmin sətrin üzərinə gətirib F5 düyməsini sıxmaq lazımdır.

F7 düyməsinin sıxılması ilə mühit F8 düyməsinin sıxılmasına analoji işləri yerinə yetirir. Lakin cari sətirdə istifadəçinin altproqramına müraciət vardırsa, proqram öz işini bu altproqramın birinci icra olunan operatorundan əvvəl dayandıracaq. Belə ki, F7 düyməsi çağrılan altproqramların yerinə yetirilməsinə nəzarət etməyə imkan verir.

Proqramın müəyyən fraqmentini trassirovka etdikdən sonra F9 düyməsini sıxmaqla onun normal işini davam etdirmək olar.

Trassirovka dedikdə, proqramın icrası zamanı əmrlər ardıcıllığının yerinə yetirilməsi dəyişənlərin aldığı qiymətləri və başqa hadisələr haqqında məlumat almaq nəzərdə tutulur.

5.8. Kəsilmə nöqtələrində əməliyyatlar

Delphi 6-da ixtiyari bir nöqtə ilə bir və ya bir neçə əməliyyatı birləşdirmək olar. Bunun üçün dayanma nöqtəsi pəncərəsini aktivləşdirib, açılan kontekst menyudan Properties əmrini seçmək lazımdır. Açılan dayanma nöqtəsinin xassələri pəncərəsində Advanced düyməsinə vurmaqla, əlavə xassələrə müraciət əldə edə bilərik.

Pəncərənin aşağı hissəsində Actions paneli vardır ki, pəncərənin yuxarı hissəsində göstərilən dayanma nöqtəsi üçün əməliyyat təyin etmək olar.

- Break- qeyd olunmuş operatorun yerinə yetirilməsindən əvvəl proqramın işini dayandırması;

- Ignore subseqment exeptions- əgər bayraqcıq qoşulmuşdursa, cari sazlama seansı zamanı növbəti dayanma nöqtəsinə qədər bütün mümkün müstəsna hallar nəzərə alınmır;
- Handle subseqment exeptions- bu bayraqcıq qoşulan zaman əvvəlki bayraqcığın gördüyü işlər təxirə salınır və mümkün müstəsna halların emalı bərpa olunur;
- Log message- bu siyahı dayanma nöqtəsi ilə bağlı istənilən məlumatı seçməyə imkan verir;
- Eval expression- bu siyahı hər hansı ifadənin qiymətini hesablamağa və onun nəticəsini Log message siyahısından seçilən məlumata verməyə imkan verir.

Delphi 6-da dayanma nöqtələrini qruplaşdırmaq imkanı vardır. Bunun üçün pəncərədə Group və Disable Group siyahısından qrupun adını, Enable Group və Disable Group siyahılarının köməyi ilə seçilmiş qrupa aid olan bütün dayanma nöqtələrinin yerinə yetirilməsinə uyğun olaraq icazə və qadağa qoymaq olar.

5.9.Ifadənin qiymətinin hesablanması və dəyişdirilməsi

Enabute/Modify pəncərəsinin (şəkil) köməyi ilə istənilən ifadənin qiymətini öyrənmək və ya dəyişənə digər qiymət mənimsətmək olar. Bu pəncərə sazlama rejimində Ctrl+F7 düymələrinin sıxılması ilə çağrılır.

Bu pəncərə modal pəncərə olub, bağlanana qədər proqramın sazlanma rejimini təxirə salacaqdır.

Expression siyahısı olan sahəyə dəyişənin adını və ya bizi maraqlandıran ifadəni daxil edə bilərik.

Evalvate düyməsinə vurulması Result sahəsinə dəyişənin (ifadənin) cari qiymətini çıxaracaqdır.

5.10. Exception sinfi –müstəsna halların emalı

Exception sinfi **Tobject** baza sinfinin birbaşa varisidir. Özünün varisləri ilə birlikdə o, proqramda qeyri-korrekt əməliyyatların yerinə yetirilməsi nəticəsində meydana çıxan müstəsna halların emalı üçün nəzərdə tutulmuşdur: məsələn, sıfra bölmə, mövcud olmayan faylın açılması və s. Biz indi proqramın etibarlılığını yüksəltmək üçün müstəsna halların emalı proseduralarının istifadəsi və xassələrini öyrənəcəyik.

Müstəsna halların emalı üsullarına görə bir-birindən fərqlənən iki tip qoruyucu blok vardır:

Except (aradan qaldırmalı) və **Finally** (sona getməli).

Except blokunun strukturu aşağıdakı kimidir:

```
try
  operatorlar
except
 müstəsna halların emalediciləri
else
   operatorlar
end;
```

Finally blokunun strukturu:

```
try
  operatorlar
finally
  operatorlar
end;
```

hər iki tip qoruyucu blok **try** ehtiyyat sözü ilə başlayır, **end** ehtiyyat sözü ilə qurtarır.

Except blokunda operatorların yerinə yetirilmə ardıcıllığı aşağıdakı kimidir: Əvvəlcə **try... except** seksiyasında olan operatorlar yerinə yetirilir. Əgər operatorların yerinə yetirilməsi müstəsna halların baş verməsi ilə nəticələnmirsə, qoruyucu blok işini başa çatdırır və idarə **end** operatorundan sonrakı operatora ötürülür. Operatorların yerinə yetirilməsi müstəsna halların baş verməsi ilə nəticələnərsə, idarə **except** seksiyasındakı müstəsna halların emaledicilərinə ötürülür və əgər belə emaledici tapılmazsa, idarə **else** sözündən sonrakı birinci operatora ötürülür. **Finally** blokunda **try...finally** seksiyasında müstəsna hal baş verib-verməməsindən asılı olmayaraq idarə **finally...end** seksiyasına ötürülür. Əgər müstəsna hal baş verirsə, onda try...finally seksiyasında müstəsna hal baş verən operatordan sonrakı bütün operatorlar ötürülür və idarə **finally...end** seksiyasında yerləşən birinci operatora ötürülür. Əgər müstəsna hal baş vermirsə, onda bu operatora **try...finally** seksiyasında axırıncı operator yerinə yetirildikdən sonra idarə ötürülür.

Except blokunda müstəsna halların emaledicisi aşağıdakı sintaksisə malikdir:

on istisna_sinfi **do** operator;

burada **on , do**- ehtiyyat söz, istisna_sinfi- müstəsna halları emal edən sinif, operator- **except** blokundan kənara müraciətə malik **goto** operatorundan başqa Object Pascal-ın istənilən operatorlarıdır.

Müstəsna vəziyyət (**exception**) dedikdə, proqramın icrası zamanı yarana bilən və əməllərin normal ardıcıllığında dəyişikliklər edən xüsusi anomal vəziyyət nəzərdə tutulur. Dildə bu vəziyyəti aşkar, təsnif və emal edən vasitələr nəzərdə tutulur.

Bir fakta diqqət yetirmək lazımdır ki, sinfin adı özünə məxsus seçmə açarı rolu oynayır, müstəsna halın emalı isə do sözündən sonra gələn operator vasitəsi ilə yerinə yetirilir. Bu operator xüsusi halda mürəkkəb operator ola bilər.

Lazımi emaledicinin axtarışı siyahının əvvəlindən verilmiş tip müstəsna halı aradan qaldırıla bilən sinif tapılana qədər davam etdirilir. Əgər uyğun sinif tapılmırsa, idarə else sözündən sonra gələn operatora ötürülür. Operatorda else sözü olmaya da bilər, bu halda müstəsna halların emalı susmaya görə yerinə yetirilir.

Əgər proqramçı üçün yalnız müstəsna halın baş verməsi faktını aydınlaşdımaq və ona bağlı səhvin tipinin əhəmiyyəti yoxdursa, o, except...end seksiyasında emalediciləri və else sözü ilə birlikdə yazmaya bilər.

```
try
.....
except
ShowMessage('Sehv!')
……
end;
```

Qoruyucu blokların bir-birinin daxilində yerləşməsinə heç bir məhdudiyyət qoyulmur. Başqa sözlə, yuxarıdakı təsvirlərdə operatorlar və ya operator olaraq Object Pascal-ın istənilən operatoru, o cümlədən **try...except** və ya **try...finally** istifadə oluna bilər.

```
try
 ......
  try
   ......
    finally
     ......
    end;
except
     on EmatchError do
      begin
       ......
       try
        ......
         try
          ......
           end;
            ......
       end;
        ......
end;
 ......
 end;
```

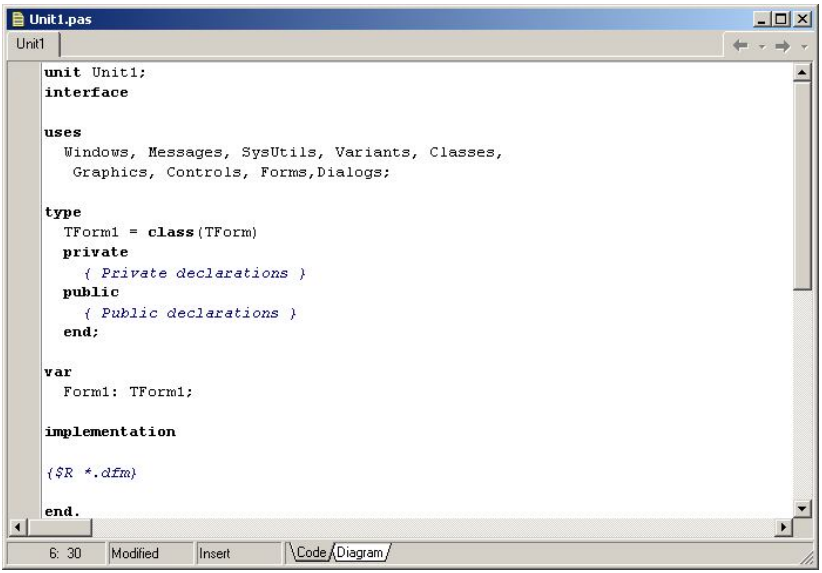
6. Delphi əsas komponentləri və onların işləmə prinsipləri

6.1. Komponentlərlə tanışlıq

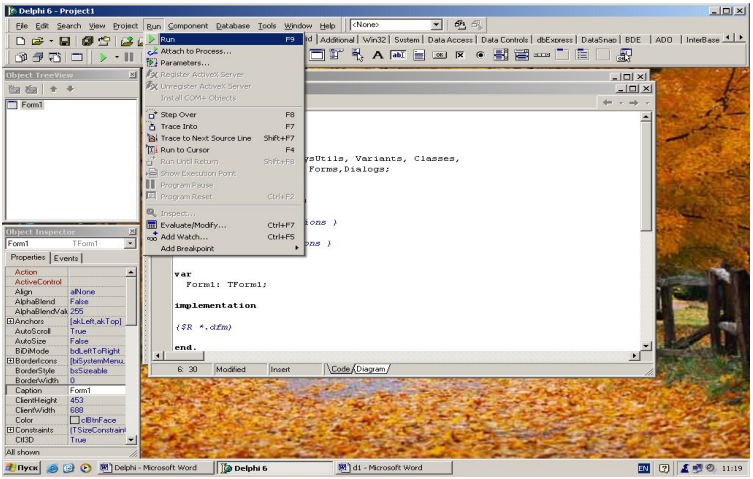
Komponentlər işləyən proqram tərəfindən yaradılan görünən təsvirlərin konstruksiyası üçün lazım olan elementlərdən təşkil olunur. Qeyd etmək lazımdır ki, bir çox komponentlər vardır ki, onlar görünməyən təsvirlər yaratmaqla bu və ya digər məsələlərin həllində çox vacib rol oynayırlar. Komponent dedikdə, işlənən proqramın əvvəlcədən hazırlanmış müəyyən fraqmenti nəzərdə tutulur. Delphi 1-ci versiyasında 70-ə yaxın, 6-cı versiyasında isə 300-dən artıq komponent vardır.

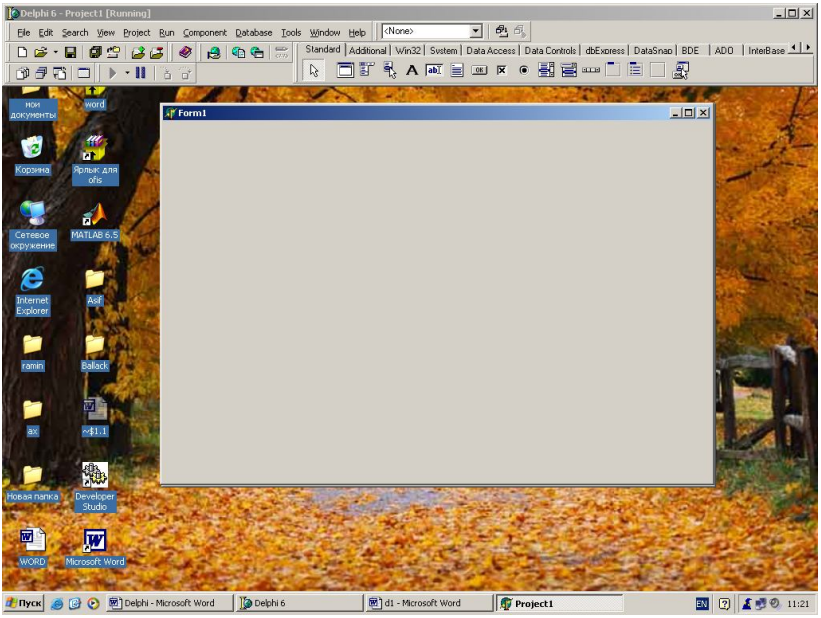
6.2. Forma komponenti

Windows işləmələrində pəncərə olaraq qarpşımıza çıxan bütün obyektlər Forma adlandırılırlar.Buna görə də bütün **Delphi işləmələri heç olmasa bir** Forma daxilinə alır. **Delphi işə salındıqda** Form1 və Unit1 (proqramların yazıldığı)adında pəncərə hazır verilməkdədir.Bu zaman susmaya görə adı Project1 olan bir proyekt və bu proyekt üçün avtomatik olaraq Form1 adında bir forma hazırlanır.Eyni zamanda unit1.pas adında Pascal proqramı daxilinə alan bir unit hazırlanaraq bu proyektə daxil edilir.Başlanğıcda Pascal proqramı daxilinə alan pəncərə form pəncərəsinin altında qaldığına görə masaüstündə görünmür.



Bu pəncərədəki bütün proqram sətirləri avtomatik olaraq hazırlanmaqdadır. Pascal proqramında istifadə ediləcək bütün unit-lər Uses operatoru ilə proqrama daxil edilir. Pascal proqramları daxilində var dəyişənlərin təsviri bloku və proqramın sonunda end. operatorunun verilməsi məcburidir.Delphi işimizi sadələşdirmək üçün hər proyektdə yer alan təsvir bloklarını və proqram sətirlərini avtomatik olaraq hazırlayır.Avtomatik olaraq hazırlanan Delphi proyekti üzərində heç bir iş görmədən çalışdırsaq,Delphi tərəfindən avtomatik olaraq hazırlanaraq proyektə daxil edilən Forma və ya pəncərə ekrana gəlir.Üzərində işlədiyimiz Delphi proyektini çalışdırmaq üçün Run menyusundan Run düyməsini və ya F9 funksional düyməsini sıxmaq lazımdır.





6.2.1. Forma komponenti.Xassələr.Hadisələr

Forma-Delphi 6-nın ən vacib komponenti olub, Windows pəncərədən ibarətdir. Formada proqramın işini idarə edən (vizual) görünən və (vizual olmayan) görünməyən elementlər yerləşmişdir.

Forma (TForm) özündə çoxlu sayda xassələri, üsul və hadisələri saxlayır. (Cədvəl 1.1)

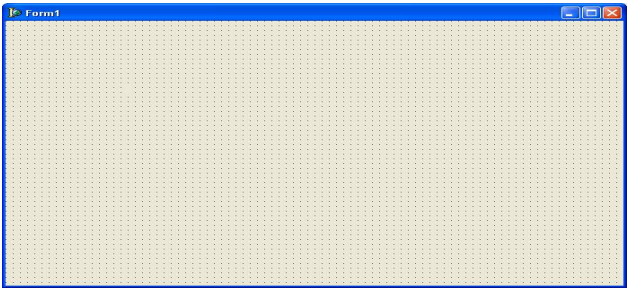
Xassə	Təyinatı
Active	Forma giriş fokusuna malikdirsə, true qiymətini alır.
ActiveControl	Forma üzərində giriş fokusuna malik obyekt
BorderIcons	Formanın sistem ikonalarının siyahısı
BorderStyle	Formanın sərhədlərinin görünüşü
Canvas	Formada şəkilçəkmə oblastı
ClientRect	
ClientHeight	
ClientWidth	Formanın ölçüsü
DropTarget	Əgər forma daşıma əməliyyatlarında qəbul-edici kimi çıxış edə bilirsə, true qiymətin alır.
Floating	Əgər forma digər pəncərələrə daşına bilirsə , true qiymətini alır.
FormState	Formanın cari vəziyyəti
FormStyle	Formanın stili
HelpFile	Forma üçün məlumat faylının adı
Icon	Forma gizlədildikdə,formanı təsvir edən ikona
KeyPreview	Əgər forma basılan düymələr haqqında informasiyanı onun üzərində yerləşən obyektlərdən tez alırsa,true qiymətini alır.
Menu	Formanın əsas meyusuna müraciət(Tmenu)
ModalResult	Əgər forma modal dialoq pəncərəsi kimi işləyirsə onun qaytardığı qiymət
Parent	Formanın ”sahibi”
PixelsPerInch	Düymədə olan piksellərin sayı.
Position	Proqramda forma açılarkən ekrandakı vəziyyəti
PrintScale	Formanı çapa verərkən miqyaslaşdırma
Scaled	Formanın ölçüləri PixelsPerInch xassəsinin qiymətlərinə uyğunlaşdırılırsa,true qiymətini alır.
Visible	Proqram işləyərkən forma görünürsə,true qiymətini alır
WindowState	Formanın vəziyyəti

Cədvəl1 .2. TForm sinfi ilə dəstəklənən hadisələr

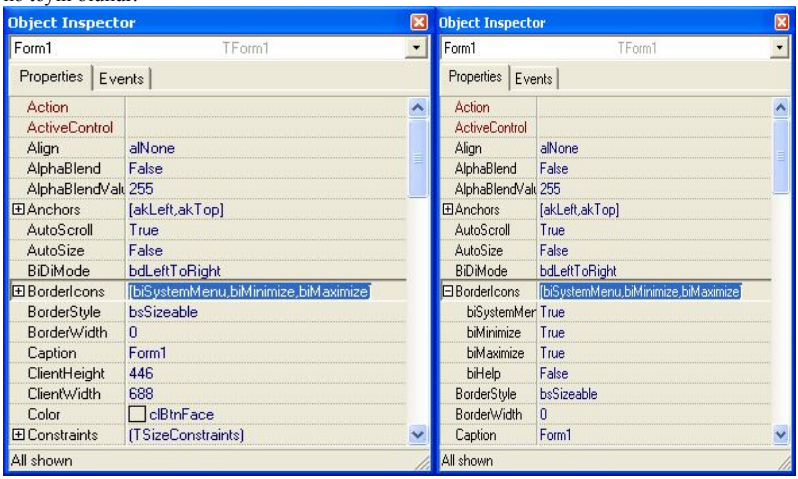
Hadisələr	**Generasiya şərti**
OnActivate	**Forma aktiv olur**
OnClose	**Forma bağlanır**
OnCloseQuery	Formanın bağlanmasına sorğu daxil olur
OnCreate	**Forma yaradılır**
OnDeactivate	Forma giriş fokusunu itirmişdir
OnDestroy	**Forma ləğv olunur**
OnHelp	**Forma məlumat almaq üçün sorğu alır.**
OnHide	Forma görünməz olur.
OnPaint	Formada şəkil çəkilə bilər.

OnShortCut	Istifadəçi hələlik istifadə olunmamış klaviatura kombinasiyasını basmışdır.
OnShow	Forma görünən olur.

6.2.1.1.Minimize-maximize xassəsi

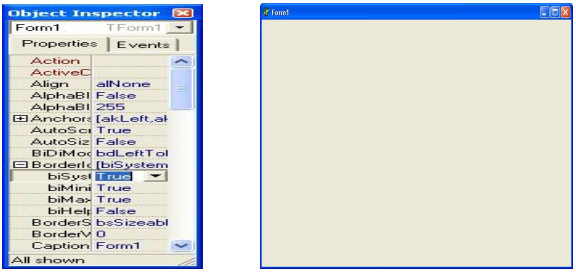


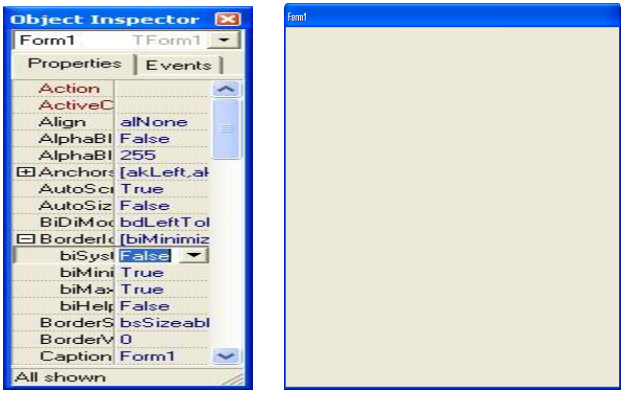
Yuxarıda görünən forma standart bir Windows pəncərəsinin sahib olduğu minimize, Maximize və formanı qapat xassələrinə malik olub bu pəncərənin layihələndirmə zamanı xassələridir.Proqram işləyərkən bu xassələr Object Inspector pəncərəsinin BorderIcons xassəsi ilə təyin olunur.



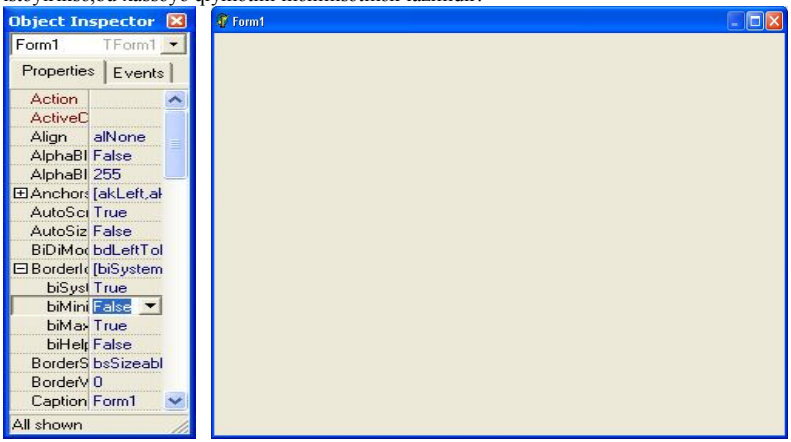
Bu xassənin biSystemMenu,biMinimize,biMaximize,biHelp alt xassələri vardır.

- biSystemMenu xassəsinin qarşısında true qiyməti vardır.Bu proqram işləyərkən Windows-un standart düymələrinin ekranda görünməsini təmin edir.

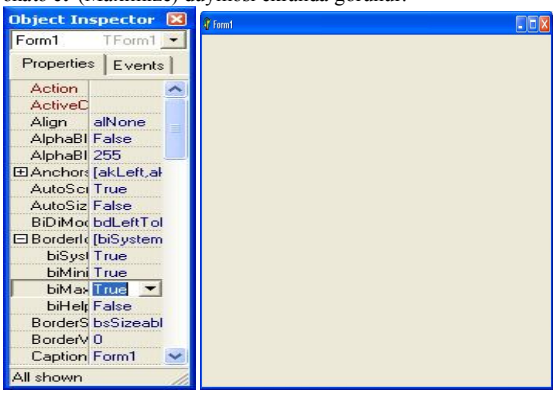




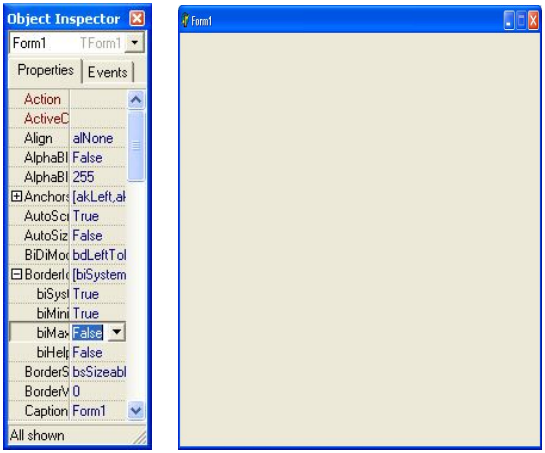
•biMinimize xassəsi başlanğıcda true qiymətinə malik olduğundan proyekt işlədikdən sonra formanın Minimize düyməsinə malik olduğunu göstərir.Əgər bu düyməni götürmək istəyiriksə,bu xassəyə qiymətini mənimsətmək lazımdır.



•biMaximize xassəsi başlanğıcda true qiyməti aldığından proqram işlədikdən sonra "ekranı əhatə et"(Maximize) düyməsi ekranda görünür.



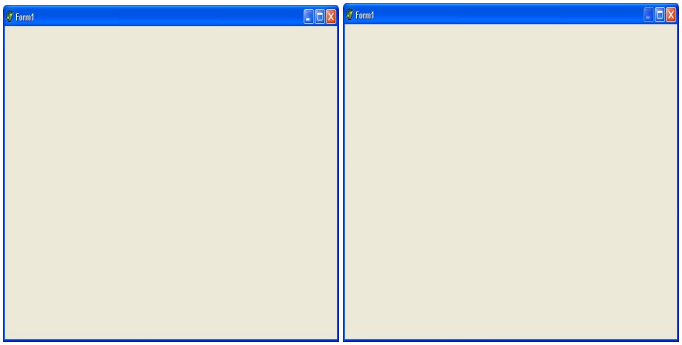
Bu düymənin funksiyasını dayandırmaq üçün biMaximize xassəsinə false qiyməti mənimsədilməlidir.



Yuxarıda göstərdiyimiz xassələrə qiymətlər layihələndirmə zamanı mənimsədilir.Proqram işləyərkən də bu xassələrə qiymət mənimsətmək mümkündür.Obyektlər inspektoru pəncərəsində BorderIcons xassəsinin ala biləcəyi qiymətlər onun qarşısında çoxluqlar konstruktoru şəklində göstərilmişdir.Aşağıdakı proqram görüntüsündə layihələndirmə zamanı true qiyməti alan biMinimize xassəsi proqram işlədikdən və forma üzərində mausun sol düyməsini vurduqdan sonra false qiymətini alır.

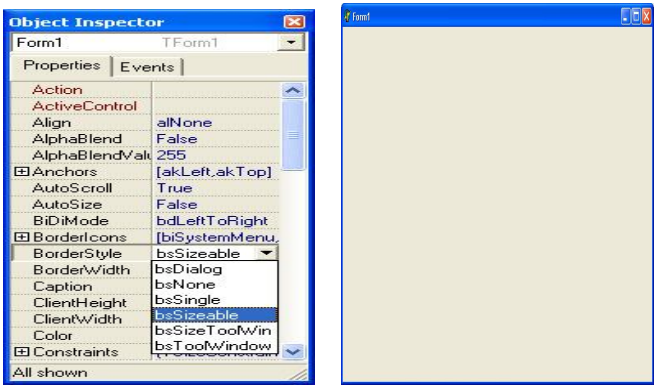
```
procedure TForm1.FormClick(Sender: TObject);
begin
  BorderIcons:=BorderIcons-[biMinimize];
end;
```

end.



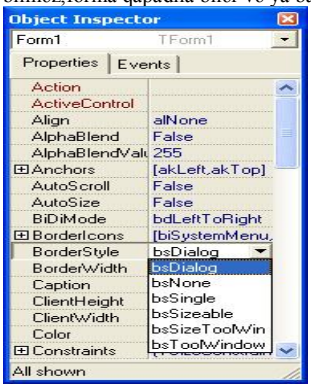
6.2.1.2.Formaların ölçüsünün dəyişdirilməsi.

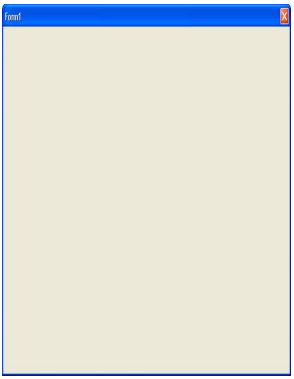
Delphi-də formaların ölçüsünü layihələndirmə zamanı Obyektlər Inspektoru pəncərəsindən və proqram işləyərkən dəyişdirmək mümkündür.Bu məqsədlə Obyektlər Inspektoru pəncərəsinin BorderStyle xassəsinin qarşısında mausun sol düyməsini vuraraq alt xassələrin görünməsini təmin etmək lazımdır.



Başlanğıcda bu xassədə bsSizeable qiyməti seçilmişdir və bu mausun köməyi ilə pəncərənin böyüdülüb-kiçilməsini(normal Windows pəncərələrində olduğu kimi) təmin edir. BsNone qiyməti seçildikdə isə formanın sərhəd xətləri,forma başlığı,minimize,maximize və qapat düymələri görünməz.

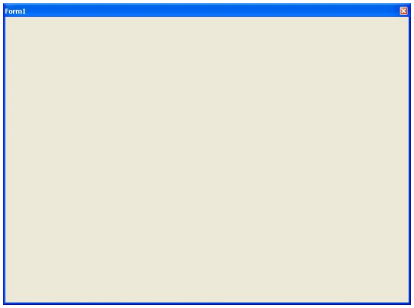
bsDialog qiyməti seçilərsə, BorderIcons xassəsinə mənimsədilən qiymətlər öz funksiyalarını itirir və minimize,maximize düymələri ləğv edilir.Bu tip formanın ölçüsü dəyişdirilə bilməz,forma qapadıla bilər və ya başlıq hissədən tutularaq masaüstünə daşına bilər.



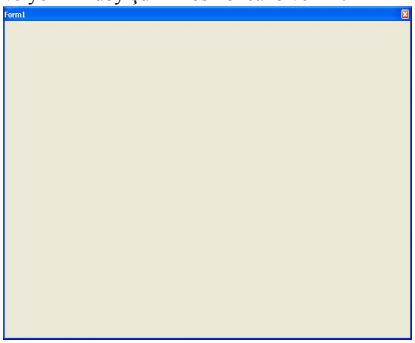


bsSingle qiyməti seçilərsə, minimize,maximize və qapat düyməsi mövcud olan formanın yeri dəyişdirilə bilər,lakin ölçüsü dəyişdirilə bilməz.

bsSizeToolWin qiyməti seçilərsə qapat düyməsi mövud olub formanın yeri və ölçüsü dəyişdirilə bilər.

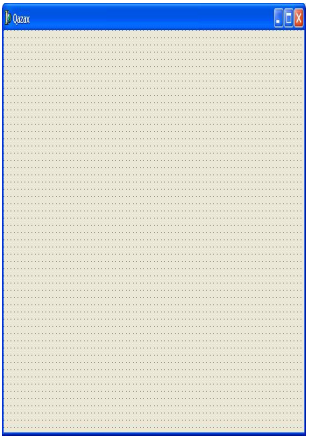


bsToolWindow qiyməti seçilərsə ,forma qapatma düyməsi mövcud olub ,formanın ölçülərinin və yerinin dəyişdirilməsinə icazə verilir.



6.2.1.3 Formalara verilən başlıq və adlar.

Vaşlanğıcda proyektdə yer alan formalar Form1,Form2,…başlığına(caption)malik olurlar.Caption xassəsini qarşısındakı Form1 başlığı silinərək formaya istənilən ad verilə bilər.Formaya Qazax adı verərkən onun eyni zamanda forma başlığına verildiyini görərik.



Proqram işlədikdə forma başlığı aşağıdakı üsulla dəyişdirilə bilər.

```
procedure TForm1.FormClick(Sender: TObject);
begin
Form1.Caption:='Qazax';
end;

end.
```

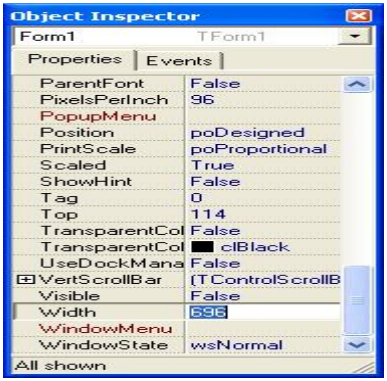
Formaların Name xassəsi Obyektlər Inspektoru pəncərəsindən dəyişdirilə bilər.Proqram işləyərkən onun dəyişdirilməsi mümkün deyildir.

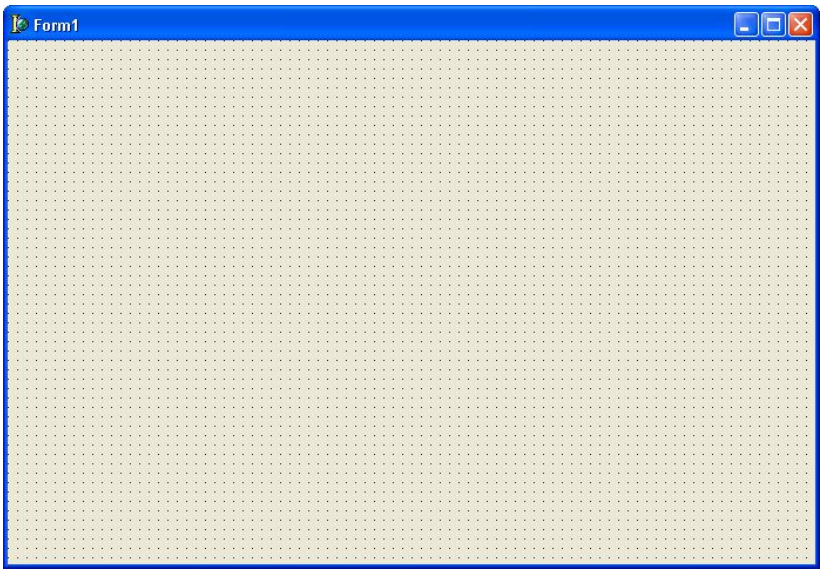
Form1 formanın proyekt daxilində Name səhifəsində yazılan adıdır.Əgər Name xassəsi qarşısına Sebuhi yazmış olsaydıq,onda yuxarıdakı əmr Sebuhi.Caption:='Qazax'; şəklində yazılardı.

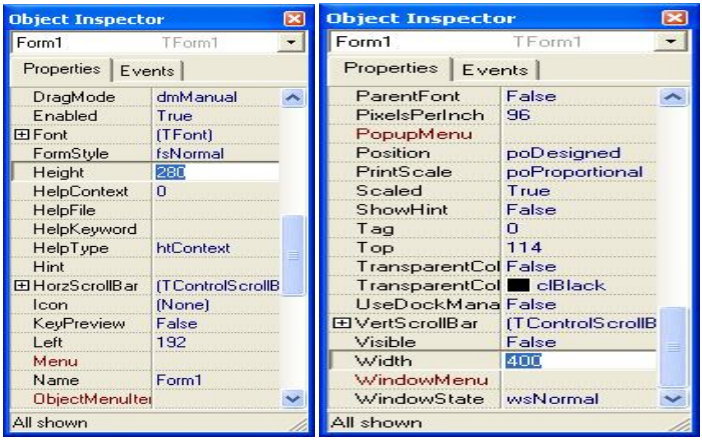
6.2.1.4.Forma ölçülərinin Width və Height xassələri ilə dəyişdirilməsi.

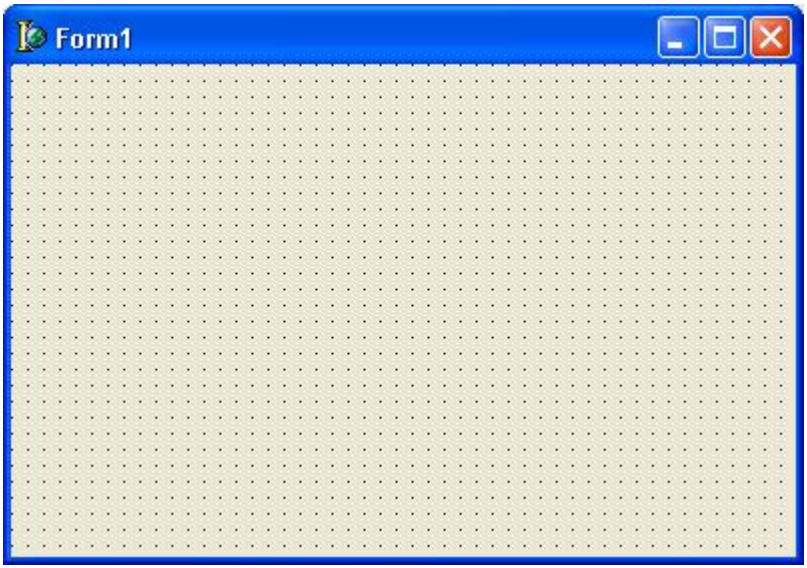
Başlanğıcda formanın genişliyi (Width) və hündürlüyü (Height) aşağıdakı şəkildə göstərildiyi kimi olub, Obyektlər Inspektoru pəncərəsindən Width və Height xassələrinin köməyi ilə həm layihələndirmə və həm də proqram kodu yazılaraq onun ölçüləri dəyişdirilə bilər.


Object Inspector
Form1 TForm1
Properties Events
DockSite False
DragKind dkDrag
DragMode dmManual
Enabled True
Font (TFont)
FormStyle fsNormal
Height 480
HelpContext 0
HelpFile
HelpKeyword
HelpType htContext
Hint
HorzScrollBar (TControlScrollB
Icon (None)
KeyPreview False
Left 192
Menu
All shown


Object Inspector
Form1 TForm1
Properties Events
ParentFont False
PixelsPerInch 96
PopupMenu
Position poDesigned
PrintScale poProportional
Scaled True
ShowHint False
Tag 0
Top 114
TransparentCol False
TransparentCol clBlack
UseDockMana False
VertScrollBar (TControlScrollB
Visible False
Width 698
WindowMenu
WindowState wsNormal
All shown


Form1

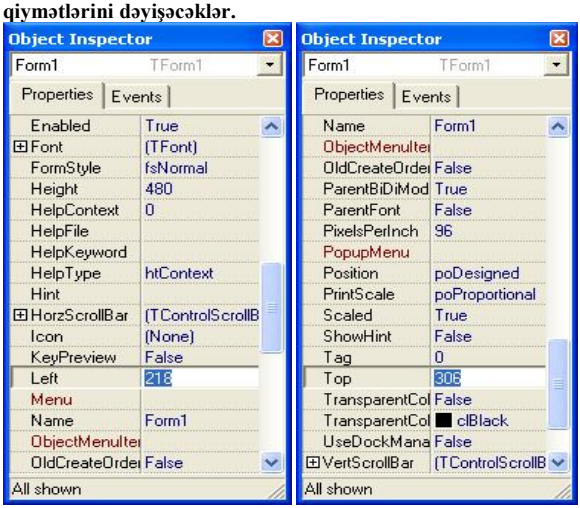




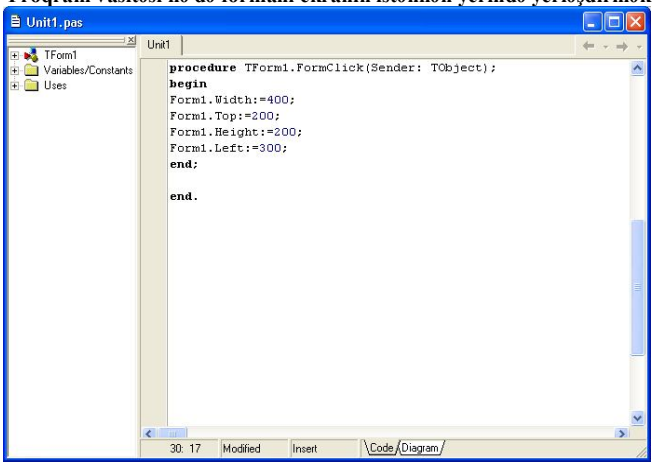

```
procedure TForm1.FormClick(Sender: TObject);
begin
Form1.Width:=300;
Form1.Height:=200;
end;

end.
```

6.2.1.5. Formanın başlanğıc nöqtəsinin Left və Top xassələri ilə təyin olunması.

Proyekt ilk dəfə işə salındıqda alınan formanın x,y koordinatları (yuxarı sol küncünün) Obyektlər Inspektoru pəncərəsinin Left və Top **xassələri ilə təyin olunub,layihələndirmə zamanı formanın maus ilə yerinin dəyişdirilməsi zamanı** Left və Top **xassələri də öz qiymətlərini dəyişəcəklər.**

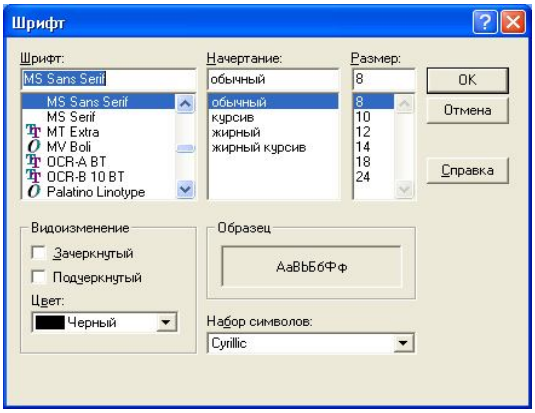


Proqram vasitəsi ilə də formanı ekranın istənilən yerində yerləşdirmək olar.



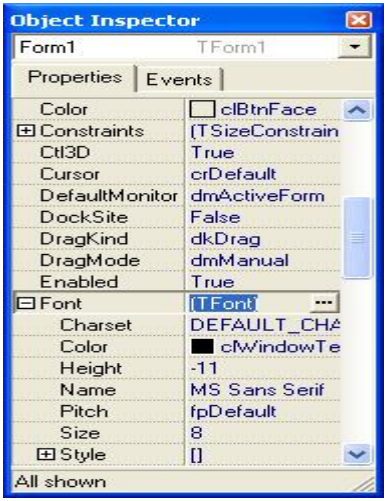
6.2.1.6. Font xassəsi.

Delphi-də forma üzərində yazılacaq yazıların şriftini müəyyən etmək üçün iki üsuldan istifadə olunur.Birinci üsulda Obyektlər Inspektoru pəncərəsində Font xassəsinin qarşısındakı (üç nöqtə) düyməni vuraraq dialoq pəncərəsinin açılmasını gözləməkdir.

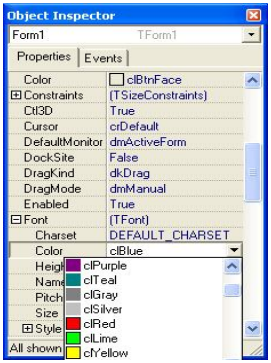


Bu dialoq pəncərəsində susmaya görə MS Sans Serif şrifti,Normal yazı tipi,qara rəng götürülmüş və yazıların ölçülərini istədiyimiz kimi seçmək imkanı vardır.

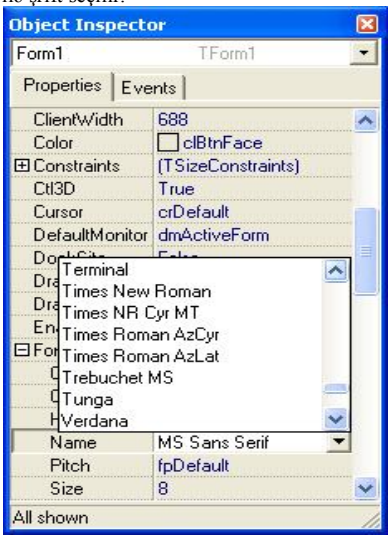
Ikinci üsul isə bu işləri dialoq pəncərəsindən deyil,Obyektlər Inspektoru pəncərəsindən yerinə yetirməkdir.Bu məqsədlə Font xassəsinin solundakı + işarəsini mausun sol düyməsi ilə vuraraq alt xassələrin açılmasını gözləyirik.



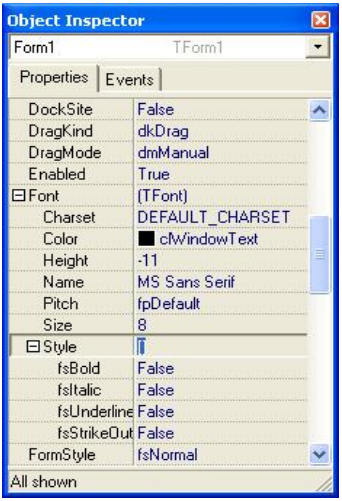
Burada Color xassəsinin sağında olan siyahıdan yazı rəngini seçmək olar.



Height xassəsi ilə yazının hündürlüyü təyin olunur.
Name xassəsi ilə şrift seçilir.

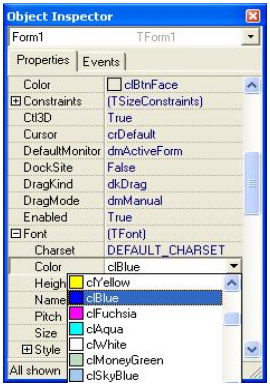


Style xassəsi ilə yazıların qalınlığı,kursiv və altı xətli olması müəyyən edilir.Alt xassələrə true qiyməti mənimsədildikdə həmin yazı tipi seçilir.

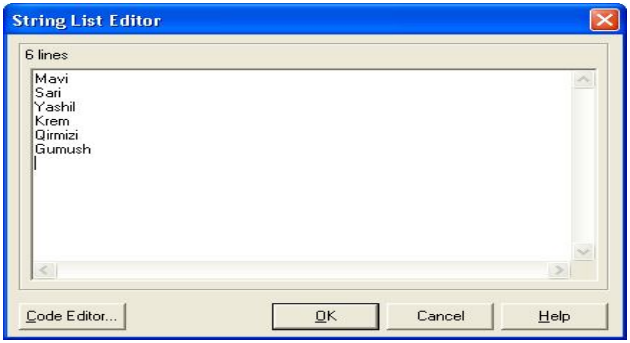


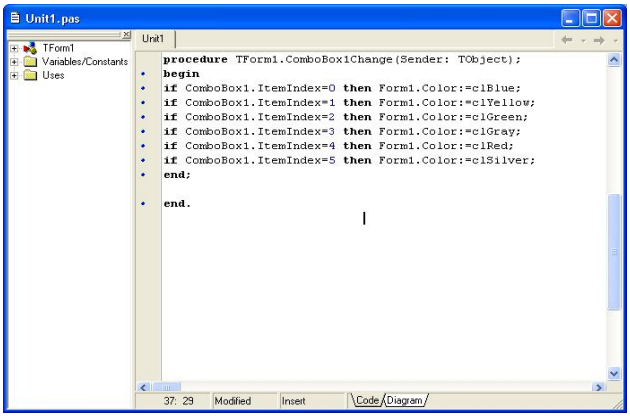
6.2.1.7. Forma rənginin seçilməsi(Color).

Delphi-də başlanğıcda formaların rəngi krem rəng olaraq seçilmişdir.Formaların rəngini dəyişdirmək üçün Obyektlər Inspektoru pəncərəsində Color xassəsinin qarşısında mausun düyməsini vuraraq ,sağda yerləşən ▼ düyməsini sıxaraq açılan siyahıdan rəngi seçmək lazımdır.

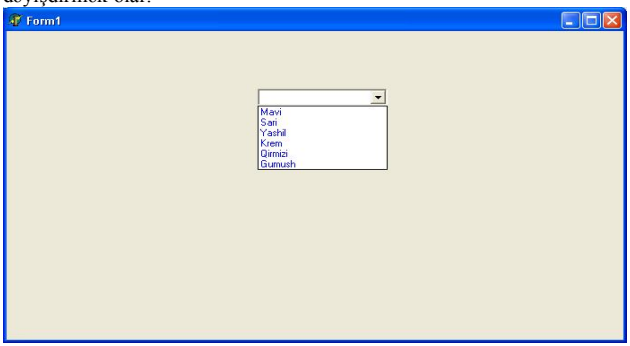


Formalara aid rəngi proqram işləyərkən də dəyişdirmək mümkündür.Komponentlər palitrasının Standard panelindən ComboBox komponentini forma üzərində yerləşdirərək formanın rəngini dəyişdirmək üçün aşağıdakı proqramı yazaq. ComboBox komponentinin Items xassəsinin qarşısını vuraraq String List Editor dialoq qutusunun açılmasını gözləyirik.





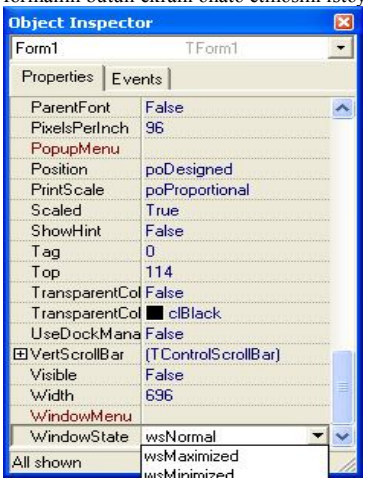
Proqram işlədikdən sonra aşağıda açılan pəncərədən rəngi seçməklə formanın rəngini dəyişdirmək olar.



6.2.1.8. Proyektlərdə formanın vəziyyətinin təyin olunması(WindowState).

Object Inspector pəncərəsindəki Form1-ə aid olan xassəsi başlanğıcda qiymətinə malikdir.Bunun mənası ondan ibarətdir ki,forma layihələndirmə mərhələsində hansı

ölçüdədirsə,proyekt işlədikdən sonra da həmin ölçüdə olacaqdır.Əgər proyekt işlədikdə formanın bütün ekranı əhatə etməsini istəyirsinizsə,onda xassəsinə qiyməti mənimsədilməlidir.



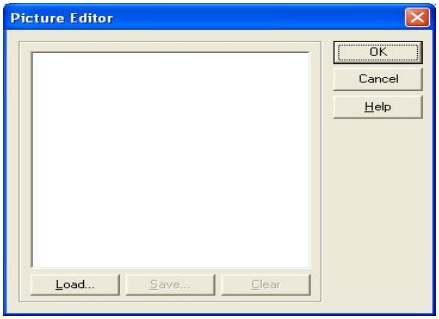
Ikinci üsul kimi proqram vasitəsi ilə bu xassəni dəyişdirmək mümkündür.

Form1.WindowState:=wsMaximized;

6.2.1.9. Ikonanın seçilməsi.(Icon).

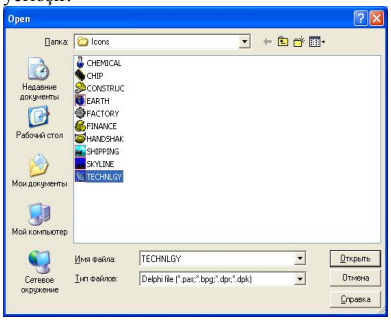
Delphi daxilində Object Inspector pəncərəsinin Icon xassəsinin qarşısında mausun düyməsini vurduqda üç nöqtəli kiçik bir düymə əmələ gəlir.

Bu düymənin üzərində mausun düyməsini sıxaraq Picture Editor dialoq pəncərəsini açırıq.

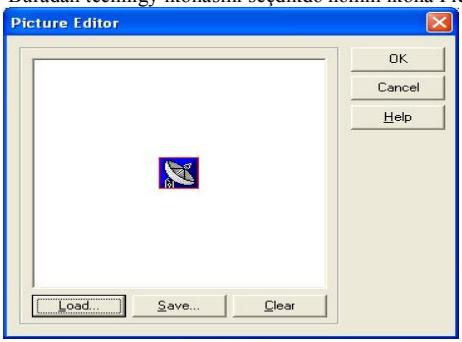


Ikona seçmək üçün Load düyməsi üzərində mausun sol düyməsini vuraraq Load Picture dialoq pəncərəsini açırıq.

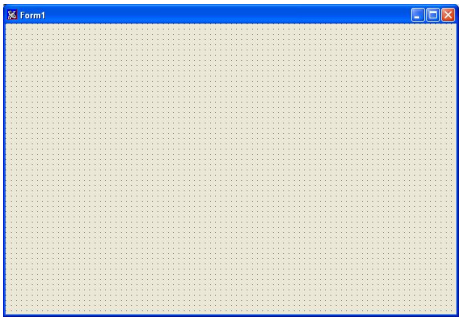
*.Ico uzantılı fayllar Program Files\Common Files\Borland Shared\Images\Icons kataloqunda yerləşir.



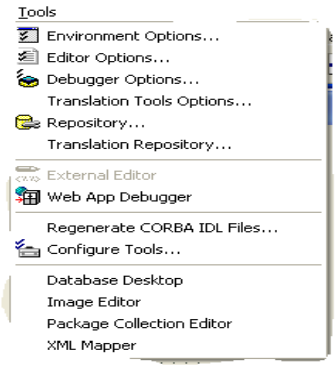
Buradan technlgy ikonasını seçdikdə həmin ikona Picture Editor dialoq pəncərəsində yer alır.

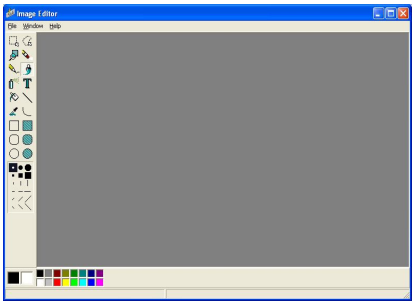


OK düyməsini sıxaraq Form1 üçün ikona dəyişdirmək işini başa çatdırmış oluruq.



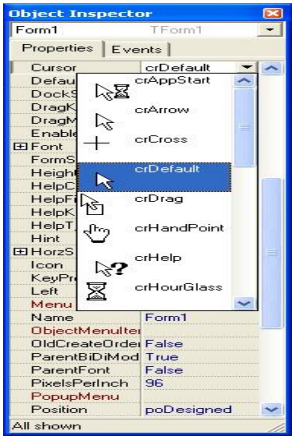
Əgər özümüz ikona yaratmaq istəyiriksə,Tools menyusundan Image Editor əmrini seçərək Image Editor dialoq pəncərəsini ekrana gətiririk.





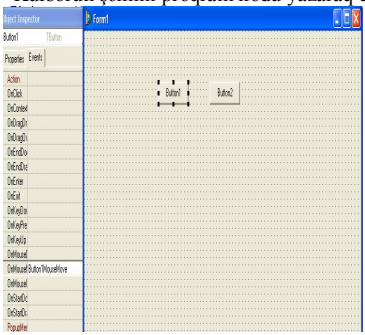
6.2.1.10. Kursorun şəklinin dəyişdirilməsi.(Cursor).

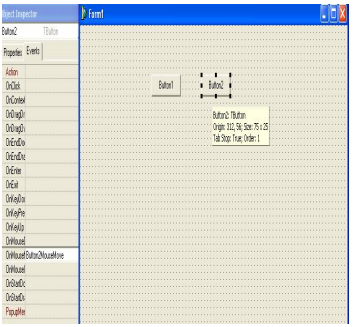
Kursorun şəklinin dəyişdirilməsi üçün Cursor xassəsinin qiyməti dəyişdirilməlidir.



Burada 21 kursor tipi olub susmaya görə crDefault qiyməti seçilmişdir ki,bu da normal kursor şəklidir.

Kursorun şəklini proqram kodu yazaraq da dəyişdirmək olar.

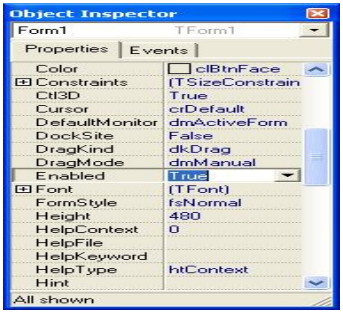




```
procedure TForm1.Button1MouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
begin
Cursor:=crAppStart;
end;

procedure TForm1.Button2MouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
begin
Cursor:=crHourGlass;
end;

end.
```

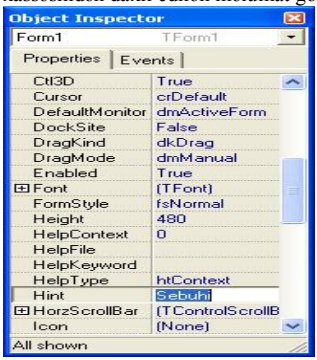
6.2.1.11.Formaların Enabled xassəsi.

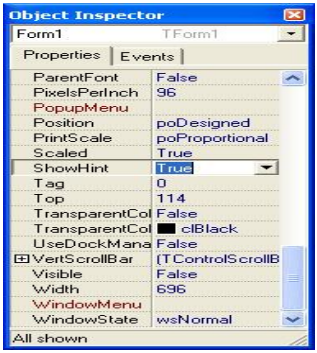
Məntiqi True/False qiymətlərini alan Enabled **xassəsi formadan istifadə olunub olunmamasını təyin edir.Bu xassədən bəzi proyektlərdə birdən artıq forma olduqda onlardan hər hansı birinin üzərində iş görülməsinə qadağa qoymaq üçün istifadə olunur.**



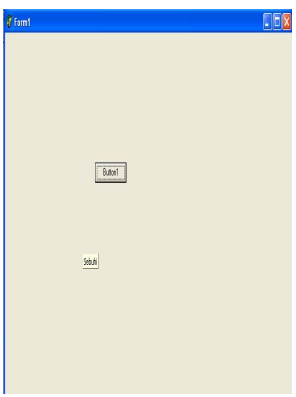
6.2.1.12 Formaların Hint **xassəsi.**

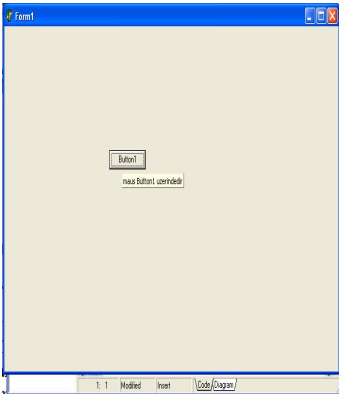
Bu xassənin köməyi ilə maus forma üzərində hər hansı bir obyektin üzərinə gətirildikdə həmin obyektin funksiyasını yadda saxlamaq fiziki cəhətdən çətinlik törətdiyindən müəyyən məlumat almaq olar.Maus obyektin üzərindən götürüldükdə bu məlumat da götürülür.Bu məqsədlə maus formanın üzərində olarkən hansı məlumatın ekrana gətirilməsini istəyiriksə, həmin məlumatı Object Inspector pəncərəsinin Hint xassəsindən daxil etmək lazımdır.Əvvəlcədən ShowHint xassəsinə true qiyməti mənimsədilməlidir.Əks təqdirdə Hint xassəsindən daxil edilən məlumat görünməz olur.





Forma üzərində bir düymə Button1 yerləşdirərək,bu düymə seçili ikən onun Hint xassəsinə "maus Button1 uzerindedir" məlumatını yazırıq.Eyni zamanda Button1-ə aid olan ShowHint xassəsinə true qiyməti mənimsədilməlidir.



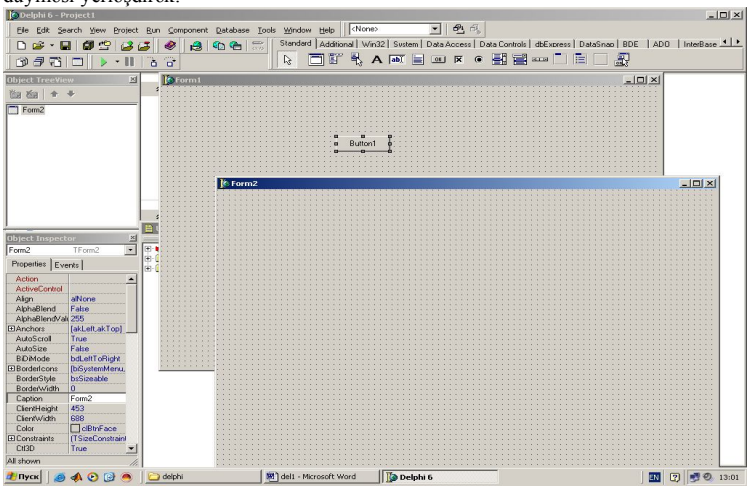


6.2.2. Metodlar

6.2.2.1. Show metodu

Cari proyektdə bir neçə formadan istifadə etdikdə onlardan birinin gizlədilib digər formanın göstərilməsi üçün bu metoddan istifadə edilir.

Cari proyektdə Form1 daxilində yeni bir forma (Form2) yerləşdirək. Form1 üzərinə Button1 düyməsi yerləşdirək.



OnClick proqramına Form2. Show sətrini əlavə edirik.Unit1-in Impletation sətrinə uses unit2; yazılaraq Unit2 vasitəsilə Unit1 daxilində Form2 təsvir olunmalıdır.

```
unit Unit1;

interface
```

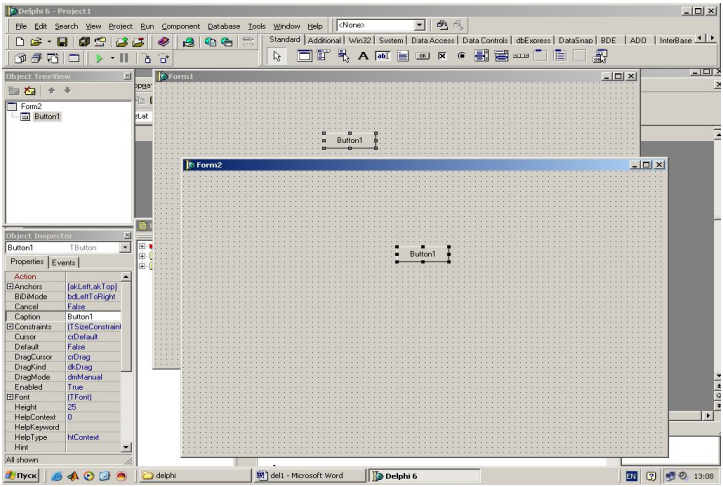
```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
   Controls, Forms,Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
uses unit2;

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
begin
Form2.show;
end;

end.
```

Əgər biz Form1-in gizlədilib Form2-nin cari olmasını istəyiriksə,yuxarıdakı proqramı işlətdikdən sonra Button1 düyməsini sıxırıq.

Bəzi hallarda istifadəçinin ekranda olan formanı qapatmadan digər formaya keçməsinə icazə verilmir. Bu halda **Show** metodu əvəzinə **showmodal** metodundan istifadə edilir.

6.2.2.2. Close metodu

Bu metodla proyektdə istifadə edilən formaları bağlamaq olar.Yuxaridaki proqramda Form2 – yə Button1 obyekti yerləşdirək.



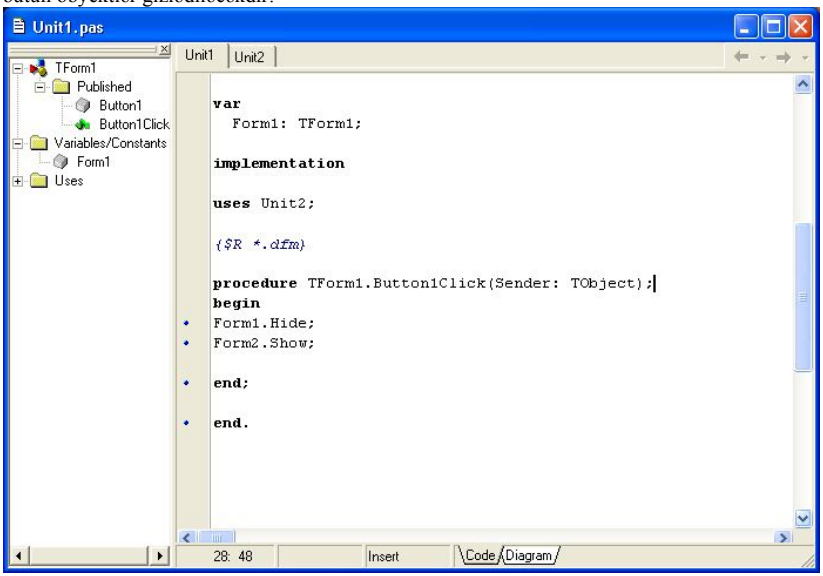
Onclick proqramına Form 2 .close yazaraq ,proqramı işlətdikdə Form1-dəki düyməni sıxdıqda Form2 görünür.Form2-dəki düyməni sıxdıqda isə Form2 bağlanır.

```
unit Unit2;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TForm2 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form2: TForm2;

implementation

{$R *.dfm}

procedure TForm2.Button1Click(Sender: TObject);
begin
Form2.Close;
end;
```

end.

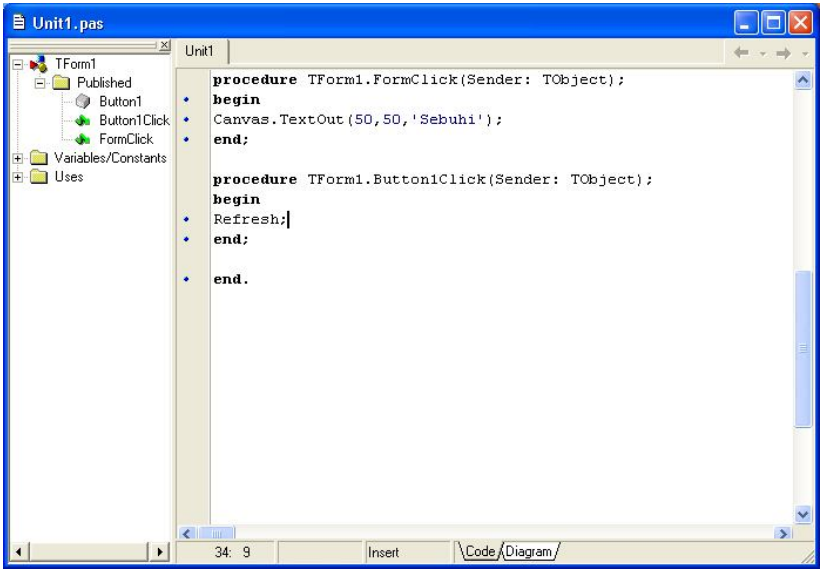
6.2.2.3. Hide metodu

Bu metod ilə forma üzərindəki obyektlər gizlədilir. Form1.Hide əmri ilə forma üzərindəki bütün obyektlər gizlədiləcəkdir.

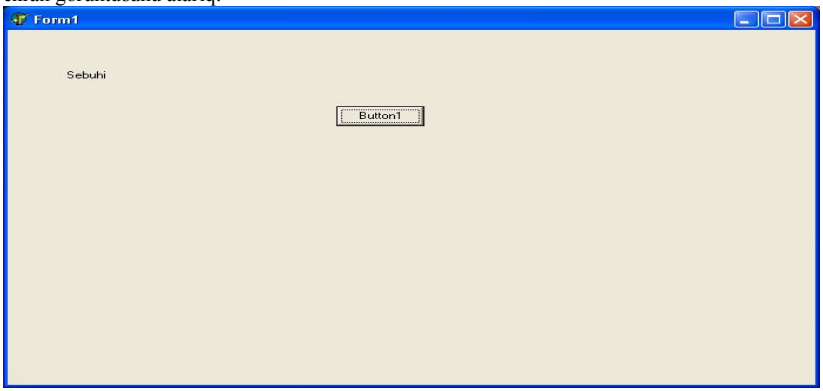


6.2.2.4. Refresh metodu

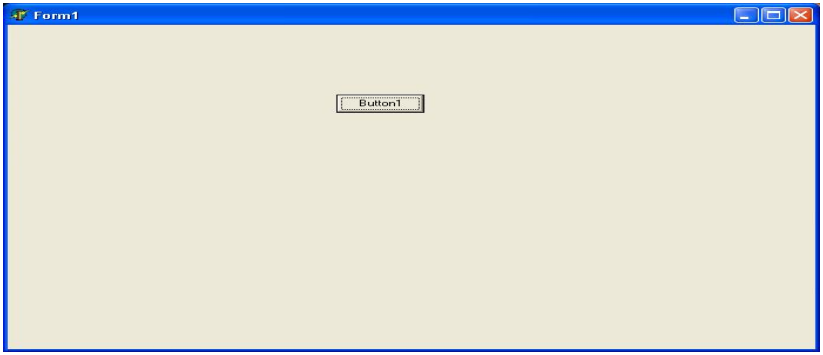
Bu metodun köməyi ilə formanın üzərini təmizləmək mümkündür. Aşağıdakı proqramda forma üzərinə TextOut metodu ilə yazılan təsvir Refresh metodu ilə təmizlənir.



Proqram işə salındıqdan sonra forma üzərində mausun sol düyməsini vurduqda aşağıdakı ekran görüntüsünü alarıq.



Button1 düyməsi üzərində mausun sol düyməsini vurduqda isə aşağıdakı ekran görüntüsünü alarıq.

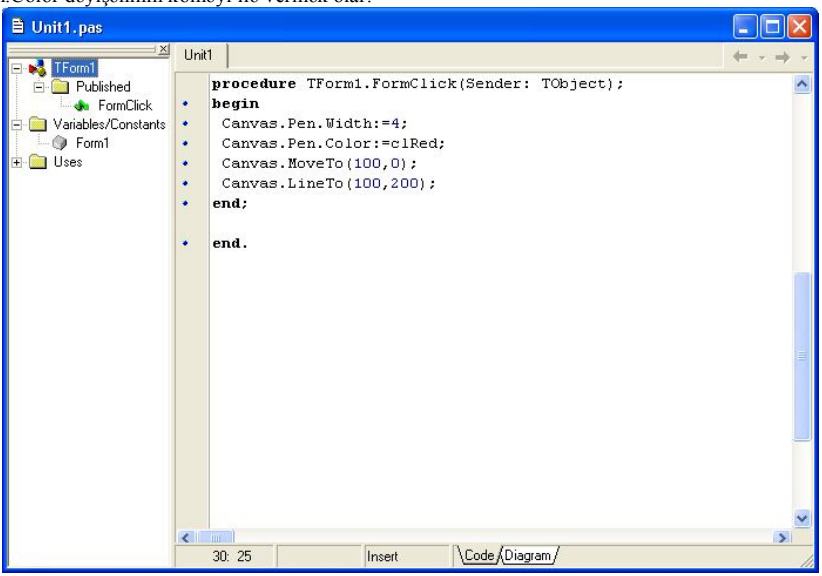


6.2.2.5. LineTo –MoveTo metodları

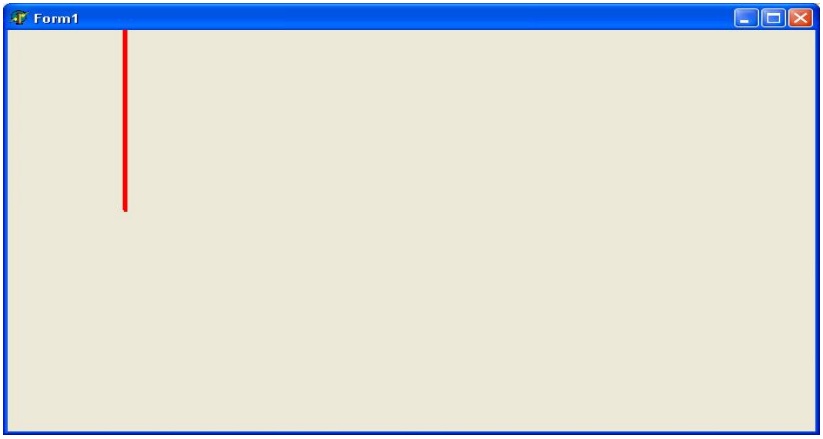
LineTO metodu forma üzərində cizgi çəkmək üçün istifadə olunaraq , kənardan iki parametr alır.

Canvas.LineTo(x,y); burada x üfqi koordinatı ,y isə şaquli koordinatı göstərir. **MoveTO( )** metodu tətbiqolunmadıqda cizginin başlanğıc koordinatı (0,0) qəbul edilir və formanın yuxarı sol küncündən cizgi çəkilməyə başlayır.

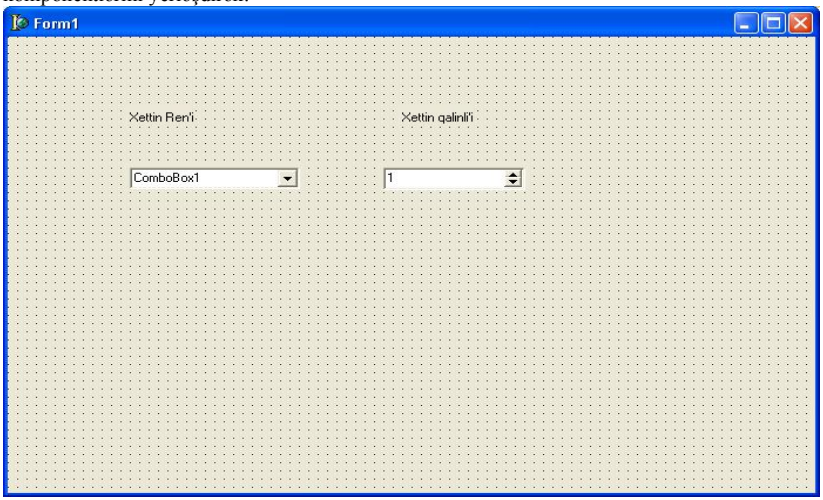
MoveTO( ) metodu tətbiq olunduqda cizgi bu koordinatlı nöqtədən çəkilməyə başlayır. Çəkilən xəttin qalınlığı **Pen.Width** dəyişəninin köməyi ilə dəyişdirilə bilər. Xəttin rəngini Pen.Color dəyişəninin köməyi ilə vermək olar.



Proqram yerinə yetirildikdən sonra forma üzərində mausun sol düyməsini vurduqda aşağıdakı ekran görüntüsü alınır.

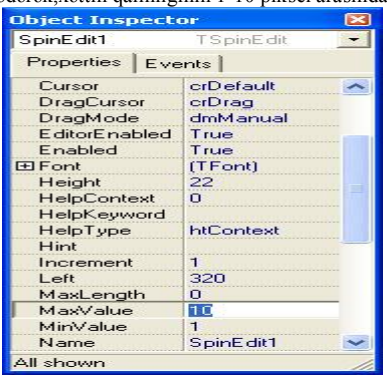


Forma üzərinə Standard səhifəsindənComboBox1 və Samples səhifəsindən SpinEdit1 komponentlərini yerləşdirək:



ComboBox1-in Items xassəsinin qarşısında … mausun düyməsini vuraraq String List Editor dialoq qutusunu açaraq,rəng adlarını qara,mavi,… aşağıdakı kimi daxil edirik.

Bütün rəngləri daxil etdikdən sonra OK düyməsinə vururuq.Sonra isə SpinEdit1 komponentini seçərək Object Inspector pəncərəsindən MinValue xassəsinə 1,MaxValue xassəsinə isə 10 qiymətini mənimsədərək,xəttin qalınlığının 1-10 piksel arasında dəyişməsini təmin edirik.

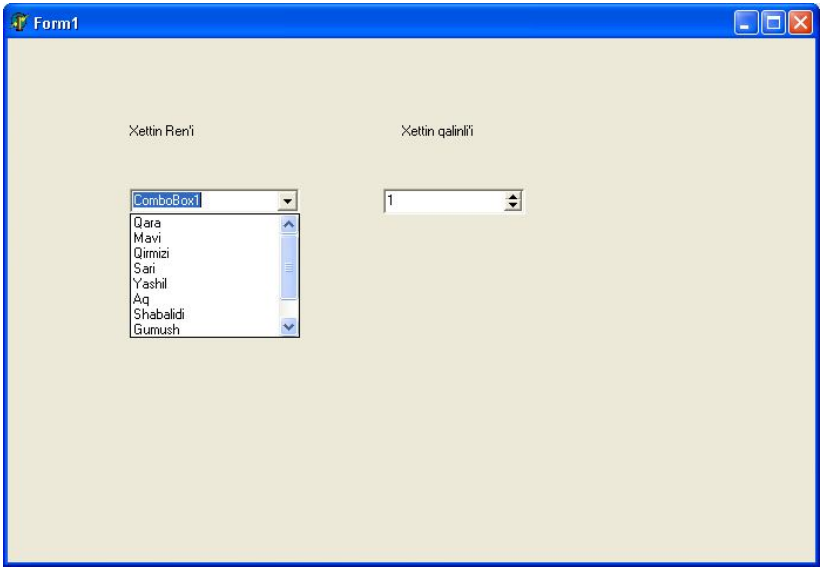


Məqsədimiz maus sıxılı olduğu halda xəttin çəkilməsinə başlanması,buraxıldıqda isə sona yetməsidir.Bildiyimiz kimi MouseDown hadisəsi mausun vurulması, MouseMove hadisəsi mausun hərəkəti zamanı, MouseUp hadisəsi isə mausun düyməsinin buraxılması zamanı baş verir.

Deyilənləri həyata keçirmək üçün A dəyişənini məntiqi tip olaraq təsvir edirik.FormMouseDown hadisəsi baş verərkən A dəyişəninə True qiyməti mənimsədilir, (x,y) koordinatlarından çəkiləcək xəttin qalınlığının SpinEdit1.Value qiyməti alacağı, ComboBox1-də isə ilk rəng(ItemIndex=0) və son rəng (ItemIndex=9) yerləşdiriləcəkdir.

```
var
  Form1: TForm1;
  A:Boolean;

implementation

{$R *.dfm}
```

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
A:=True;
Canvas.MoveTo(x,y);
Canvas.Pen.Width:=SpinEdit1.Value;
if ComboBox1.ItemIndex=0 then Canvas.Pen.Color:=clBlack;
if ComboBox1.ItemIndex=1 then Canvas.Pen.Color:=clBlue;
if ComboBox1.ItemIndex=2 then Canvas.Pen.Color:=clRed;
if ComboBox1.ItemIndex=3 then Canvas.Pen.Color:=clYellow;
if ComboBox1.ItemIndex=4 then Canvas.Pen.Color:=clGreen;
if ComboBox1.ItemIndex=5 then Canvas.Pen.Color:=clWhite;
if ComboBox1.ItemIndex=6 then Canvas.Pen.Color:=clGray;
if ComboBox1.ItemIndex=7 then Canvas.Pen.Color:=clSilver;
if ComboBox1.ItemIndex=8 then Canvas.Pen.Color:=clAqua;
if ComboBox1.ItemIndex=9 then Canvas.Pen.Color:=clTeal;

end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
   A:=False;
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
begin
 if A then Canvas.LineTo(x,y);
end;

end.
```

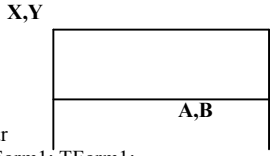


Proqram işlədilərək istənilən rəng və qalınlıqlı xətt çəkmək olar.

6.2.2.6. Rectangle(düzbucaqlı) və Ellipse metodları

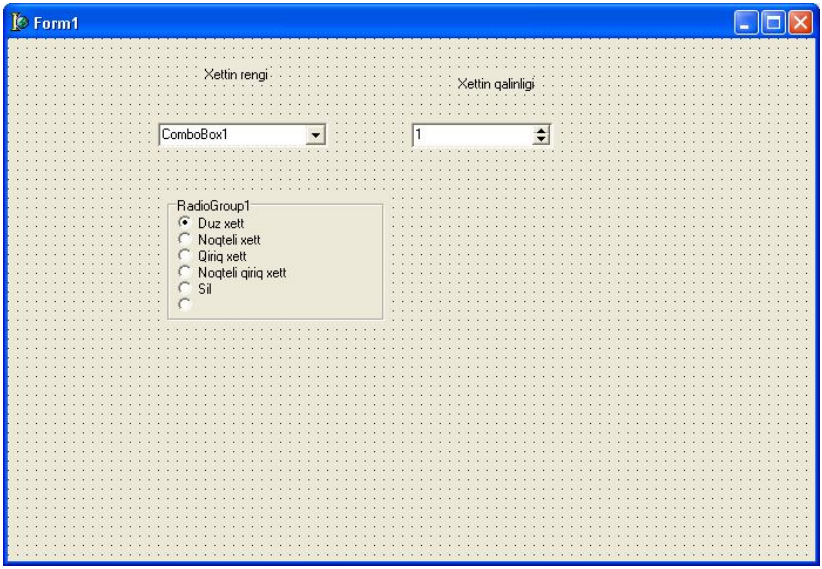
Canvas.Rectangle(X,Y,A,B) metodu ilə düzbucaqlı çəkilir. X,Y koordinatları düzbucaqlının başlanğıc , A,B koordinatları isə şəkildə göründüyü kimi son koordinatlarını müəyyən edir.Onu da qeyd edək ki,(A,B X,Y) koordinatlarından çəkilən düzbucaqlı əvvəlki ilə eyni olacaqdır.

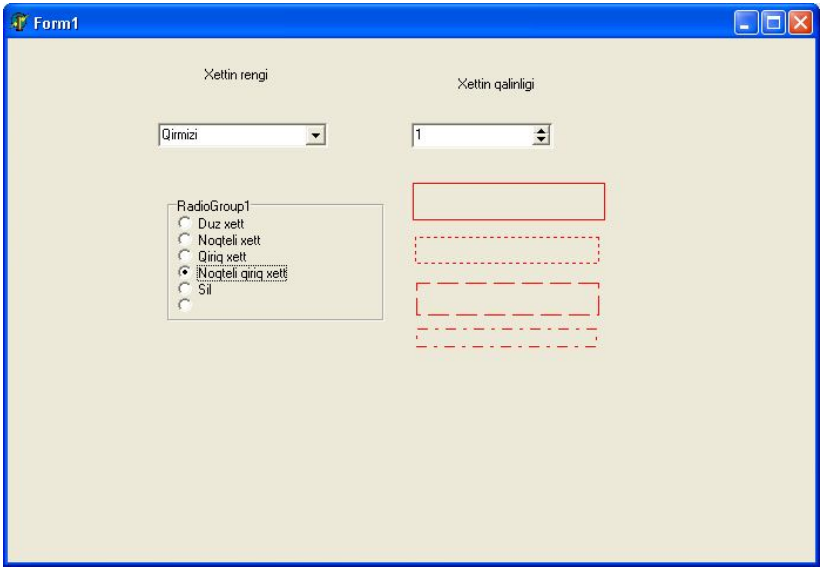
Aşağıdakı proqramda forma üzərinə RadioButton1 komponenti yerləşdirərək Object Inspector pəncərəsinin Item xassəsinin qarşısında mausun düyməsini vuraraq Düz xett…Sil yazılarını daxil edirik.RadioGroup1-in ItemIndex xassəsinə 0 qiyməti mənimsədilir. Beləliklə başlanğıcda ilk qiymət olaraq Düz xett seçiləcəkdir.




```
var
  Form1: TForm1;
  A,B:integer;

implementation

{$R *.dfm}

procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
A:=X;
B:=Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
if comboBox1.ItemIndex=0 then Canvas.pen.Color:=clblack;
if comboBox1.ItemIndex=1 then Canvas.pen.Color:=clblue;
if comboBox1.ItemIndex=2 then Canvas.pen.Color:=clred;
if comboBox1.ItemIndex=3 then Canvas.pen.Color:=clyellow;
if comboBox1.ItemIndex=4 then Canvas.pen.Color:=clgreen;
if comboBox1.ItemIndex=5 then Canvas.pen.Color:=clwhite;
if comboBox1.ItemIndex=6 then Canvas.pen.Color:=clgray;
if comboBox1.ItemIndex=7 then Canvas.pen.Color:=clsilver;
if comboBox1.ItemIndex=8 then Canvas.pen.Color:=claqua;
if comboBox1.ItemIndex=9 then Canvas.pen.Color:=clteal;
if RadioGroup1.Itemindex=0 then Canvas.Pen.style:=psSolid;
if RadioGroup1.Itemindex=1 then Canvas.Pen.style:=psDot;
```

```
if RadioGroup1.Itemindex=2 then Canvas.Pen.style:=psDash;
if RadioGroup1.Itemindex=3 then Canvas.Pen.style:=psDashDot;
if RadioGroup1.Itemindex=4 then Canvas.Pen.style:=psClear;
Canvas.pen.Width:=SpinEdit1.value;
Canvas.Rectangle(X,Y,A,B);
end;

end.
```

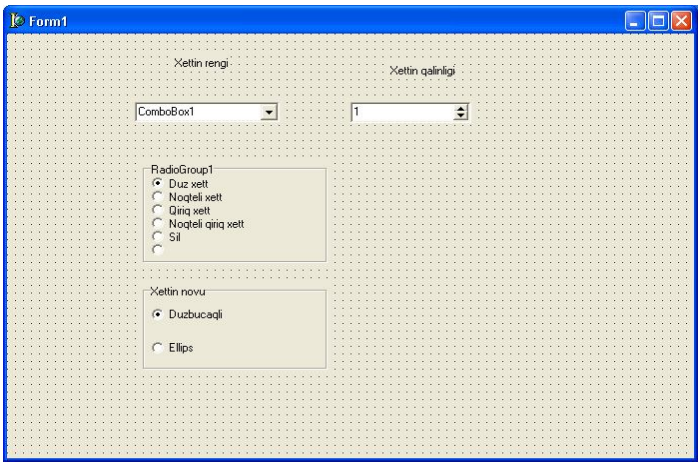


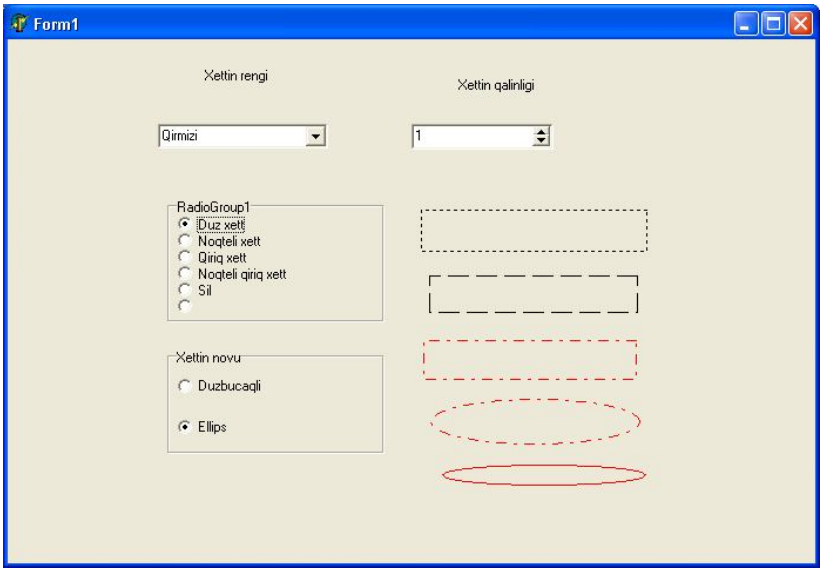


Canvas.Ellipse(X,Y,A,B); metodu ilə A,B koordinatlarından , X,Y koordinatlarına ellirs çəkilir.(Bu koordinatlar bir düzbucaqlını ifadə edir.Ellips bu düzbucaqlı daxilinə çəkilir,düzbucaqlı özü isə görünmür.)

Formaya ikinci bir RadioGroup komponenti yerləşdirərək,Items xassəsinə Düzbucaqlı və Ellips yazırıq.

ItemIndex xassəsinə 0 qiymətini mənimsədərək,aşağıdakı formanı alarıq.



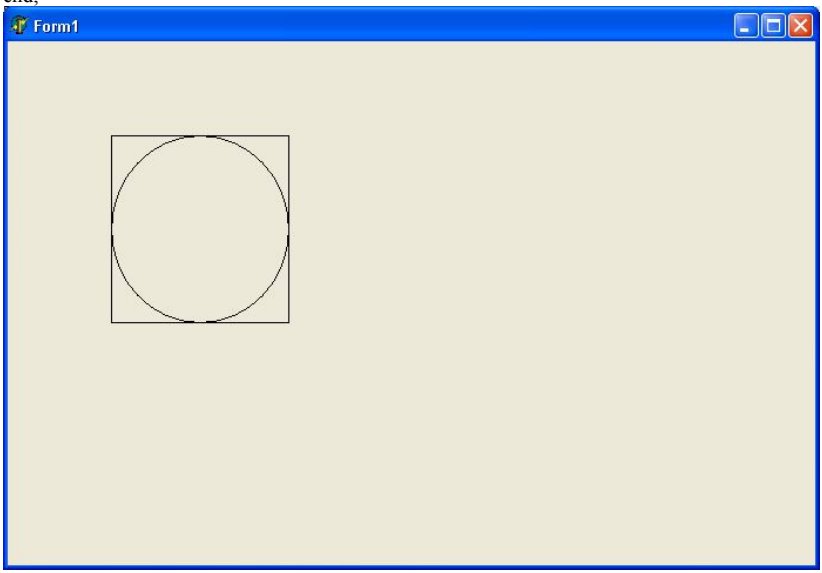



```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls, Spin;

type
  TForm1 = class(TForm)
    SpinEdit1: TSpinEdit;
    Label1: TLabel;
    Label2: TLabel;
    ComboBox1: TComboBox;
    RadioGroup1: TRadioGroup;
    RadioGroup2: TRadioGroup;
    procedure FormMouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure FormMouseUp(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);


  private
    { Private declarations }
  public
    { Public declarations }
```

```
  end;

var
  Form1: TForm1;
  A,B:integer;


implementation

{$R *.dfm}

procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
A:=X;
B:=Y;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin


if comboBox1.ItemIndex=0 then Canvas.pen.Color:=clblack;
if comboBox1.ItemIndex=1 then Canvas.pen.Color:=clblue;
if comboBox1.ItemIndex=2 then Canvas.pen.Color:=clred;

if comboBox1.ItemIndex=3 then Canvas.pen.Color:=clyellow;
if comboBox1.ItemIndex=4 then Canvas.pen.Color:=clgreen;
if comboBox1.ItemIndex=5 then Canvas.pen.Color:=clwhite;
if comboBox1.ItemIndex=6 then Canvas.pen.Color:=clgray;
if comboBox1.ItemIndex=7 then Canvas.pen.Color:=clsilver;
if comboBox1.ItemIndex=8 then Canvas.pen.Color:=claqua;
if comboBox1.ItemIndex=9 then Canvas.pen.Color:=clteal;
if RadioGroup1.Itemindex=0 then Canvas.Pen.style:=psSolid;
if RadioGroup1.Itemindex=1 then Canvas.Pen.style:=psDot;
if RadioGroup1.Itemindex=2 then Canvas.Pen.style:=psDash;
if RadioGroup1.Itemindex=3 then Canvas.Pen.style:=psDashDot;
if RadioGroup1.Itemindex=4 then Canvas.Pen.style:=psClear;
if RadioGroup2.ItemIndex=0 then Canvas.Rectangle(X,Y,A,B);
if RadioGroup2.ItemIndex=1 then Canvas.Ellipse(X,Y,A,B);
  Canvas.pen.Width:=SpinEdit1.value;
end;




end.
```

Aşağıdakı misalda (240,240,88,80) koordinatları və (88,80,240,240) koordinatlarında ellips və düzbucaqlı çəkilmişdir.

```
procedure TForm1.FormClick(Sender: TObject);
begin
Canvas.Ellipse(240,240,88,80);
Canvas.Ellipse(88,80,240,240);
end;

procedure TForm1.FormDblClick(Sender: TObject);
begin
 Canvas.Rectangle(240,240,88,80);
Canvas.Rectangle(88,80,240,240);
end;
```



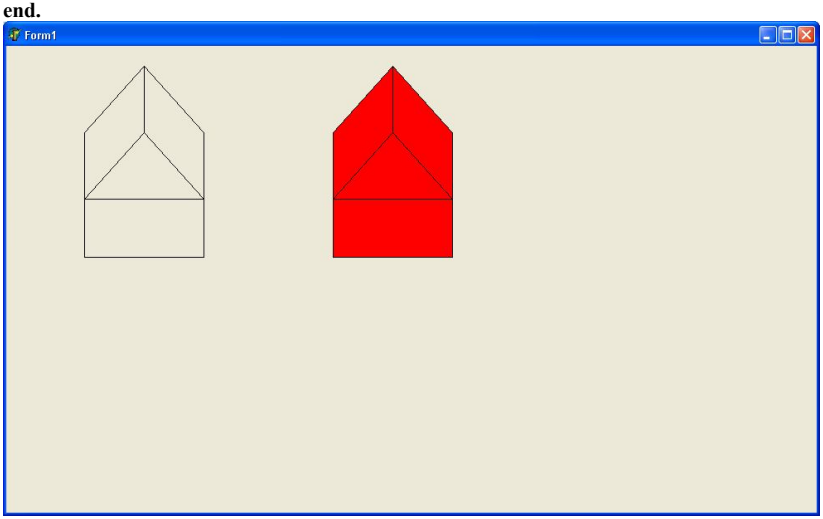
Təpələri oval düzbucaqlı:
Canvas.RoundRect(X,Y,A,B,M,N);
Düzbucaqlıdan fərqi M,N olaraq 5-ci və 6-cı parametrin verilməsidir.Bu parametrlər ilə düzbucaqlının təpə nöqtələrinin ovallığı müəyyən edilir.

6.2.2.7 Polygon-PolyLine metodları.

Canvas.PolyLine([Point(X1,Y1),Point(X2,Y2),.....Point(Xn,Yn)]);
X1,Y1 koordinatlarından X2,Y2 koordinatlarına , X2,Y2 koordinatlarından bir sonrakı koordinatlara düz xətt çəkilir. Polygon metodu ilə çəkilən xətlərin içərisi doldurulur.

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
Canvas.Polyline([Point(94,184),Point(94,104),Point(166,24),Point(238,104),
point(238,184),point(166,104),point(94,184),point(238,184),
point(166,104),point(166,24),point(166,104),point(94,184),point(94,254),
point(238,254),point(238,184)]);
```

```
Canvas.Brush.Color:=clred;
Canvas.Polygon([Point(394,184),POint(394,104),Point(466,24),Point(538,104),
point(538,184),point(466,104),point(394,184),point(538,184),
point(466,104),point(466,24),point(466,104),point(394,184),point(394,254),
point(538,254),point(538,184)]);
end;
end.
```



6.2.2.8Canvas.Pie-Canvas.Arc-Canvas.Chord metodları

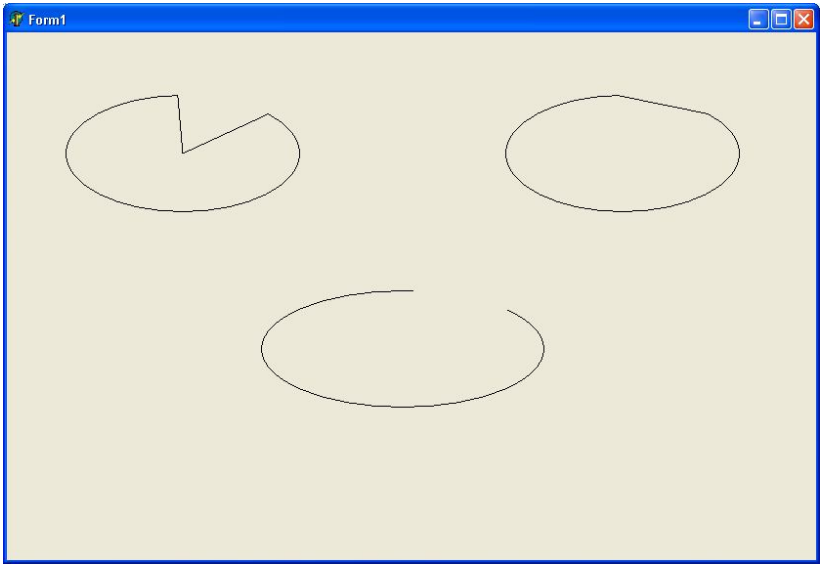
Canvas.Pie(X,Y,A,B,K,L,M,N);

Bu metod ilə X,Y koordinatları ilə A,B koordinatlarına çəkilən görünməz düzbucaqlı içərisinə K,L və M,N koordinatlarından dairə mərkəzinə düz xət çəkilərək qapalı bir yay alınır.

Canvas.Arc(X,Y,A,B,K,L,M,N); Bu metod Canvas.Pie metoduna oxşar olub ondan fərqi çəkilən yayın uclarının açıq olmasıdır.

Canvas.Chord(X,Y,A,B,K,L,M,N); Bu metod ilə X,Y koordinatları ilə A,B koordinatlarına çəkilən görünməz düzbucaqlı içərisinə K,L və M,N koordinatları düz xətlə birləşdirilərək dairə çəkilir.

```
procedure TForm1.FormClick(Sender: TObject);
begin
Canvas.Pie(60,64,300,184,172,32,292,72);
Canvas.Arc(260,264,550,384,422,232,542,272);
Canvas.Chord(510,64,750,184,622,32,742,72);
end;

end.
```



6.2.3. Formanın obyektlər saxlancına yazılması

Lazım olan formanı File▶New ▶Application əmrlərinin köməyi ilə yaradıb onu obyektlər ambarına yerləşdiririk. Bunun üçün formanı yaratdıqdan sonra File menyusundan Save as əmrini seçirik. Sonra həmin proyekti Delphi6 kataloqunda yerləşən Objrepos kataloquna yerləşdiririk.

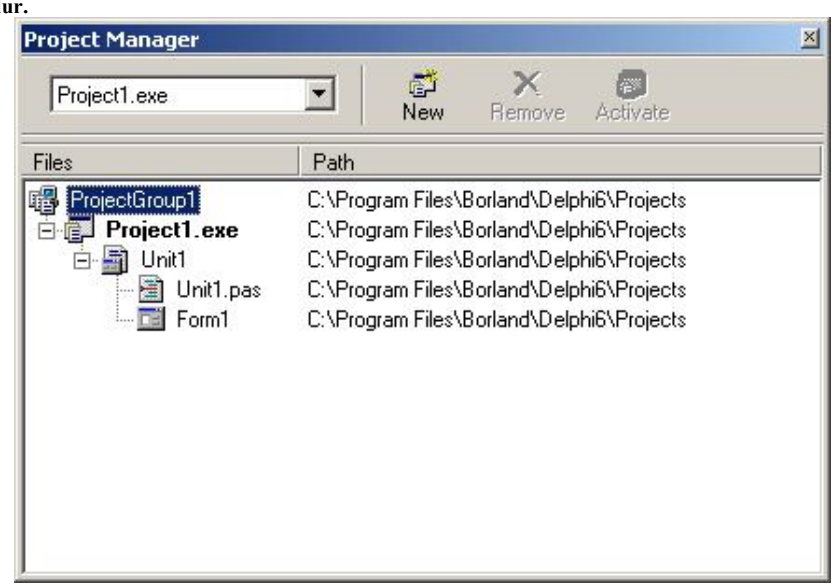
C:/Program Files/ Borland/Delphi 6

Formanın Objrepos kataloquna yerləşdirilməsi heçdə formanın arxivə yerləşdirilməsini təmin etmir. Formanın arxivdə qeydiyyatını təmin etmək üçün forma üzərində mausun sağ düyməsini vuraraq açılan kontekst menyudan Add to Repozitory əmrini seçirik. Title sahəsinə formanın adını, Description sahəsinə izahedici mətn daxil edilir. Sonra Page siyahısını açaraq oradan Forms bəndini seçirik. Qeydiyyat pəncərəsini OK düyməsini sıxmaqla bağlamaq olar. Indi artıq forma qeyd olunmuşdur. Istənilən vaxt ona müraciət etmək üçün File▶New▶Other seçmək lazımdır.

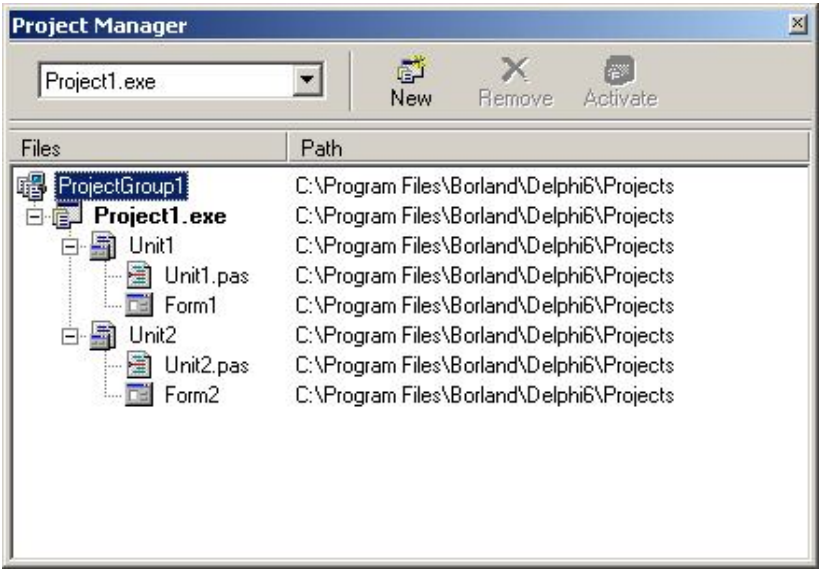
Kompüterdə Delphi mühiti işə salınan zaman biz monitorda lazım gələn formanı almaq istəyiriksə, əsas menyuda Tools alətlər panelindən Repozitory... əmrini seçirik. Pages siyahısından Forms seçib mausun sol düyməsini basırıq. Sağ tərəfdə Obyects siyahısından lazım olan formanı seçib, MainForm qarşısında olan işarədə mausun sol düyməsini vururuq. Ok düyməsini basıb obyektlər repozitorisindən çıxırıq. Delphi mühitini yenidən isə saldıqda artıq forma pəncərəsində bizə lazım olan forma görünəcəkdir.

6.3. Delphi proyektlərinin elementləri

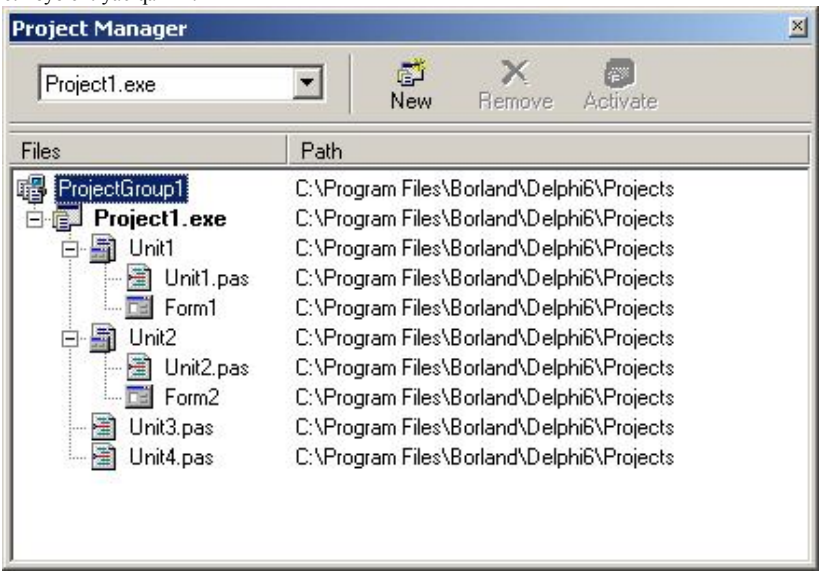
Delphi proqramları və proyektlərinin ən əsas elementləri Formalardır.Delphi proyektlərində formadan başqa pas genişlənməli Pascal proqramı daxilinə alan fayllar mövcuddur.



Pas uzantılı proqramı özündə saxlayan fayllara Delphi daxilində UNIT adı verilir.Delphi proyektində bir neçə pas genişlənməli UNIT faylı və ya forma ola bilər.Aşağıda göstərilən ekran təsviri Project1 adlı proyektə Form2 adında ikinci bir forma yerləşdirildikdən sonra alınmışdır.Təsvirdən göründüyü kimi proyektə avtomatik olaraq ikinci bir UNIT qoşulur.



Buradan bu nəticəyə gəlirik ki,Delphi ilə hazırlanan proyektdəki hər Forma üçün proyektdə ən azı bir UNIT olmalıdır.Proqramlarda formalardan çox sayda da UNIT ola bilər.Delphi proyektlərinə daxil edilən UNIT-lərin hər biri diskə pas uzantılı fayla yazılır.Delphi proyektləri sərt diskə qeyd olunana qədər proyektə daxil edilmiş hər forma üçün avtomatik olaraq dfm uzantılı forma faylı hazırlanır.Proyektdəki formaları daxilinə alan faylları diskə qeyd etməyə ehtiyac qalmır.



Delphi ilə hazırlanan fayllar diskə dpr uzantısı ilə qeyd olunurlar.

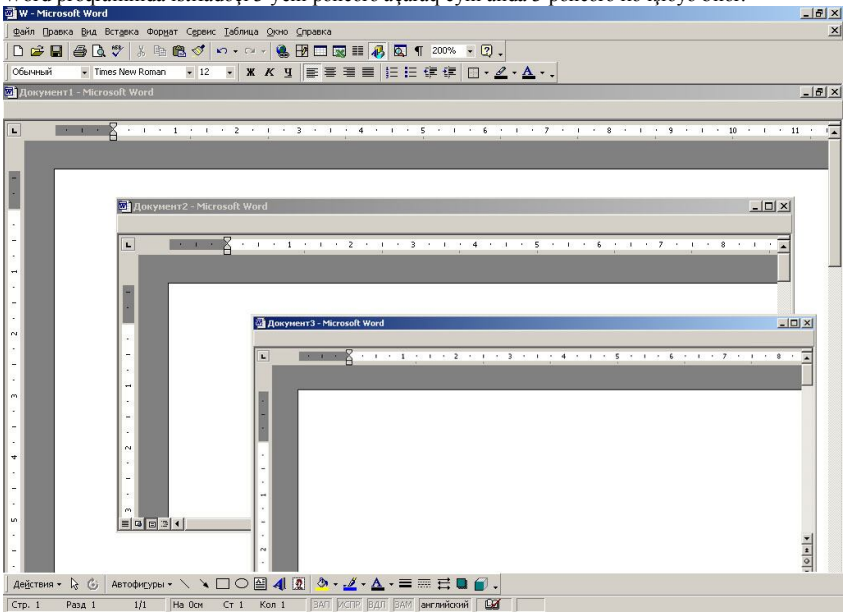
```
program Project1;

uses
        Forms,
  Unit1 in 'Unit1.pas' {Form1},
  Unit2 in 'Unit2.pas' {Form2},
  Unit3 in 'Unit3.pas',
  Unit4 in 'Unit4.pas';

{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.CreateForm(TForm2, Form2);
  Application.Run;
end.
```

6.4 MDI-SDI Forma əlavələri.

Delphi daxilində MDI(Multiple Document Interface) SDI(single Document Interface) əlavələri yaratmaq mümkündür.

6.4.1 MDI Forma əlavələri.

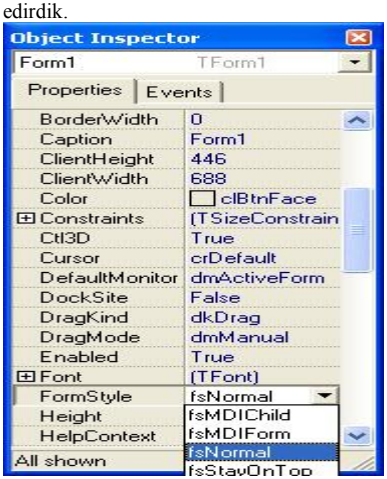
MDI əlavələrinə misal olaraq Word və ya Excel proqramlarını göstərmək olar.Aşağıdakı Word proqramında istifadəçi 3 yeni pəncərə açaraq eyni anda 3 pəncərə ilə işləyə bilər.



Burada Word pəncərəsi əsas pəncərə və bu pəncərə içərisində yer alan Pen1,Pen2 və Pen3 isə alt pəncərələrdir.Alt(Child)pəncərələr yalnız əsas pəncərələr içərisində hərəkət edə bilər.Əsas

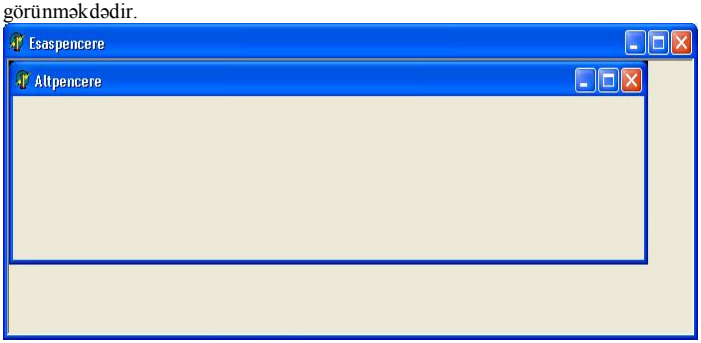
pəncərələrin daşınması ilə balt pəncərələr də onunla birlikdə daşınır.MDI işləmələrində alt pəncərələr də əsas pəncərələr kimi kiçiltmə,böyütmə və qapatma düymələrinə malik olub öz aralarında Tile və Cascade metodları ilə düzlənirlər.
Yuxarıdakı Word proqramında eyni anda iki pəncərənin aktiv olduğunu görürük. MDI işləmələrinin gətirdiyi yenilik ondan ibarətdir ki,birdən çox alt pəncərə işlənən proyektlərdə bir formadan digərinə keçmək və eyni anda bütün pəncərələrin açıq olması(yalnız birisi aktiv ola bilir)mümkündür.

MDI işləmələrində FormStyle xassəsi ilə formanın əsas və ya alt pəncərə olması təyin edilir.Bu vaxta qədər işlədiyimiz proyektlərdə fsNormal xassəsinə malik formalardan istifadə edirdik.



MDI xassəsi veriləcək formanın FormStyle xassəsinə fsMDIForm qiyməti mənimsədilərkən, Child(alt) forma üçün fsMDIChild qiyməti mənimsədilir.

- Indi yeni bir proyekt hazırlayaraq FormStyle xassəsinə fsMDIForm qiyməti mənimsədərək,Caption xassəsinə Esas pencere yazırıq.
- Sonra isə proyektə yeni bir forma yerləşdirmək üçün File menyusundan New Form əmrini verərək ikinci formanın FormStyle xassəsinə fsMDIForm qiyməti mənimsədərək, Caption xassəsinə Altpencere adını veririk.
- Proqram bu halında işə salınarkən alt formanın əsas forma içərisində hərəkət etdiyi görünməkdədir.



Yuxarıda proyektə daxil etdiyimiz Esaspencere və Altpencere1 başlığına, FormStyle xassəsi fsMDIForm və fsMDIChild qiymətlərinə malik proyektdə əsas pəncərə üzərinə Button1 komponenti yerləşdirərək, Caption xassəsinə Yeni pencere yazaraq,OnClick proqramını aşağıdakı kimi nizamlayırıq.

```
 implementation
uses unit2;

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var
  AltPencere:TForm2;
begin
 AltPencere:=TForm2.Create(Application);
 AltPencere.Caption:='AltPencere'+IntToStr(MDIChildCount);

end;

end.
```

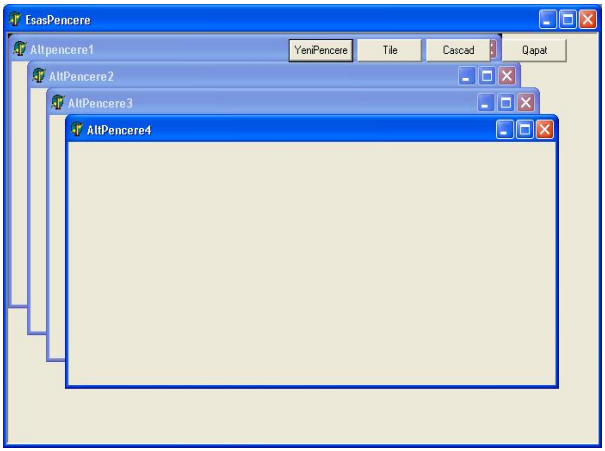
Alt forma hazırlamaq üçün TForm2 tipində bir AltPencere dəyişəni təsvir olunur. Form2-ni Unit1–lə əlaqələndirmək üçün implementation sətrinə uses unit2; proqram sətri əlavə olunur.Unit2-nin Type bölməsində Delphi tərəfindən TForm2=class(TForm) əmr sətri ilə class() metodu ilə hazırlanmış obyekt tipli dəyişən kimi təsvir olunaraq create metodu ilə child xassəsinə malik alt forma yaradılır.

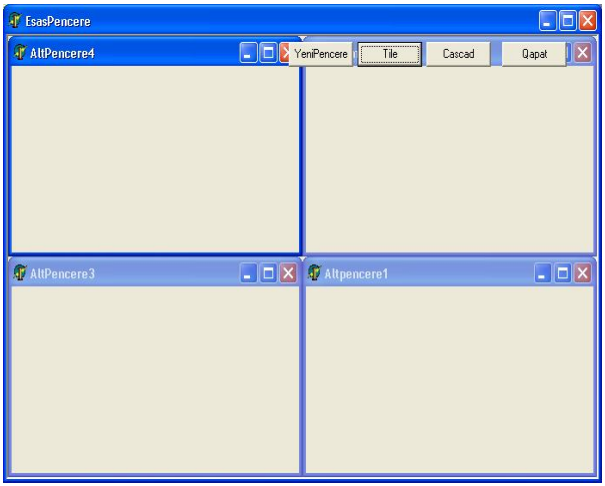
Button1 –in hər vurulmasından alt pəncərələrin sayı bir vahid artdığından AltPencere1, AltPencere2, AltPencere3…və sair alınacaqdır.

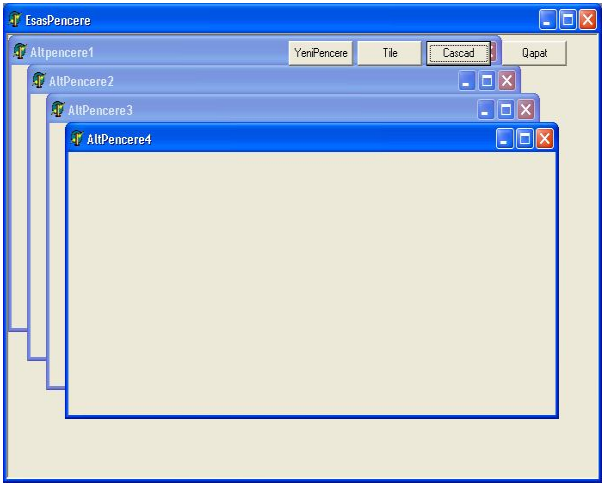
Form1 üzərinə 4 ədəd Tbutton obyekti yerləşdirərək OnClick hadisəsinə aşağıdakı proqram kodlarını yazırıq.

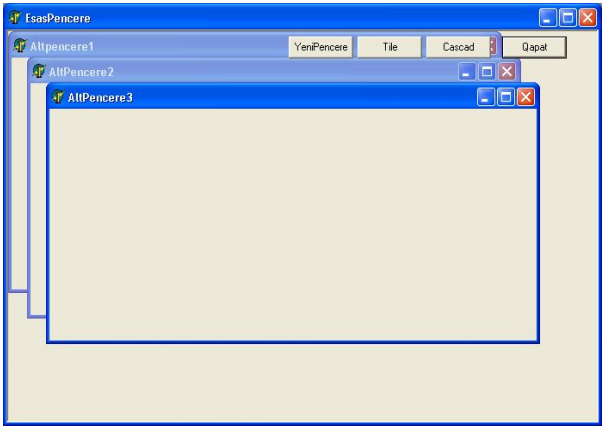
```
  implementation
uses unit2;

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var
  AltPencere:TForm2;
begin
 AltPencere:=TForm2.Create(Application);
 AltPencere.Caption:='AltPencere'+IntToStr(MDIChildCount);

end;

procedure TForm1.Button2Click(Sender: TObject);
begin
Tile;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
Cascade;
end;
```

```
procedure TForm1.Button4Click(Sender: TObject);
begin
if ActiveMDIChild<>Nil then ActiveMDIChild.Release;
end;

end.
```









6.4.2. SDI Forma əlavələri.

Bu əlavələrdə birdən çox formadan istifadə olunur,ancaq burada alt pəncərələr və bu pəncərələri daxilinə alan əsas pəncərə yoxdur.Hər bir pəncərə sərbəst olaraq hərəkət etdirilərək ekranda istənilən yerdə yerləşdirilə bilər.Alt pəncərə əsas pəncərənin sərhədləri xaricinə çıxa bilməz.Bu tip işləmələrdə alt pəncərələri daxilinə alan əsas pəncərə yoxdur.Lakin bu işləmələrdə də əsas pəncərə məfhumu mövcuddur.Buradakı əsas pəncərə bağlandıqda digər pəncərələr də bağlanacaqdır.

6.5. Standard paneli

6.5.1. Frame(Şablon) komponenti

Təyinatı.Frame komponenti formaya oxşar olub, onunla birlikdə ayrıca element kimi və özü müstəqil pəncərə kimi istifadə oluna bilər.Freymin formadan fərqi onun digər freymdə və ya formada yerləşdirilə bilməsidir.

Freym obyektlər qrupunun çevik və effektiv birləşdirilməsi və saxlanması üçün nəzərdə tutulmuşdur. Ondan lazimi struktura üçün yeni freymlər yaradılması üçün şablon kimi istifadə etmək olar. Freymlər qeyri-məhdud sayda bir-birinin daxilində ola bilərlər. Əgər şablonda dəyişiklik olunarsa, onda avtomatik olaraq onun əsasında yaranmış bütün freymlərdə dəyişiklik olunacaqdır. Freymdən palitra kimi istifadə etməklə (çox pancərəli əlavələr yaratmaqla) müxtəlif obyektlər üçün daşıma və qoşma rejimlərinin köməyi ilə mürəkkəb çöxpəncərəli əlavələr yaratmaq olar.

Yaradılması.Başlanğıcda proyektə əvvəldən yaradılmış freymlər barədə heç nə məlum deyil, ona görə də boş formada freym komponentini yerləşdirsək aşağıdakı məlumat çıxacaqdır.

No frames in project
To create a frame select File /New/Frame

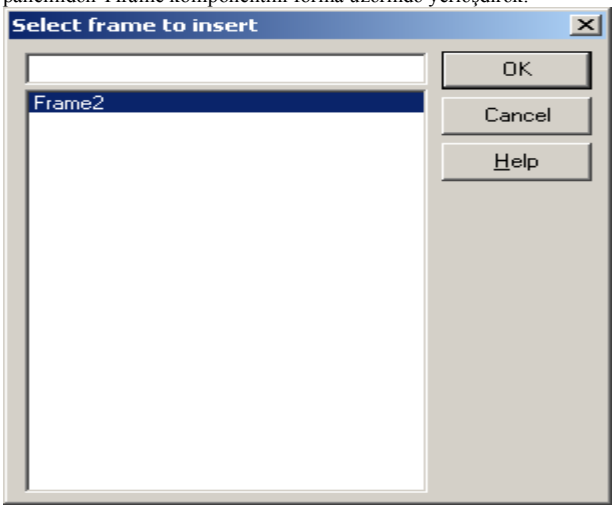
(Proyektdə freymlər yoxdur. Yeni freym faratmaq üçün File/New/Frame əmrini verin).
Bu məlumat və şablonun yerləşdirilməsi üçün aşağıda təsvir olunan yeganə metodika, şablonlardan istifadə mexanizminin digər standart komponentlərin istifadəsindən fərqləndiyini göstərir.

Tarix və zamanı göstərəcək bir freym yaradaq.Bunun üçün freymin üzərinə iki ədəd Label və bir ədəd Timer komponenti yerləşdirək.Tarix və zamanı nişan komponentlərində göstərmək üçün Timer-in OnTimer hadisəsinin emaledicisinə aşağıdakı proqram kodlarını yazaq.

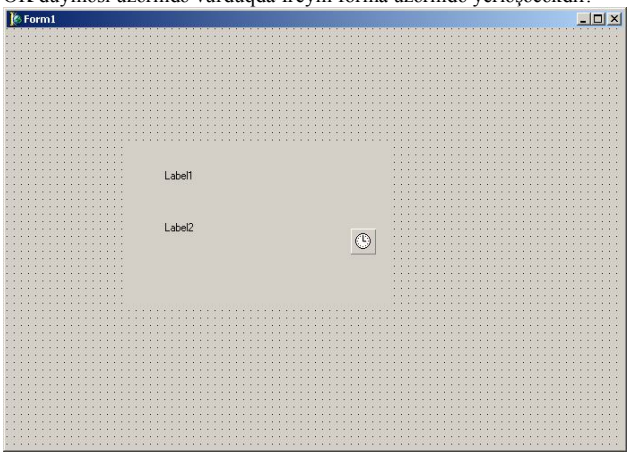
```
procedure TFrame2.Timer1Timer(Sender: TObject);
begin
```

```
Label1.Caption:='Saat'+TimeToStr(Time);
Label2.Caption:='Tarix'+DateToStr(Date);
end;
```

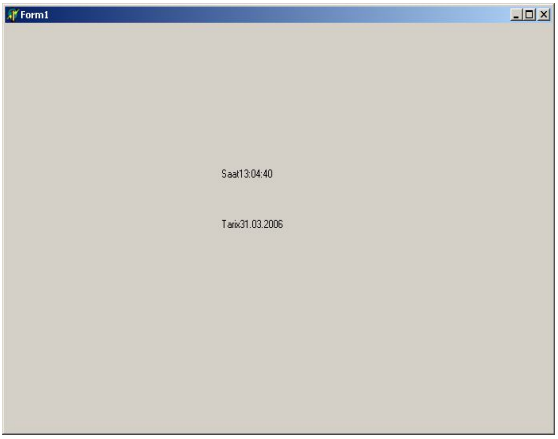
Freymi qeyd etdikdən sonra yenidən formaya qayıdırıq.Komponentlər palitrasının Standart panelindən Tframe komponentini forma üzərində yerləşdirək.



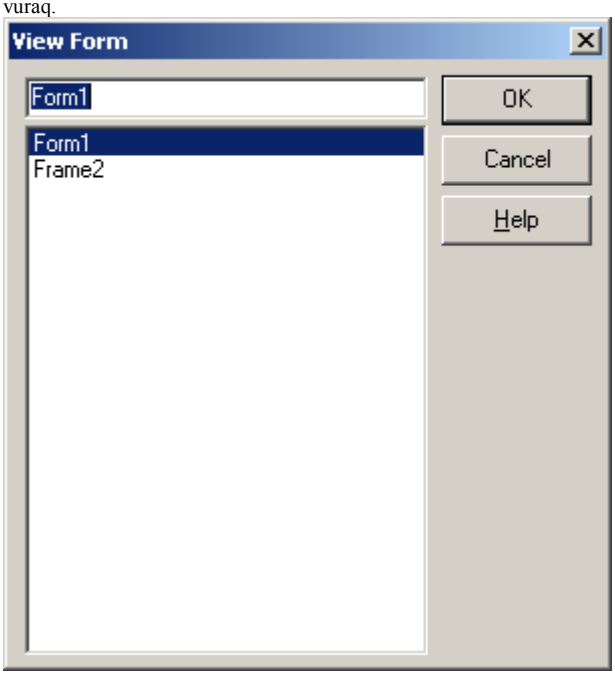
OK düyməsi üzərində vurduqda freym forma üzərində yerləşəcəkdir.

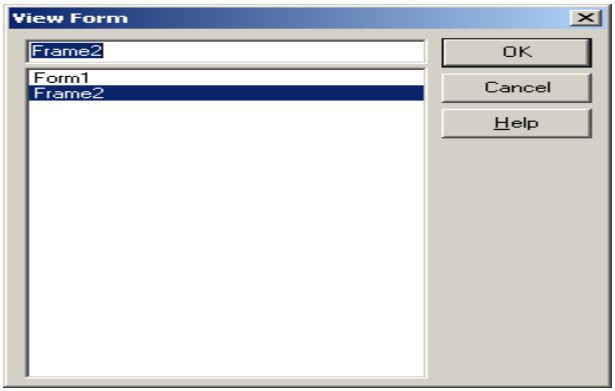


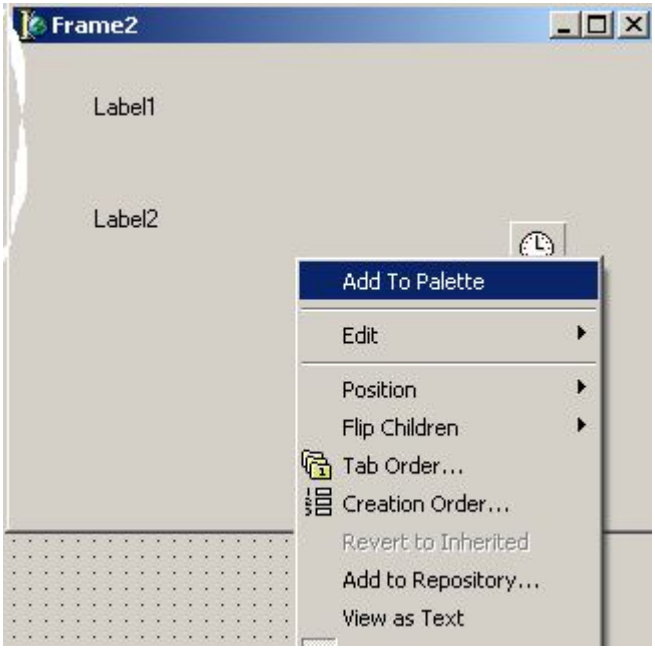
Proyekti işə salsaq,aşağıdakı pəncərəni alarıq.



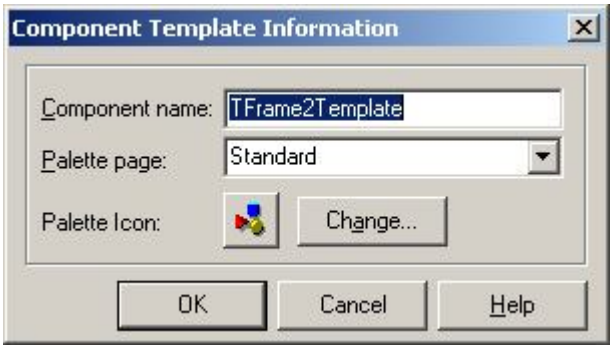
Gördüyümüz kimi freym üzərində yerləşən komponentlər və proqram kodları özlərini yeni bir komponent kimi aparırlar.Freym pəncərəsinə keçərək Freym üzərində mausun sağ düyməsini vuraq.



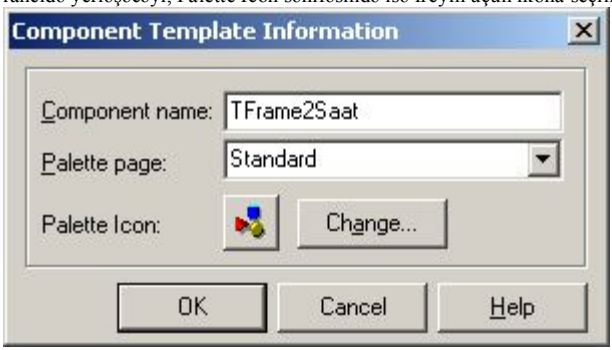




Açılan menyudan Add To Palette əmrini seçirik.
Açılan pəncərədə freymimizə aid xassələrin verilməsi tələb olunacaqdır.



Component Name səhifəsində freymimizə ad ,Palette page səhifəsində komponentin hansı raneldə yerləşəcəyi, Palette Icon səhifəsində isə freym üçün ikona seçilir.



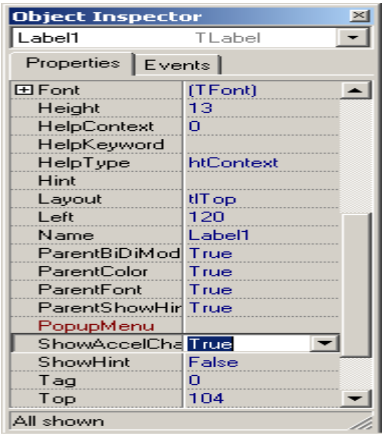
OK düyməsini sıxdıqda freym komponentlər palitrasının Standart panelində görünəcəkdir.

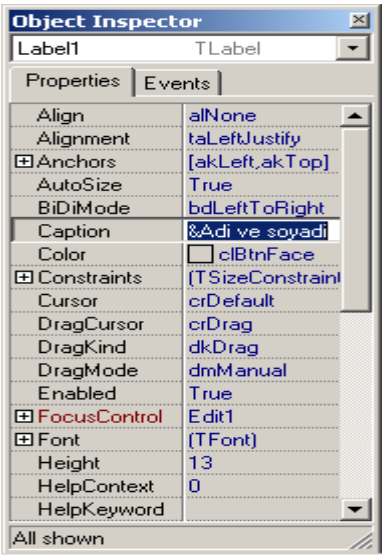
6.5.2. Nişan komponenti (TLabel)

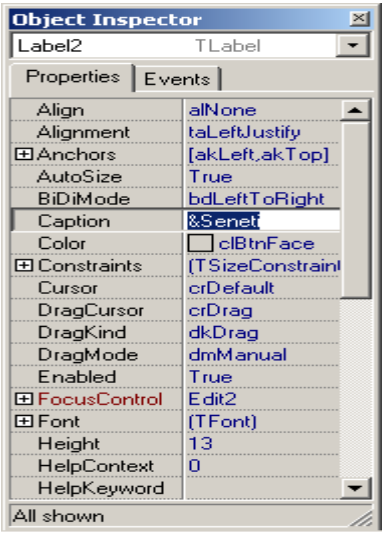
Təyinatı. Bu görünən komponent formada mətn informasiyasını təsvir etmək üçündür. Onun köməyi ilə göstərilən mətn Caption xassəsi ilə verilir. Bu komponentdən çox sətirli mətnləri daxil etmək üçün də istifadə etmək olar. Bu zaman yeni sətrə keçmək üçün ^M kombinasiyasından istifadə etmək olar. Məsələn,

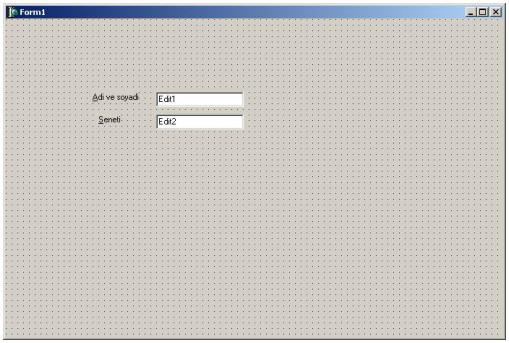
Label1.Caption:=sətir1^ Msətir2;

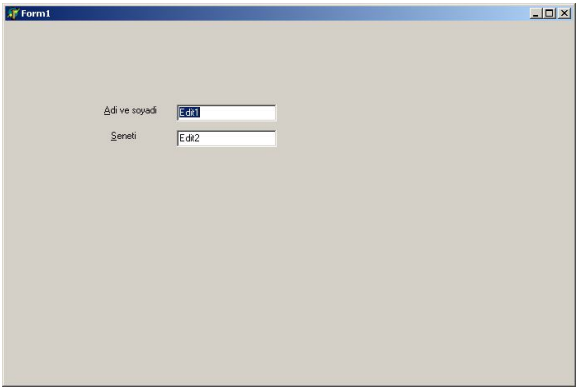
ShowAccelChar xassəsinə True qiyməti verilərsə, & işarəsindən qısa yol düyməsi kimi istifadə etmək olar.Alt+A düyməsi sıxıldıqda Edit1-ə, Alt+S düyməsi sıxıldıqda isə Edit2-yə keçilməsi üçün Label1-in FocusControl xassəsinə Edit1 ,Label2-nin FocusControl xassəsinə isə Edit2 qiymətləri verilməlidir.


Object Inspector
Label1 TLabel
Properties Events
Font (TFont)
Height 13
HelpContext 0
HelpKeyword
HelpType htContext
Hint
Layout tlTop
Left 120
Name Label1
ParentBiDiMod True
ParentColor True
ParentFont True
ParentShowHir True
PopupMenu
ShowAccelCha True
ShowHint False
Tag 0
Top 104
All shown


Object Inspector
Label1 TLabel
Properties Events
Align alNone
Alignment taLeftJustify
Anchors [akLeft,akTop]
AutoSize True
BiDiMode bdLeftToRight
Caption &Adi ve soyadi
Color clBtnFace
Constraints (TSizeConstrain
Cursor crDefault
DragCursor crDrag
DragKind dkDrag
DragMode dmManual
Enabled True
FocusControl Edit1
Font (TFont)
Height 13
HelpContext 0
HelpKeyword
All shown


Object Inspector
Label2 TLabel
Properties
Events
Align alNone
Alignment taLeftJustify
Anchors [akLeft,akTop]
AutoSize True
BiDiMode bdLeftToRight
Caption &Seneti
Color clBtnFace
Constraints (TSizeConstrain
Cursor crDefault
DragCursor crDrag
DragKind dkDrag
DragMode dmManual
Enabled True
FocusControl Edit2
Font (TFont)
Height 13
HelpContext 0
HelpKeyword
All shown


Form1
Adi ve soyadi
Edit1
Seneti
Edit2



Alt+S düymələrini sıxdıqda Edit2 –yə keçid təmin olunur.

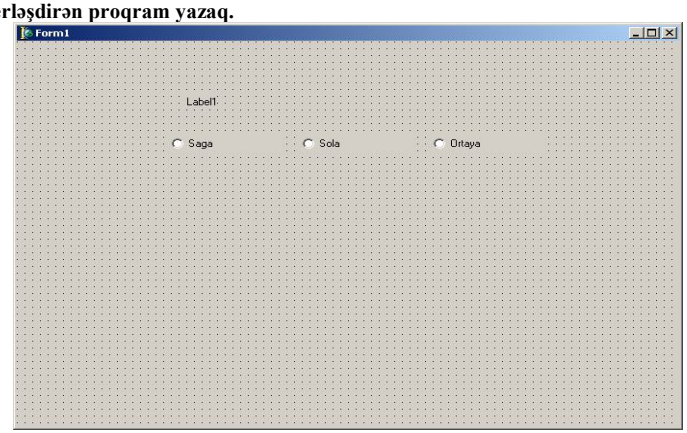
AutoSize **xassəsinə** true **qiyməti verilərsə,Label-in içindəki yazı dəyişdikdə onun ölçüsü də ona uyğun dəyişir.**

Alignment xassəsindən yazını sağa,sola və ortaya yerləşdirmək üçün istifadə olunur.

TaRightJustify : sağa
TaLeftJustify : sola
TaCenter : ortaya

Bu xassənin işləməsi üçün AutoSize **xassəsinə false qiymət mənimsədilməlidir.**

Misal olaraq Label içindəki yazını RadioButton-lar vasitəsi ilə sağa,sola və ortaya yerləşdirən proqram yazaq.




```
procedure TForm1.FormCreate(Sender: TObject);
begin
Label1.AutoSize:=False;
end;

procedure TForm1.RadioButton1Click(Sender: TObject);
begin
  Label1.Alignment:=taRightJustify;
```

```
end;

procedure TForm1.RadioButton2Click(Sender: TObject);
begin
Label1.Alignment:=taLeftJustify;
end;

procedure TForm1.RadioButton3Click(Sender: TObject);
begin
Label1.Alignment:=taCenter;
end;
```

Layout xassəsinin köməyi ile label de yazılan yazinin şaqülü yeri müəyyən edilir. Aşdaki qitmətlərdən birini ala biler.

tlTop üstdə
tlButtom altda
tlCenter ortada

WordWrap xassəsinin qiyməti True gotürülərsə label komponentinə daxil edilən ifadənin qiyməti onun ölçüsünü aşarsa bir alt sətrə keçirilər. Autosize xassəsi ilə birlikdə true qiyməti verilərsə həm hündürlüyə, həm də enə görə avtomatik olaraq nizamlanar.

6.5.3. Mətn sətri komponenti (TEdit)

Bu komponent mətni ekranda göstərməklə bərabər, onu redaktə etməyə imkan verir.

Xassələri

Redaktə olunan sətir **Text** xassəsində saxlanır.

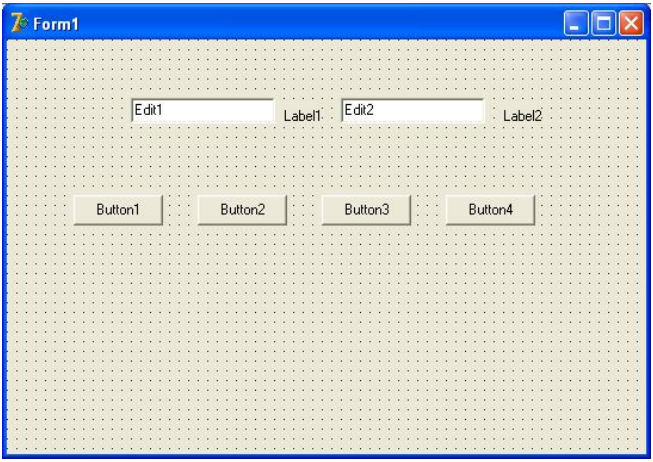

```
Edit1.text:='Sebuhi';
```

Istifadəçinin Edit mətn sətri komponentinə yazdığı qiyməti öyrənmək üçün aşağıdakı mənimsəmə operatorundan istifadə etmək olar.

```
Ad:=Edit1.Text;
```

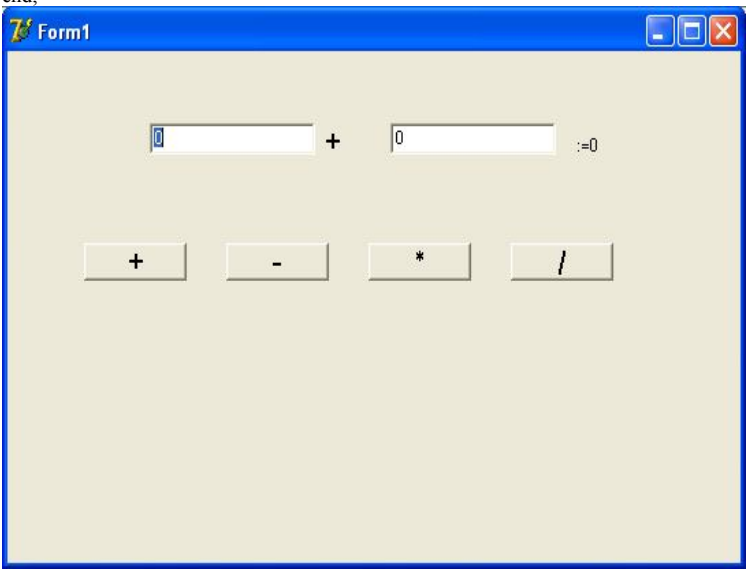
Misal.Istifadəçinin iki ədəd Edit komponentindən və 4 ədəd düymədən istifadə edərək hesab əməllərin yerinə yetirməsi proqramını işləyək.

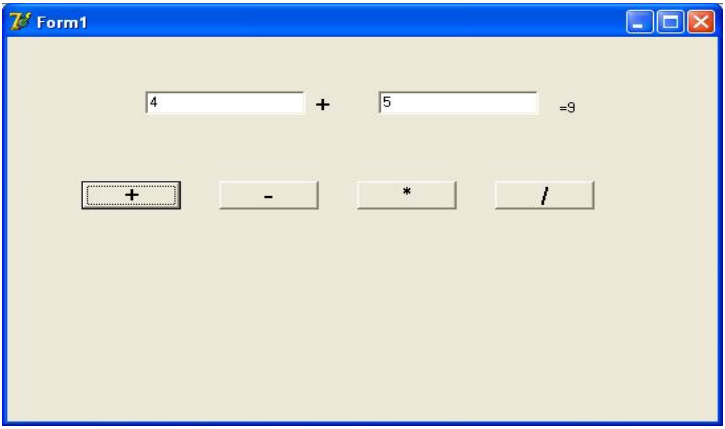
Misalımız üçün aşağıdakı formanı hazırlayaq.




```
procedure TForm1.Button1Click(Sender: TObject);
var
  s,s1,s2:real;
begin
s1:=StrToFloat(Edit1.Text);//onluq edede cevir
s2:=StrToFloat(Edit2.Text);//onluq edede cevir
s:=s1+s2;//Topla
Label1.Caption:='+';
Label2.Caption:='='+FloatToStr(s);
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  s,s1,s2:real;
begin
s1:=StrToFloat(Edit1.Text);//onluq edede cevir
s2:=StrToFloat(Edit2.Text);//onluq edede cevir
s:=s1-s2;//Cix
Label1.Caption:='-';
Label2.Caption:='='+FloatToStr(s);
end;
procedure TForm1.Button3Click(Sender: TObject);
var
  s,s1,s2:real;
begin
s1:=StrToFloat(Edit1.Text);//onluq edede cevir
s2:=StrToFloat(Edit2.Text);//onluq edede cevir
s:=s1*s2;//Vur
Label1.Caption:='*';
Label2.Caption:='='+FloatToStr(s);
end;
procedure TForm1.Button4Click(Sender: TObject);
```

```
var
  s,s1,s2:real;
begin
s1:=StrToFloat(Edit1.Text);//onluq edede cevir
s2:=StrToFloat(Edit2.Text);//onluq edede cevir
s:=s1/s2;//Bol
Label1.Caption:='/';
Label2.Caption:='='+FloatToStr(s);
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
Edit1.Text:='0';
Edit2.Text:='0';
Label1.Font.Size:=14;
Label1.Caption:='+';
Label2.Caption:=':=0';
Button1.Font.Size:=14;
Button1.Caption:='+';
Button2.Font.Size:=14;
Button2.Caption:='-';
Button3.Font.Size:=14;
Button3.Caption:='*';
Button4.Font.Size:=14;
Button4.Caption:='/';
end;
```

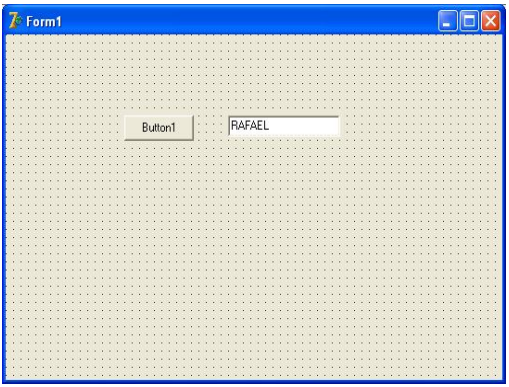




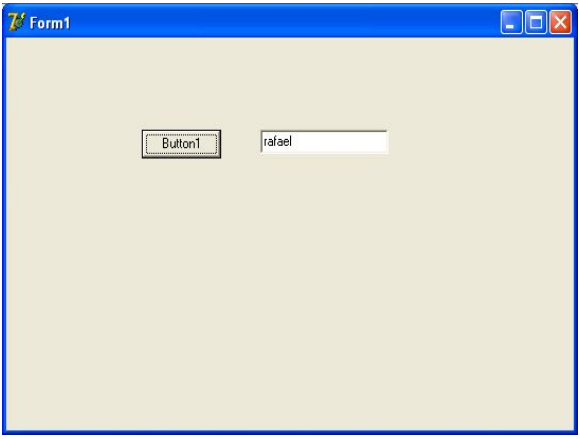
CharCase xassəsi

Bu xassənin köməyi ilə istifadəçinin Edit mətn sətri komponentinə daxil edəcəyi hərflərin hamısının böyük,kiçik və normal(həm böyük ,həm də kiçik) olmasını təmin etmək olar.Bu xassə həm layihələndirmə həm də proqram işləyərkən dəyişdirilə bilər.

EcUpperCase hərflər böyük hərfə çevirir
EcNormal normal giriş
EcLowerCase hərflər kiçik hərfə çevirir

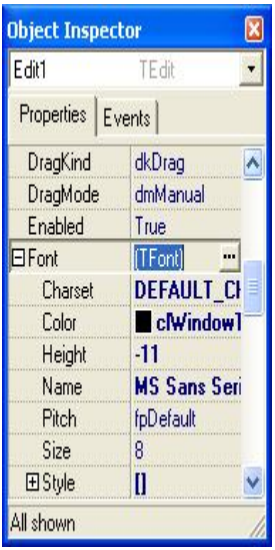



```
procedure TForm1.Button1Click(Sender: TObject);
begin
Edit1.CharCase:=ecLowerCase;
end;
```

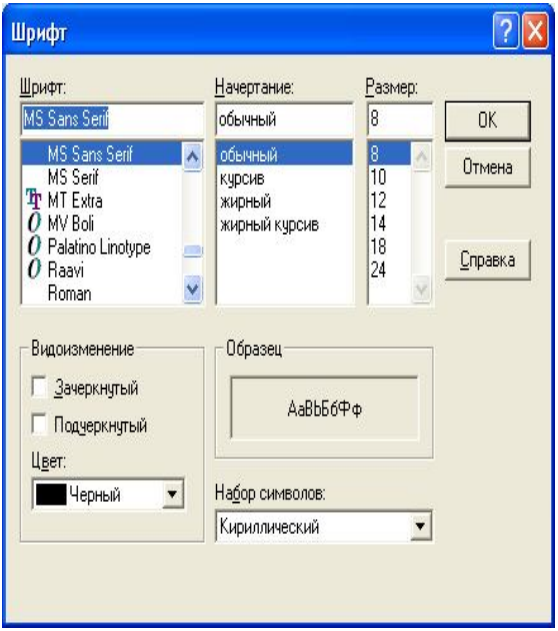


Font xassəsi

Font xassəsi komponent elementinin şrift ölçüsü və böyüklüyünü təyin edir.Açılan şrift pəncərəsi ilə bütün xassələr görülərək dəyişdirilə bildiyi kimi aşağıdakı xassələrlə tək-tək də dəyişdirilə bilər.Bu xassəninönündə + işarəsinin olması digər xassələrdən fərqli olaraq onun alt xassələrinin olduğunu göstərir.



Layihələndirmə zamanı Font pəncərəsindən istifadə etmək üçün Font xassəsinin qarşısındakı ... işarəsini vurmaq lazımdır.



Tək-tək dəyişdirmək üçün isə Font xassəsinin aşağıdakı alt xassələrindən istifadə etmək lazımdır.

Font.Color yazının rəngini təyin edir

Edit1.Font.Color:=clRed;//yazı rəngi qırmızıtəyin edilir

Font.Height Şriftin piksel olaraq ölçüsü

Font.Size Şriftin points olaraq ölçüsü

Size və Height xassələri eyni işi görərlər.Onların fərqi ölçü vahidlərinin fərqli olmasıdır.

Font.Name şriftin adı.Sistemə yüklənmiş şriftlərdən hər hansı birinin adının verilməsi lazımdır.

Edit1.Font.Name:=’Courier New’;

Font.Style Şriftin qalın,maili,altıcizgili və üstücizgili olması bu xassə ilə təyin olunur.

FsBold,fsItalic,fsUnderline,fsStrikeOut

Edit mətn sətri komponentinin şriftin qalın etmək üçün:

Edit1.Font.Style:=[fsBold];

Edit mətn sətri komponentinin şriftin qalın və maili etmək üçün:

Edit1.Font.Style:=[fsBold,fsItalic];

Digər xassələri dəyişdirmədən şrifti qalın etmək üçün:

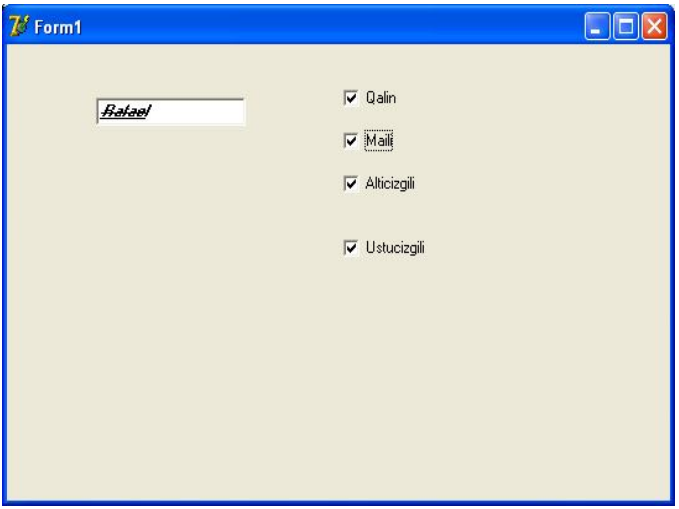
Edit1.Font.Style:= Edit1.Font.Style +[fsBold];

Şriftin qalın xassəsini aradan qaldırmaq üçün:

Edit1.Font.Style:= Edit1.Font.Style -[fsBold];

Misal.Edit1 mətn sətri komponenti içərisindəki yazını bayraqcıq(CheckBox) komponentinin köməyi ilə qalın,maili , altıcizgili və üstü xətli edəcək proqram tərtib edək.

```
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
if Checkbox1.Checked then
  Edit1.Font.Style:=Edit1.Font.Style+[fsBold]
   else
     Edit1.Font.Style:=Edit1.Font.Style-[fsBold]
end;

procedure TForm1.CheckBox2Click(Sender: TObject);
begin
if Checkbox2.Checked then
  Edit1.Font.Style:=Edit1.Font.Style+[fsItalic]
   else
     Edit1.Font.Style:=Edit1.Font.Style-[fsItalic]
end;

procedure TForm1.CheckBox3Click(Sender: TObject);
begin
if Checkbox3.Checked then
  Edit1.Font.Style:=Edit1.Font.Style+[fsUnderline]
   else
     Edit1.Font.Style:=Edit1.Font.Style-[fsUnderline]
end;

procedure TForm1.CheckBox4Click(Sender: TObject);
begin
if Checkbox4.Checked then
  Edit1.Font.Style:=Edit1.Font.Style+[fsStrikeOut]
   else
     Edit1.Font.Style:=Edit1.Font.Style-[fsStrikeOut]
end;
```



Font.Pitch bu xassə şriftdəki hər bir işarənin bərabər genişlikdə olub-olmadığını yoxlayır.Alabiəcəyi qiümətlər aşağıdakılardır.

FsFixed hər bir hərf eyni genişlikdədir

FsVariable hər bir hərf öz genişliyindədir

FsDefault şrift nə cür vardırsa,eləcə də istifadə olunacaq.Yəni şrift tipi Courier New olarsa,hər bir hərf eyni genişlikdə, Times New Roman olarsa,hər bir hərf öz genişliyində olacaqdır. Fixed və Variable şriftləri arasındakı fərqi aşağıdakı iki sətirdə görə bilərsiniz.olan

AHMEDOVSADIQIIIXXX

AHMEDOVSADIQIIIXXX

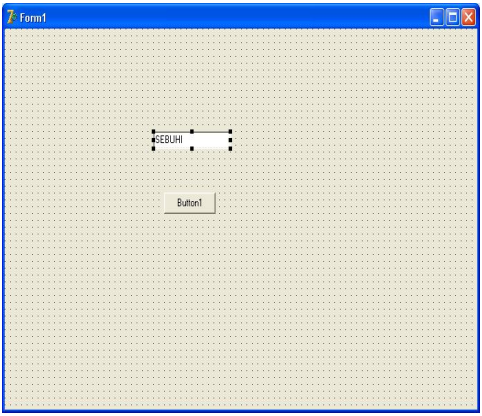
Birinci sətir Fixed Pitch şrifti olan Courier New ilə yazıldı.Hər bir hərf eyni genişlikdədir.(I hərfi ilə X hərfi eyni sahəni tutar.)Ikinci sətir isə Variable Length olan Times New Roman ilə yazıldı.(I hərfi ilə X hərfi müxtəlif sahəni tutar.)

PasswordChar xassəsi

PasswordChar xassəsi Edit qutusu şifrə üçün nəzərdə tutulduqda istifadə olunur.Bir çox hallarda girilən şifrə üçün * işarəsi nəzərdə tutulur. PasswordChar xassəsinə şifrə olaraq bir işarə daxil etdikdə işarənin yerinə * işarəsi görünür.

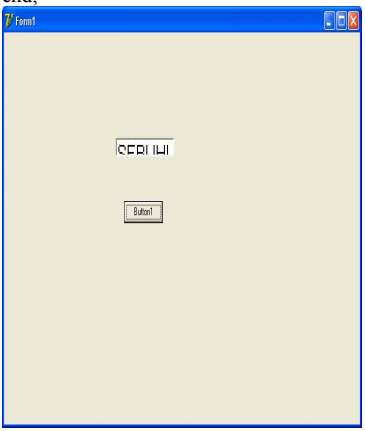
Edit1. PasswordChar:='*';

AutoSize xassəsi true olarsa, Edit1-in şrift ölçüsünü artırdıqda onun özü də eyni ölçüdə dəyişəcəkdir.Əks halda Edit1-in ölçüsü sabit qalacaq və yazının bir hissəsi görünməyəcəkdir.Alt-alta bir neçə komponentimiz vardırsa, bunlardan birinin AutoSize xassəsi true olarsa ,yazının ölçüsü böyüdükdə komponentin ölçüsü də böyüdüyndən digər komponentin üzərini örtəcəkdir.



Yuxarıdakı formadan istifadə edərək aşağıdakı proqramm kodlarını Button düyməsininClick hadisəsinin emaledicilərinə yazaq.

```
begin
Edit1.AutoSize:=false;
Edit1.Font.Size:=20;
end;
```




```
begin
Edit1.AutoSize:=true;
Edit1.Font.Size:=20;
end;
```

Anchors xassəsi

Bu xassə komponentin üzərində yerləşdiyi komponent üzərindəki vəziyyətin təyin edir.Aşağıdakı xassələri təyin etməklə əsas komponent hansı ölçüdə böyüyürsə içərisində olan komponentləri də eyni ölçüdə böyütmək olar.

akTop komponentin yuxarı nöqtəsi əsas komponentə görə sabit qalır.

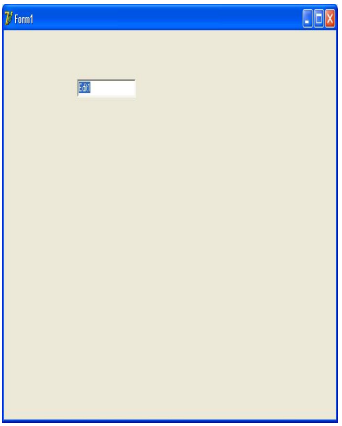
akLeft komponentin sol nöqtəsi əsas komponentə görə sabit qalır.

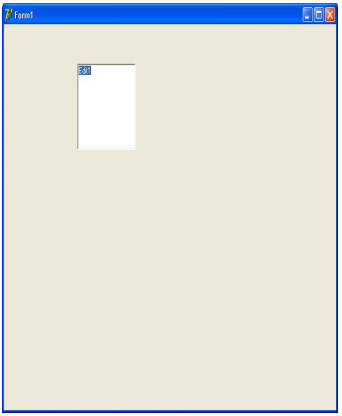
akRight komponentin sağ nöqtəsi əsas komponentə görə sabit qalır.

akBottom komponentin aşağı nöqtəsi əsas komponentə görə sabit qalır.

Edit1 komponentin formaya bağlayaq.Belə ki,forma hansı ölçüdə böyüyəcəksə, Edit1-də həmin ölçüdə böyüyəcəkdir.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
Edit1.Anchors:=[akLeft,akRight,akTop,akBottom];
end;
```





Əgər biz Edit1.Anchors:=[akLeft,akRight]; götürsək, Edit qutusu formanın genişliyinə görə dəyişəcək,hündürlüyü isə sabit qalacaqdır.

Constraints xassəsi

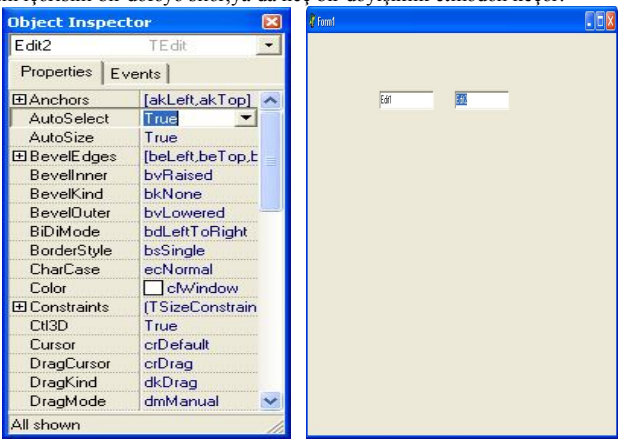
Bu xassənin köməyi ilə komponentin böyüdülməsinə məhdudiyyətlər qoyulur.Komponentin Constraints.MaxHeight, Constraints.MaxWidth, Constraints.MinWidth, Constraints.MinHeight qiymətləri verilərək minimal və maksimal dəyişmə ölçüləri təyin olunur.

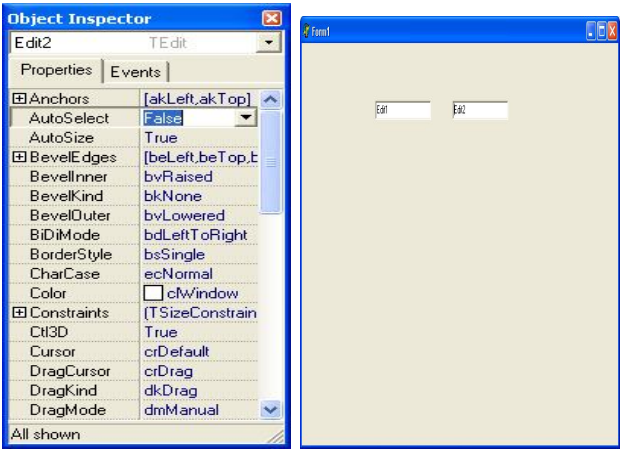
```
procedure TForm1.FormCreate(Sender: TObject);
begin
Edit1.Anchors:=[akLeft,akRight,akTop,akBottom];
with Edit1.Constraints do
 begin
  MaxHeight:=100;
  MaxWidth:=100;
  MinWidth:=50;
  MinHeight:=50;
 end;
end;
```

Proqramı işlətdikdən sonra formanı böyütdükdə Edit-in müəyyən həddən sonra böyümədiyini görəcəyik.

AutoSelect xassəsi

AutoSelect xassəsinin True olması Tab düyməsinin köməyi ilə bu komponentə keçdikdə onun içərisinin seçilməsini təmin edir.Istifadəçi istəyindən asılı olaraq bu komponentin içərisini bir dəfəyə silər,ya da heç bir dəyişiklik etmədən keçər.

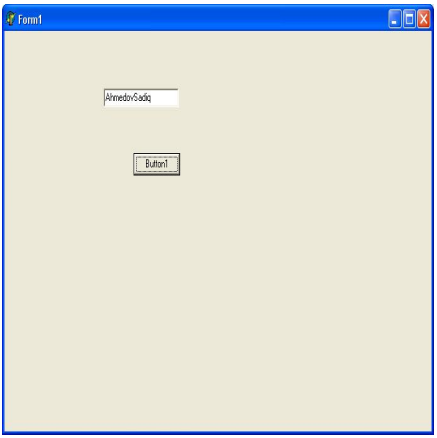




MaxLength xassəsi

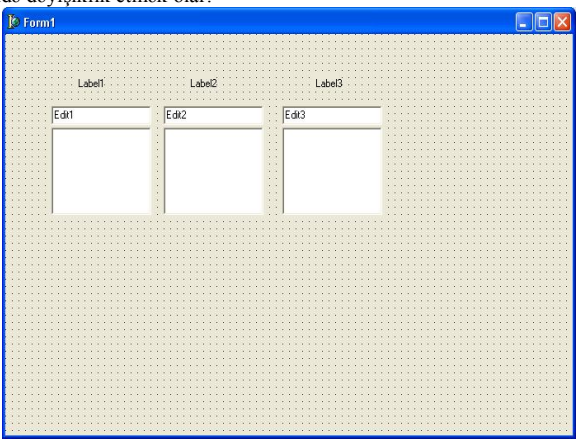
MaxLength xassəsi qutuya daxil ediləcək işarələrin sayını təyin edir.Istifadəçinin məlum saydan artıq işarə daxil etməsinin qarşısını almaq üçün istifadə olunur.Istifadəçinin 3 hərfdən artıq işarə daxil etməsinin qarşısını almaq üçün bu xassəyə 3 qiymətini vermək lazımdır. MaxLength xassəsi ilə təyin olunan sərhədi proqram kodu vasitəsi ilə aşmaq olar.Aşağıdakı misaldan göründüyü kimi MaxLength xassəsi ilə 4 qiyməti verildiyi halda proqram kodu ilə bu sərhəd keçilir.

```
begin
Edit1.MaxLength:=4;
Edit1.Text:='AhmedovSadiq';
end;
```



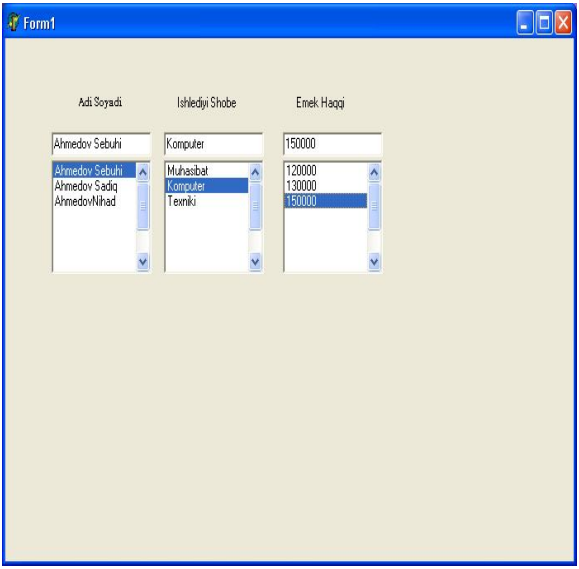
ReadOnly xassəsi

Mətn sətrində dəyişiklik və qiymət daxil etməyi qadağan etmək üçün, ReadOnly xassəsində false qiymətinin əvəzində true qiymətini seçmək lazımdır. Proqram kodu vasitəsi ilə mətn sətrində dəyişiklik etmək olar.




```
procedure TForm1.FormCreate(Sender: TObject);
var
i:integer;
sifre:string;
begin
sifre:=InputBox('Sifre Girishi','Daxil Edin','');
if (sifre<>'asr') then
  begin
   ShowMessage('Sifreni Bilmediniz.Deyishiklik ede bilmezsiniz');
   Edit1.ReadOnly:=True;
   Edit2.ReadOnly:=True;
   Edit3.ReadOnly:=True;
   end;
   if FileExists('c:\isadsoy.text') then
     begin
      ListBox1.Items.LoadFromFile('c:\students\isadsoy.text');
      ListBox1.Items.LoadFromFile('c:\students\issob.text');
      ListBox1.Items.LoadFromFile('c:\students\ishaqqi.text');
     end
     else
     begin
     for i:=1 to 10 do
     begin
     listBox1.Items.Add('');
     listBox2.Items.Add('');
     listBox3.Items.Add('');
     End;
    end;
     ListBox1.ItemIndex:=0;
```

```
    ListBox2.ItemIndex:=0;
    ListBox3.ItemIndex:=0;
    end;
procedure TForm1.Edit1Change(Sender: TObject);
begin
ListBox1.Items[ListBox1.ItemIndex]:=Edit1.Text;
end;

procedure TForm1.Edit2Change(Sender: TObject);
begin
ListBox2.Items[ListBox2.ItemIndex]:=Edit2.Text;
end;

procedure TForm1.Edit3Change(Sender: TObject);
begin
ListBox3.Items[ListBox3.ItemIndex]:=Edit3.Text;
end;

procedure TForm1.ListBox1Click(Sender: TObject);
begin
Edit1.Text:=ListBox1.Items[ListBox1.ItemIndex];
end;

procedure TForm1.ListBox2Click(Sender: TObject);
begin
 Edit2.Text:=ListBox2.Items[ListBox2.ItemIndex];
end;

procedure TForm1.ListBox3Click(Sender: TObject);
begin
Edit3.Text:=ListBox3.Items[ListBox3.ItemIndex];
end;

procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
 ListBox1.Items.SaveToFile('c:\students\isadsoy.text');
 ListBox2.Items.SaveToFile('c:\students\issob.text');
 ListBox3.Items.SaveToFile('c:\students\ishaqqi.text');
end;
```

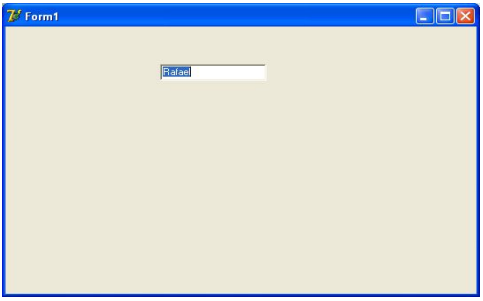


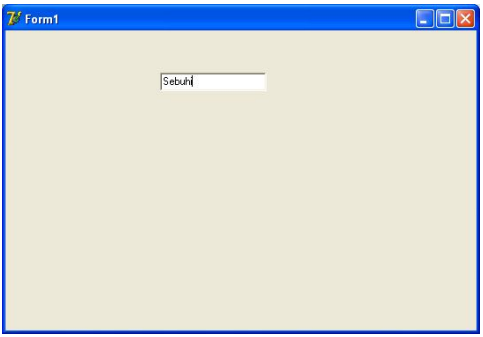
Modified xassəsi

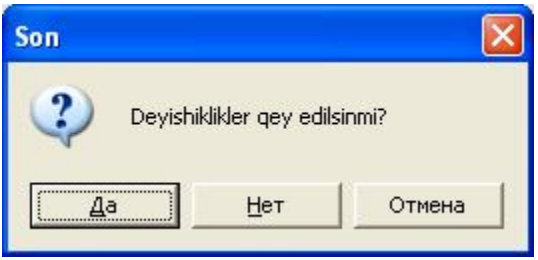
True və ya False qiymətlərini ala bilən bu xassə Text xassəsində hər hansı dəyişiklik edildikdə True qiymətini alır.

Misal. Yalnız dəyişiklik olunduğu zaman proqramdan çıxarkən'Deyishiklikler qeyd edilsinmi?' kimi mesaj pəncərəsi ekrana çıxaran proqram yazaq.

```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
var
  c:integer;
begin
if edit1.modified then
  begin
    c:=application.MessageBox('Deyishiklikler qeyd
edilsinmi?','Son',Mb_YesNoCancel+Mb_IconQuestion);
     case c of
     IdYes:;//Qeyd etmek ucun lazimli kod
     IdCancel:Action:=caNone;
     end;
     end;
end;
```







Left,Top,Width,Height xassələri

Bu xassələr proqram işlədikdə ekranda görünən bütün komponentlərdə vardır.Komponentin sol və üst komponentlərini,genişliyini ,yüksəkliyini təyin edir.

```
procedure TForm1.Button1Click(Sender: TObject);
var
i,w,h:integer;
begin
w:=form1.Width-edit1.Width;
h:=form1.Height-edit1.Height;
//formanin yuxari sol kuncune al
edit1.Left:=1;
edit1.Top:=1;
//saga hereket etdir
```

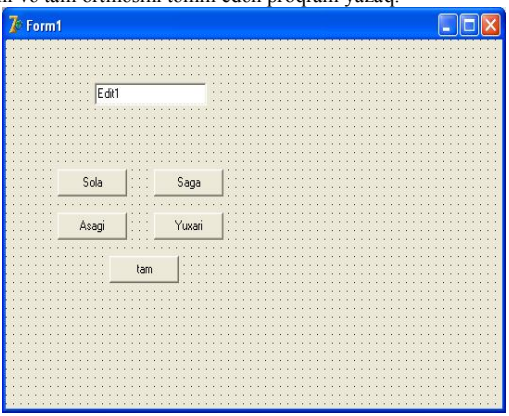
```
for i:=1 to w do
edit1.Left:=i;
//asagi hereket etdir
for i:=1 to h do
edit1.Top:=i;
//formanin yuxari sol kuncune al
edit1.Left:=1;
edit1.Top:=1;
//genisliyi artir
Caption:='Width artirilir';
for i:=1 to form1.Width do
edit1.Width:=i;
//hundurluyu artir
 Caption:='Height artirilir';
for i:=1 to form1.Height do
edit1.Height:=i;
 Caption:='Form1';

end;
```

Align xassəsi

Align xassəsi komponenti formanın istənilən hissəsinə(soluna,sağına,aşağı,yuxarı,formanı tam örtmək)yerləşməsinə xidmət edir.Formanın ölçüləri dəyişdikdə Align xassəsi verilən komponentdə onun ölçüləri də dəyişəcəkdir. Align xassəsi aşağıdakı qiymətlərdən birini ala bilər.

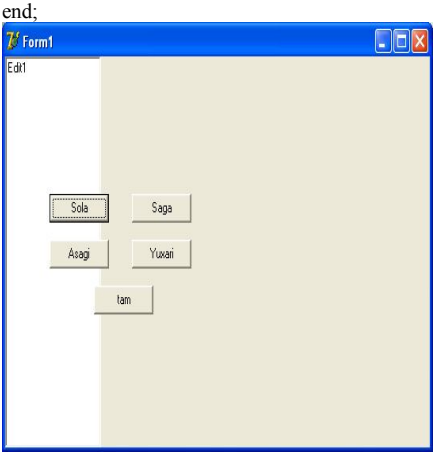
Align=(alNone,alTop,alBottom,alLeft,alRight,alClient)

Misal.Edit komponentinin formanın solunda,sağında,aşağısında , yuxarısında yerləşməsini və tam örtməsini təmin edən proqram yazaq.




```
procedure TForm1.Button1Click(Sender: TObject);
begin
edit1.Align:=alLeft;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
```

```
edit1.Align:=alRight;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
edit1.Align:=alBottom;
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
edit1.Align:=alTop;
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
edit1.Align:=alClient;
end;
```



Visible xassəsi

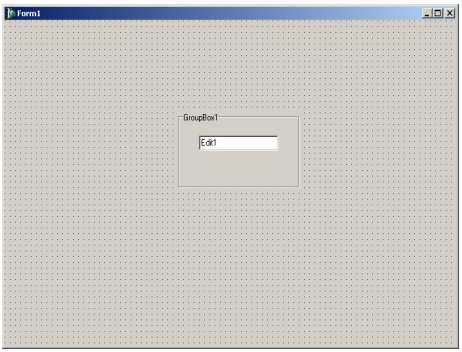
Bu xassənin köməyi ilə komponentin ekranda görünüb görünməməsi təmin edilir.Xassəyə false qiyməti mənimsədərək komponentin ekranda görünməsinin qarşısını almaq olar.

Edit1.Visible:=False;

Xassəyə true qiyməti verərək yenidən görünməsini təmin etmək olar.Sınaq imtahanları zamanı cavabın verildiyi komponenti gizləyərək sınaq bitdikdən sonra görüntüyə gəlməsini təmin etmək olar.

Showing xassəsi

Visible xassəsinə true qiyməti mənimsədilməsi komponentin həmişə görüməsini təmin etmir. Komponent əgər başqa bir komponentin (məsələn Panel kimi)üzərində yerləşmişdirsə və üzərində yerləşdirilən komponentin Visible xassəsinə false qiyməti mənimsədilmişdirsə,bu komponent görünməyəcəkdir. Komponentin forma üzərində görünüb-görünməməsi Showing xassəsi ilə təyin edilir. Bu xassə dəyişdirilə bilməz. Üzərində yerləşdiyi komponentin Visible xassəsinin dəyişməsi ilə Showing xassəsi öz-özünə dəyişir.

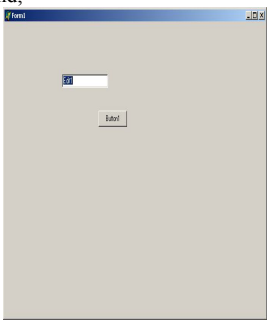


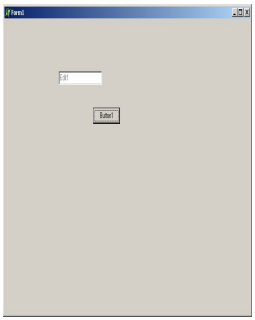
GroupBox1.Visible:=False olarsa,Edit1.Visible:=True olsa belə Edit1 komponenti ekranda görünməyəcəkdir.Çünki Edit1-in Visible xassəsinə True qiyməti verilsə də üzərində yerləşdiyi GroupBox1 komponentinin görünməzliyi təmin olunmuşdur.

Enabled xassəsi

Bu xassə komponentin aktiv və ya passiv olmasını təyin edir. Enabled xassəsinin True olması bu komponent üzərində işləməyin mümkünlüyünü ,False olması işləməyin mümkün olmadığını göstərir. Visible xassəsindən fərqli olaraq istifadəçi komponenti ekranda görür, lakin onun üzərində heç bir iş görə bilmir.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
 if Edit1.Enabled=true then
        edit1.Enabled:=false
 else
        edit1.Enabled:=true;
end;
```





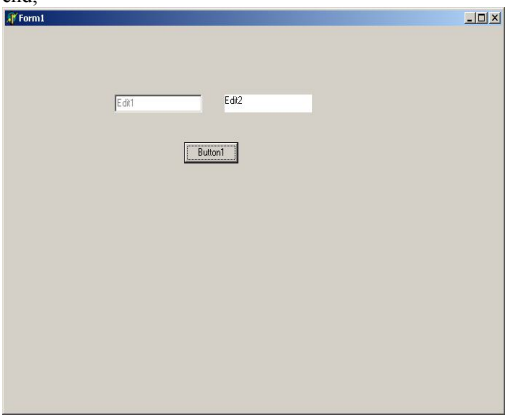
Eyni işi aşağıdakı bir sətirlik proqram kodu vasitəsi ilə də yerinə yetirmək olar.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit1.Enabled:=not edit1.Enabled;
end;
```

BorderStyle xassəsi

Bu xassə komponentin ətrafında çərçivə olub olmamasını təyin edir.bsSiingle qiymətinin verilməsi çərçivənin olmasını ,bsNone qiymətinin verilməsi isə olmamasını təmin edir.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit1.Enabled:=not edit1.Enabled;
end;
```



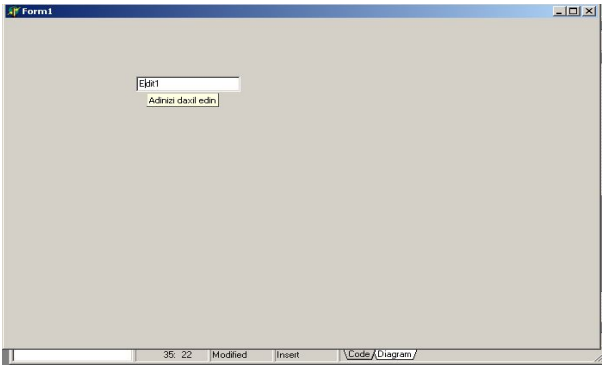
Hint,ShowHint xassələri

Bir çox proqramlarda obyektin üzərinə getdiyimiz zaman müəyyən aydınlaşdırıcı mətnin ekrana gəldiyinin şahidi olmuşuq. Delphi- də mausun göstəricisini komponentin üzərinə gətirdikdə belə mətn görünür.

Əgər istəsək biz özümüz də mausu komponentin üzərinə gətirdikdə müəyyən aydınlaşdırıcı mətnin verilməsini və onun görüntüyə gəlməsini təmin edə bilərik. Aydınlaşdırıcı mətnin məzmununu Hint xassəsi ilə onun göstərilib-göstərilməməsini isə ShowHint xassəsinə true və ya false qiymətlərini verərək yerinə yetirmək olar .

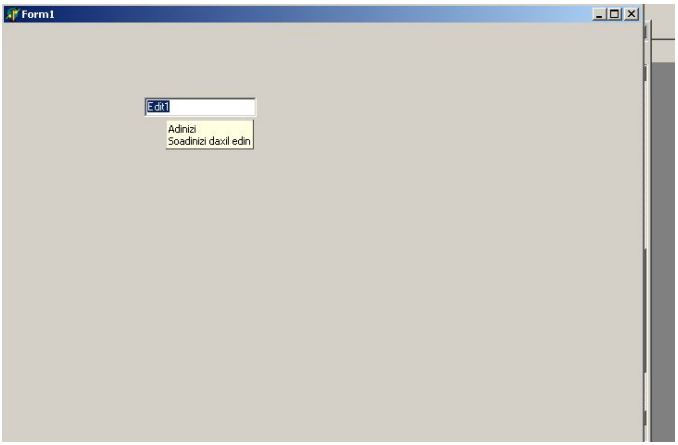
Misal. Forma yaranarkən mətn sətri komponentinin Hint xassəsi təyin olunur. Mausun göstəricisini Edit1 komponentinin üzərinə gətirdikdə Hint xassəsinə yazdığımız aydınlaşdırıcı mətn ekrana gələcəkdir.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
Edit1.Hint:='Adinizi daxil edin';
Edit1.ShowHint:=true;
end;
```



Çox sətirli izahedici mətnlər də vermək mümkündür.Sətirlər arasında Enter düyməsinin koduna uyğun gələn #13 işarəsini yazmaq lazımdır.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
Edit1.ShowHint:=true;
Edit1.Hint:='Adinizi'+#13+'Soadinizi daxil edin';
end;
```



Name xassəsi

Bu xassə ilə komponentin adı təyin edilir.Əgər istifadəçi tərəfindən Name xassəsi dəyişdirilməmişdirsə, forma Delphi üzərinə yerləşdirilmiş hər bir elementə ad verir:Edit1,Edit2,Button1,Button2,Label1.

Bu xassəni layihələndirmə zamanı Object Inspector pəncərəsindən dəyişdirmək olar Bu xassə ilə təyin olunan adla komponentin metod və xassələrinə müraciət edə bilərsiniz

TabStop xassəsi

TabStop xassəsi True olan hər bir komponentə Tab düyməsi ilə keçid mümkündür. Istifadəçi verilənləri daxil edərkən istifadə etmədiyi komponentlərin bu xassəsini

False verməklə xeyli vaxta qənaət edə bilər.Aşağıdakı kimi bir forma üzərində istifadəçi verilənləri daxil edərkən Sil ,Qeydet,Yukle,Cix düymələrinin TabStop xassəsini False verməklə işini xeyli sürətlənidrir.

TabOrder xassəsi

Bu xassənin köməyi ilə forma üzərində yerləşən komponentlərə Tab düyməsi ilə keçid ardıcıllığı müəyyən edilir.Delphi forma üzərinə ilk qoyulan elementin TabOrder xassəsinə 0 ,ikincisinə 1, üçüncüsünə 2 və s. mənimsədir.Bu sıra TabOrder xassəsi ilə də dəyişdirilə bilər.Forma üzərində yerləşən komponentlərin TabOrder xassəsini tək-tək deyil, bir neçəsini eyni vaxtda dəyişdirmək olar.Bunun üçün Shift düyməsini sıxılı vəziyyət də saxlayaraq istədiyimiz sıralama ilə komponentləri seçdikdən sonra Ctrl+X düymələrin sıxaraq onları mübadilə buferinə yazırıq. Ctrl+V düymələrini sıxaraq təkrar geriyə yapışdırırıq.Bundan sora bütün komponentlər əvvəlki qaydada TabOrder xassəsi bizə lazım olan qiymətlərlə forma üzərində yerləşəcəkdir. TabOrder xassəsini dəyişməyin digər yolu əmrlərinin köməyi ilə aşağılakı pəncərədən istifadə etməkdir.Dialoq pəncərəsindəki yuxarı və aşağı düymələrindən istifadə edərək TabOrder xassəsini asanlıqla nizamlamaq olar.

SelStart,SelLength,SelText

Windows-da seçmə ,surətini çıxarma ,kəsmə və yapışdırma əməliyyatları çox istifadə olunan xassələrdir. Istifadəçinin qiymət daxil edə bildiyi komponentlər bu xassələri standart düymələrlə heç bir proqram koduna ehtiyac qalmadan dəstəkləyir. Proqramla da bu işləri görmək mümkündür.Bu işləri görərkən seçili olan hissəni müəyyən etmək üçün SelStart və SelLength xassələrindən istifadə olunur. Seçimdəki ilk işarənin yerini verir.Ilk işarə üçün bu qiymət 0-dır. SelText xassəsi isə seçilən hissəni öyrənməyə və dəyişdirməyə imkan verir.

```
Edit1.SelStart:=0;// secme ilk herfdey baslamis
Edit1.SelLength:=6;//6 herf secimish
Edit1.SelText:='Sebuhi';
```

Aşağıda verilən misalla seçili mətn, seçili işarələrin sayı,bütün mətndə işarələrin sayı verilmişdir.

```
Procedure Tform1.Button1Click(Sender:Tobject);
Begin
  Label4.Caption:=Edit1.SelText;
  Label5.Caption:=IntTOStr(Edit1.SelLength);
  Label6.Caption:=IntToStr(Length(Edit1.Text));
End;
```

Misal Edit mətn sətri komponentində seçilmiş mətni böyük-kiçik hərfə çevirən proqram yazaq.

```
Procedure Tform1.Button1Click(Sender:Object);
Begin
 If Edit1.SetLength=0 then
   ShowMessage('Cevrilecek hisseni sec')
 Else
   Edit1.SelText:=AnsiUpperCase(Edit1.SelText);
End;

Procedure Tform1.Button2Click(Sender:Object);
Begin
 If Edit1.SetLength=0 then
   ShowMessage('Cevrilecek hisseni sec')
 Else
   Edit1.SelText:=AnsiLowerCase(Edit1.SelText);
```

End;

ComponentIndex xassəsi

Proqram işləyərkən dəyişdirilə bilən bu xassə forma üzərində yerləşən elementlərin sıra nömrəsini verir.Forma üzərində yerləşdirilən ilk komponentin sıra nömrəsi 0 ,ikinci elementin sıra nömrəsi 1 ,büün elementlərə sıra nömrəsi mənimsədilir.

Cursor xassəsi

Bu xassə maus komponentin üzərindən onun göstəricisinin keçərkən aldığı şəkli təyin edir.

Indi mausun göstəricisinin ala biləcəyi bütün şəkilləri göstərən proqram yazaq.Bunun üçün forma üzərinə bir ədəd komponenti yerləşdirək. Object Inspector pəncərəsindən ListBox1-in Items xassəsinin üzərində mausun düyməsini vuraraq açılan pəncərəyə mausun kursorunun ala biləcəyi bütün kursor tiplərini daxil edək .

Sonra hadisəsinin emaledicisinə aşağılakı proqram kodlarını yazırıq.

```
Procedure Tform1.ListBox1Click(Sender:Tobject);
 Var
   Maussekli:Hcursor;
 Begin
Faresekli:=-ListBox1.ItemIndex;
Form1.Cursor:=faresekli;
Form1.Caption:=ListBox1.Items[ListBox1.ItemIndex];
End;
```

Proqramı işlətdikdən sonra siyahıdakı istənilən elementin seçilməsi ilə mausun göstəricisi uyğun şəkli alacaqdır.

Owner xassəsi

Proqram işləyərkən istifadə olunan bu metod hansı elementin hansı elemeniə aid olduğunu göstərir.Forma üzərində yerləşən bütün elementlər forma aid olduğu kimi forma özü də TApplication obyektinə aiddir.Bir Owner yaddaşdan atılarkən ona aid olan bütün komponentlər də atılır.

Aşağıdakı forma üzərində yerləşən komponentlərin Owner -lərini öyrənmək üçün aşağıdakı kodu yazaq :

```
procedure Tform1.FormCreate(Sender:Tobject);
var s:string;
begin
 s:='Edit1> '+Edit1.Owner.ClassName+#13#10;
 s:=s+'Button1> '+Button1.Owner.ClassName+#13#10;
 s:=s+'GroupBox1> '+ GroupBox1.Owner.ClassName+#13#10;
 s:=s+'Edit2> '+Edit2.Owner.ClassName+#13#10;
 s:=s+'Form1> '+Form1.Owner.ClassName+#13#10;
 showmessage(s);
 end;
```

Göründüyü kimi forma üzərində yerləşən bütün komponentlərin Owner-i Tform ,formanın Owner–i isə TApplication –dur.

Parent xassəsi

Parent xassəsi komponentin üzərində yerləşdiyi komponentin adını verir.Bir komponent elementinin Parent –i üzərində yerinə yetirilən bütün işlər eynən həmin elementə də şamil edilir.Aşağıdakı forma üzərində yerləşən komponentlərin Parent–ini öyrənmək üçün

aşağılakı proqram kodunu yazaq

```
procedure Tform1.FormCreate(Sender:Tobject);
var s:string;
begin
 s:='Edit1> '+Edit1.Parent.ClassName+#13#10;
 s:=s+'Button1> '+Button1.Parent.ClassName+#13#10;
 s:=s+'GroupBox1> '+ GroupBox1.Parent.ClassName+#13#10;
 s:=s+'Edit2> '+Edit2.Parent.ClassName+#13#10;
 s:=s+'Form1> '+Form1.Parent.ClassName+#13#10;
 showmessage(s);
 end;
```

Proqram işlədikdən sonra aşağıdakı nəticəni alarıq.

Aşağıda verilən misalla Form1 üzərində yerləşən Button1 düyməsinin Parent –i dəyişdirilərək Form2–yə verilir.Bunun üçün Unit1–ə aşağıdakı proqram kolunu yazırıq.

```
 Uses unit2;
 {$R *.DFM}
 procedure Tform1.Button1Click(Sender:Tobject);
  begin
   button1.parent:=form2;
   form2.show;
  end;
```

Proqramı işlədib Button1 düyməsini sıxsaq, Button1 yeni sahibinə məxsus olacaqdır.

ParentColor xassəsi

Əgər ParentColor xassəsinə qiyməti mənimsədilmişdirsə , hər bir komponent üzərində yerləşdiyi Parent–in rəngini alır, əks halda öz rəngində qalır.

ParentShowHint xassəsi

ParentShowHint xassəsi true qiymətini alarsa,forma üzərindəki bütün komponentlər öz parent-lərinin ShowHint xassəsindən, əks halda isə öz ShowHint xassələrindən istifadə edirlər.Hər bir komponent üçün ShowHint xassəsini true vermək əvəzinə onların parent-lərinin ShowHint xassəsini true vermək kifayətdir.

ParentColor xassəsi

Əgər ParentColor xassəsinə qiyməti mənimsədilmişdirsə , hər bir komponent üzərində yerləşdiyi Parent–in rəngini alır, əks halda öz rəngində qalır.

ParentCtl3D xassəsi

ParentCtl3D xassəsi true isə komponent öz Parent–inin 3 ölçülü xassəsindən ,false isə komponent özünün Ctl3D xassəsindən istifalə edir.

Tag xassəsi

Bu xassə ilə LongInt tipində bir ədəd saxlanılır.Bu xassənin Delphi üçün heç bir mənası yoxdur.Istifadəçi bu xassədən qlobal bir dəyişən kimi istifadə edə bilər.

Misal Istifadəçi düyməsini sıxdıqda formanın başlığında bir ədədi 0-dan başlayaraq ardıcıl olaraq artırsın və formanın başlığına yazsın.Istifadəçi Dur düyməsini sıxana qədər bu iş davam etsin.

```
{$J+}
var
  basildi:integer;
procedure Tform1.Button1Click(Sender:Tobject);
 const
   i:integer=0;
begin
  while basildi=0 do
      begin
        i:=i+1;
       Form1.Caption:=IntToStr(i);
      End;
      Basildi:=0;
End.
procedure Tform1.Button2Click(Sender:Tobject);
begin
   basildi:=1;
end;
```

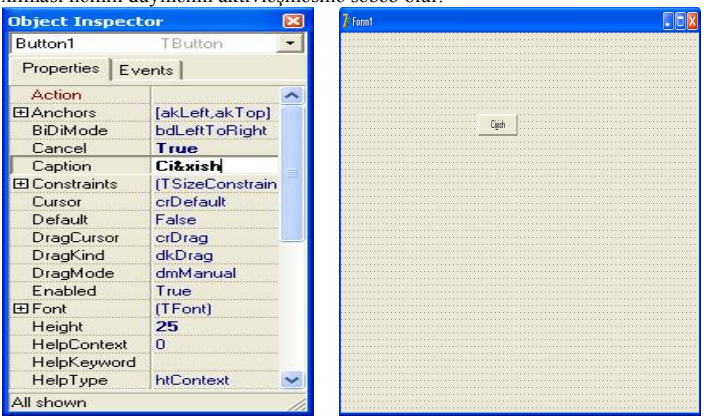
Basildi dəyişənini qlobal elan edərək düyməsi sıxılını qədər proqramın işləməsini təmin edə bilərik.
Eyni işi Basildi dəyişənini elan etmədən Button2–nin Tag xassəsindən qlobal dəyişən kimi istifadə edərək yerinə yetirmək olar.

```
{$J+}
  procedure Tform1.Button1Click(Sender:Tobject);
   const
     i:integer=0;
  begin
    while button2.tag=0 do
        begin
          i:=i+1;
         Form1.Caption:=IntToStr(i);
        End;
        Button2.tag:=0;
  End.
procedure Tform1.Button2Click(Sender:Tobject);
begin
   button2.tag:=1;
end;
```

Tətbiqi tədqiqatçıya real olaraq lazım olan **OnChange** hadisəsinin emalının mənası var. Onun emaledicisi **Text** xassəsində hər hansı dəyişiklik etdikdə çağrılır.

6.5.4. Düymə komponenti (TButton)

Təyinatı. Komponent Windows-un standart düyməsindən ibarətdir. Caption xassəsi düymə üzərində nişanı dəyişməyə imkan verir düymənin emaledicisinin işə düşməsinə səbəb olur. Caption xassəsində hansı hərfin önünə işarəsi qoyularsa, ALT və həmin hərfin eyni zamanda sıxılması həmin düymənin aktivləşməsinə səbəb olur.

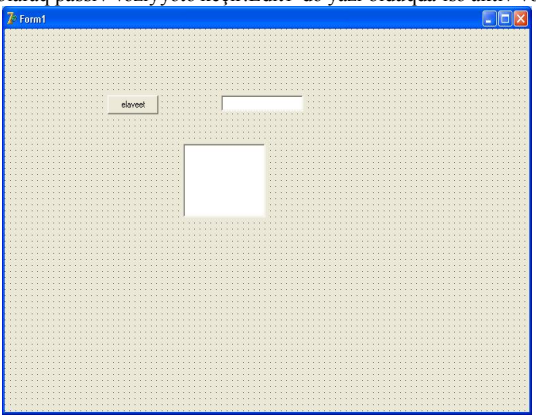


Bu işi proqram kodunun köməyi ilə də yerinə yetirmək olar.

Button1.Caption:=’Ci&xish’;

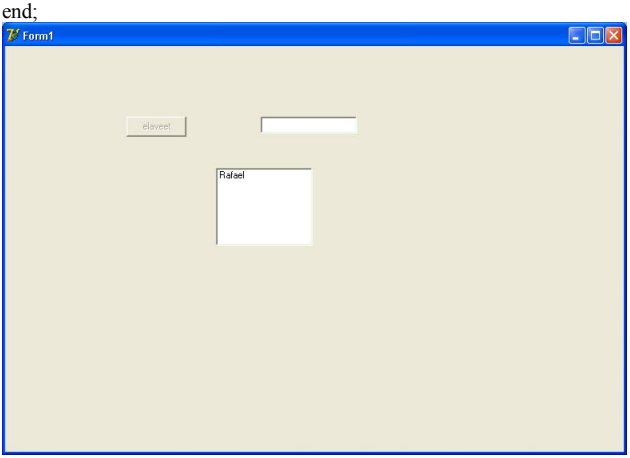
Cancel xassəsinə true qiymətinin mənimsədilməsi formanın hər hansı bir yerində ESC düyməsinin sıxılması bu düyməni aktiv vəziyyətə gətirir.

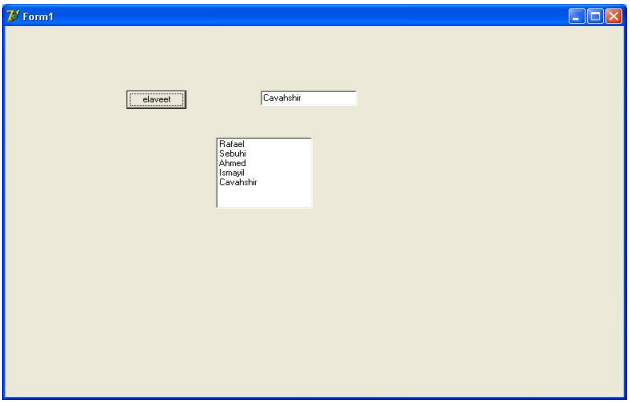
Enabled xassəsi true və ya false qiymətlərindən birini alaraq düymənin aktiv və ya passiv olmasını təyin edir.Aşağıda verilən misalda Edit1 içərisində yazı olmadıqda düymə avtomatik olaraq passiv vəziyyətə keçir.Edit1-də yazı olduqda isə aktiv vəziyyətə keçir.




```
procedure TForm1.Edit1Change(Sender: TObject);
begin
if length(edit1.Text)>0 then
  button1.Enabled:=true
else
 button1.Enabled:=false;
```

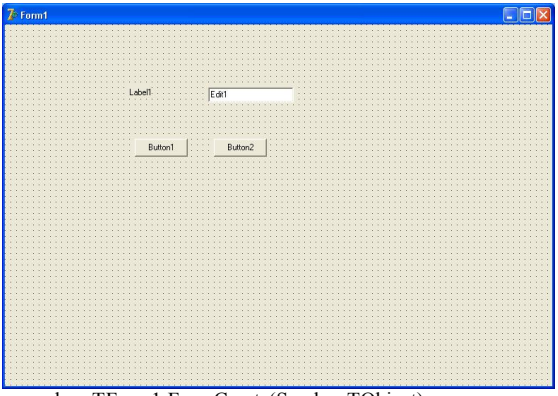
```
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
listbox1.Items.Add(Edit1.text);
end;
```





Default xassəsinə true qiymətinin mənimsədilməsi formanın hər hansı bir yerində Enter düyməsinin sıxılması bu düyməni aktiv hala gətirir.

Misal




```
procedure TForm1.FormCreate(Sender: TObject);
begin
 Edit1.PasswordChar:='*';
 {Enter duymesini sixdiqda Button1 aktiv olsun}
Button1.Default:=True;
 {ESC duymesini sixdiqda Button2 aktiv olsun}
 Button2.Cancel:=True;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
If Edit1.Text='1957' then
 begin
  ShowMessage('Shifre Dogrudur');
  Close;
 end
Else
 ShowMessage('Shifre Sehvdir')
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
Close;
end;
```

Proqramı işlətdərək şifrəni düzgün daxil edib Enter düyməsini sıxdıqdan sonra aşağıdakı mesaj pəncərəsini alarıq.

Şifrəni səhv daxil edib Enter düyməsini sıxdıqdan sonra isə aşağıdakı mesaj pəncərəsini alarıq.

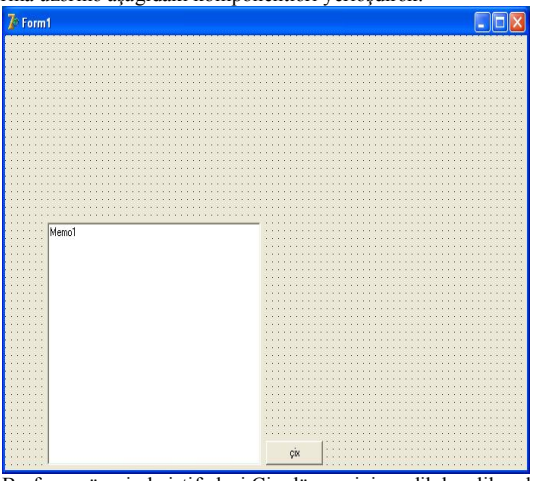
ESC düyməsinin sıxılması isə proqramdan çıxışı təmin edir.

ModalResult (TModalResult) xassəsi düymənin yerləşdiyi modal pəncərənin necə bağlanacağını göstərir. Bu xassə forma modal olaraq göstərilmişdirsə,işləyir.Forma pəncərəsinin modal olması bu pəncərə bağlanmadan proqram daxilində heç bir pəncərəyə keçidin mümkün olmadığını göstərir.Mesaj qutuları və InputBox qiymətlərin daxil edilməsi pəncərəsi belə pəncərələrdir.Formanın ShowModal xassəsindən istifadə edərək onu modal təyin etmək olar.Modal olaraq təyin edilən formalarda düymələrin ModalResult xassəsi aşağıda verilən qiymətlərə görə heçbir proqram kodu yazmadan formanı bağlayır və özünün seçildiyini formanın ModalResult xassəsinin köməyi ilə çağırıldığı proqrama bildirir.

MrNonr: Xassə aktiv deyil.

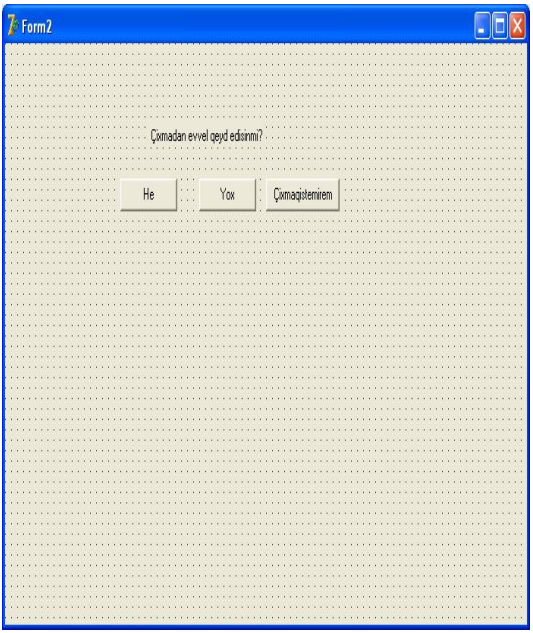
MrOk,mrCancel,mrAbort,mrRetry,mrIgnore,mrYes,mrNo

Misal.Proqramdan çıxarkən edilən dəyişikliyin qeyd edilib edilməyəcəyini istifadəçiyə bildirən bir mesaj pəncərəsi yaradaq.File ▶New▶ Application əmri ilə yeni bir proyekt yaradaq.Forma üzərinə aşağıdakı komponentləri yerləşdirək.



Bu forma üzərində istifadəçi Çix düyməsini seçdikdə edilən dəyişikliyin qeyd edilib edilməməsi üçün yeni bir forma üzərində aşağıdakı kimi özümüzə aid mesaj qutusu yaradaq.

File▶ New ▶Form əmri ilə yeni bir forma yaradaq.Bu formanın üzərində aşağıdakı komponentləri yerləşdirək.



Forma üzərindəki He düyməsinin ModalResult xassəsinə mrYes,Yox düyməsinin həmin xassəsinə mrNo və Çıxmaqistemirem düyməsinə isə mrCancel qiymətlərini mənimsədək.Düymələrə mənimsətdiyimiz bu qiymətlər düymə seçilərkən formanın ModalResult xassəsinə mənimsədiləcəkdir.Biz hansı düymənin seçildiyini formanın ModalResult xassəsi vasitəsi ilə öyrənəcəyik.Beləliklə formanın bağlanması üçün biz heç bir proqram kodu yazmayacağıq.Form1-dəki Çıx düyməsinin Click hadisəsinin emaledicisinə Form2-ni modal olaraq təyin olunması və düymə seçilərkən işləyəcək proqram kodlarını yazaq.Lakin proqramdakı bir formadan başqa bir formanı çağırmaq və ya ona aid xassələrdən istifadə etmək üçün Form1-in implementation hissəsinə uses unit2; sətrini əlavə edirik.

```
implementation

{$R *.dfm}
uses unit2;
procedure TForm1.Button1Click(Sender: TObject);
begin
form2.ShowModal;
if form2.modalresult=mrYes then
  begin
  memo1.Lines.SaveToFile('c:\students\a1.txt');
  close
  end;
if form2.modalresult=mrNo then
close;
if form2.modalresult=mrcancel then
  showmessage('cixmaq istemirem');
end;
```

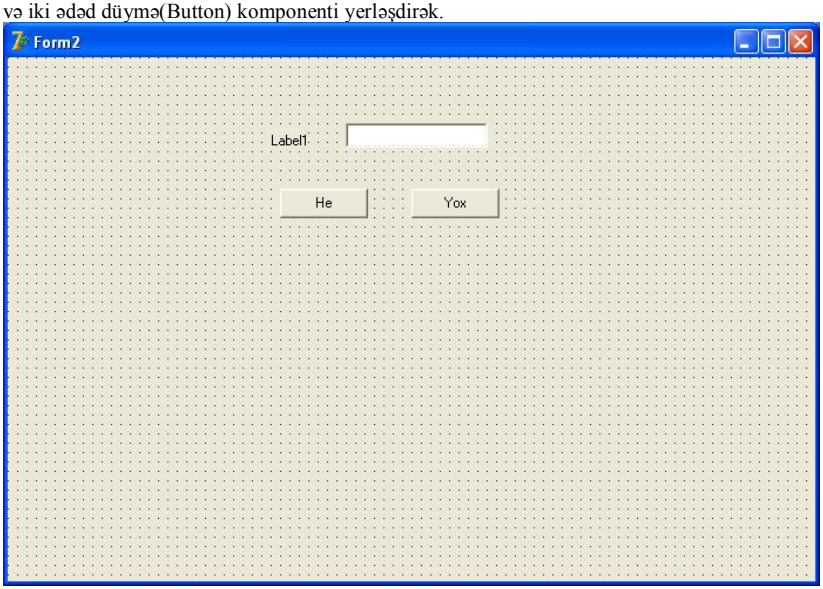
Formanın modal olaraq göstərilməsi ilə normal olaraq göstərilməsi arasında bir sıra fərqlər varldır.Bir forma normal olaraq(Show metodu ilə) göstərilirsə hər iki formada eyni anda işləyir. Modal olaraq(ShowModal metodu ilə) göstərilirsə,birinci forma modal olaraq göstərilən forma bağlanana qədər çalışmanı dayandırır.Aradakı fərqi proqramda ShowModal əvəzinə Modal yazaraq görə bilərsiniz.

Düymənin basılmasına reaksiya OnClick hadisəsinin emaledicisi tərəfindən verilir. Formada iki mətn sətri, bir nişan və bir düymə (Button1) komponentlərini yerləşdirək. Düymə üzərində iki dəfə vurmaqla OnClick hadisəsinin (emaledicisini yaradaq) mətn sahələrindən daxil edilmiş iki ədədin toplanması və nəticəni nişan komponentində yerləşdirən emaledicisini yaradaq.

```
begin
  Label1.Caption:=IntToStr(StrToInt(Edit1.Text)+StrToInt(Edit2Text));
end;
```

Misal.Proqramımız üçün şifrə forması yaradaq və proqrama girişin nə cür reallaşdırılması məsələsini həll etməyə çalışaq.

File ▶New▶ Application əmri ilə yeni bir proyekt yaradaq. File ▶New▶Form əmri ilə proyektə yeni bir forma əlavə edək.Ikinci formanın üzərinə bir mətn sətri(Edit),bir nişan(Label) və iki ədəd düymə(Button) komponenti yerləşdirək.



Mətn sətri komponentinin PasswordChar xassəsinə * qiyməti mənimsədək.He və Yox düymələrinin ModalResult xassəsinə uyğun olaraq mrOk və mrCancel qiymətlərini mənimsədək.

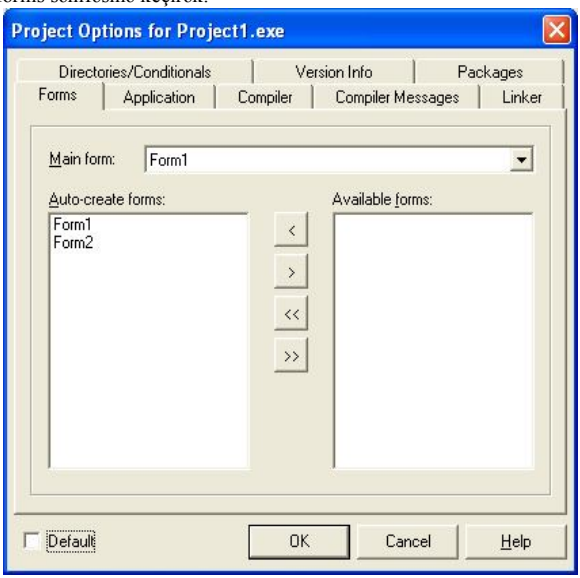
Indi isə Form2 daxilindəki düymələr üçün lazımlı proqram kodlarını yazaq.

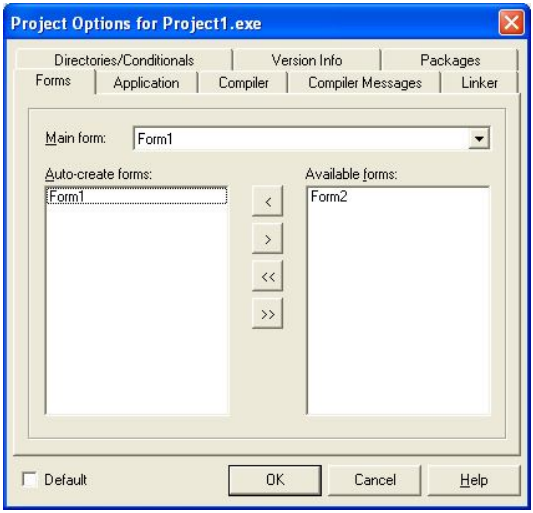
```
var
  Form2: TForm2;
  cehd:integer;

implementation

{$R *.dfm}
```

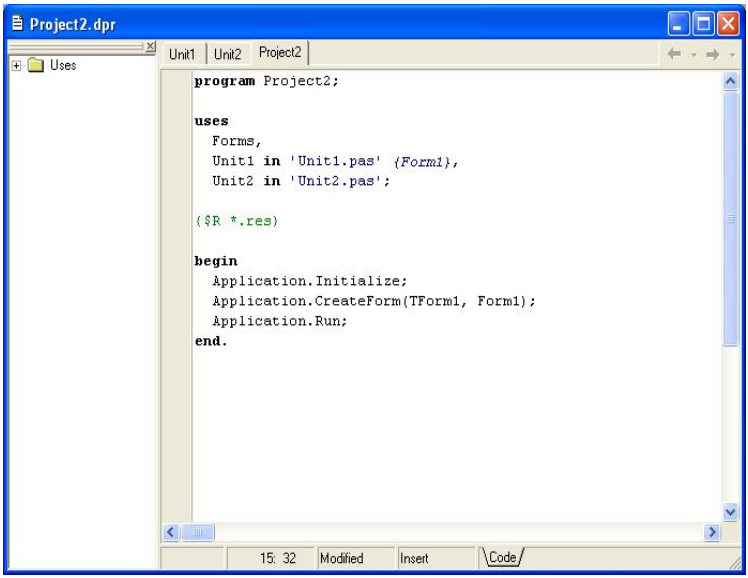
```
procedure TForm2.FormCreate(Sender: TObject);
begin
 caption:='sifre girin';
cehd:=3;
end;

procedure TForm2.Button1Click(Sender: TObject);
begin
if edit1.text='7897' then
   begin
   modalresult:=mrok;
   end
   else
   begin
   cehd:=cehd-1;
   modalresult:=mrretry;
   if cehd=0 then
   begin
   showmessage('programa her adamin girishi yoxdur');
   modalresult:=mrcancel;
   end;
   end;
end;
```

Proqramaımızda **cehd** adında bir qlobal dəyişənimiz var. **Cehd** dəyişəninin başlanğıc qiyməti 3-ə bərabər olub He düyməsi hər dəfə vurulduqda onun qiyməti bir vahid azaldılır.Əgər şifrə düz təyin olunmuşdursa,formanın ModalResult xassəsinə mrOk qiyməti mənimsədirik.Şifrə düzgün təyin olunmadıqda **Cehd** dəyişəni 0 qiyməti alana qədər proqrama daxil olmağa cəhd edirik.Bu halda geriyə mrOk qiyməti qaytarılmaması üçün ModalResult xassəsinə MrRetry qiyməti mənimsədirik.Əslində burada MrRetry qiyməti mənimsədilməsi o qədər də vacib deyil.Vacib olan mrOk və mrCancel-dən fərqli qiymət mənimənilməsidir.Çünki formanı çağırdığımız yer mrOk və ya mrCancel qiymətlərinə baxaraq şifrəni tapıb tapmadığımıza qərar verəcəkdir.Bu qiymətlərdən biri alınmamışdırsa,yenidən həmin formaya qayıdacaqdır. Cehd dəyişəni 0 qiyməti aldıqda şifrə düzgün təyin olunmamışdırsa, ModalResult xassəsinə mrCancel qiyməti mənimsədərik ki,bu da Yox düyməsinin sıxılmasına analojidir.

Indi proqramı işlətmək üçün F9 düyməsini sıxaq.Bu zaman şifrə pəncərəsi açılmayacaq.Bunun səbəbi şifrə pəncərəsini sonradan formamıza əlavə etdiyimizdən əsas formamızın Form1 olmasıdır.Əsas forma olaraq Form2 –ni götürsək bu zaman şifrə pəncərəsi açılacaqdır.Yenə də şifrə pəncərəmizi bağlaya bilmirik,çünki əsas formanı bağladıqda proqram sona yetir.Burada biz aşağıdakı metodikadan istifadə edirikProjet Options əmrləri ilə açılan pəncərəninAuto-create-forms səhifəsindən Form2-ni seçək və > düyməsini sıxaraq Available forms səhifəsinə keçirək.







Bu koda müəyyən əlavələr edərək Form2-ni özümüz yaradacağıq.

```
program Project2;

uses
  Forms,
```

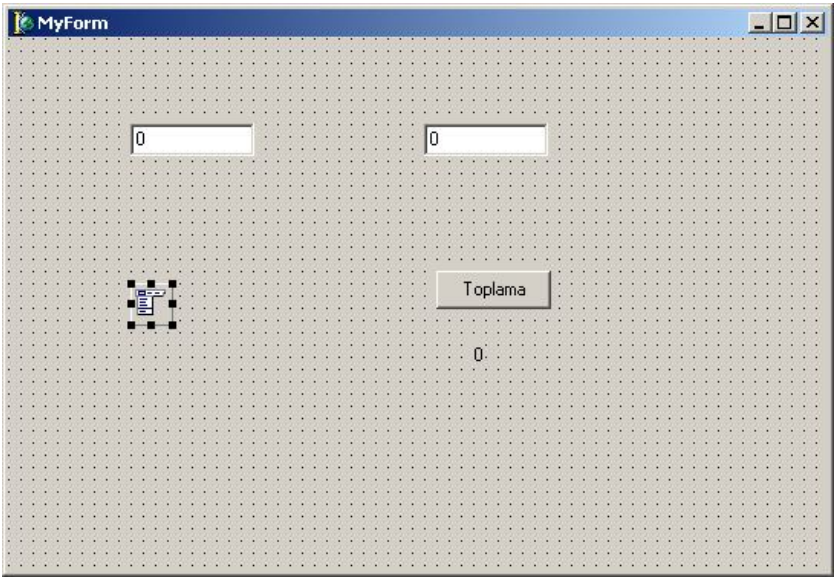
```
  Unit1 in 'Unit1.pas' {Form1},
  Unit2 in 'Unit2.pas' {Form2};

var
  islet:integer;

{$R *.res}

begin
  Application.Initialize;
  form2:=tform2.Create(nil);
  repeat
  islet:=form2.ShowModal;
  if islet=2 then
  begin
  form2.Free;
  halt;
  end;
  until islet=1;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

Yuxarıda gördüyümüz kimi əvvəlcə formanı özümüz yaradırıq və Repeat-Until dövr operatoru daxilində şifrə formamızı modal olaraq təyin edirik.Formadan mrOK qiyməti(1 qiyməti) qaytarılmayana qədər bu dövrdən çıxılmayacaq və dövrdən sonrakı sətirlər işləməyəcəkdir. Proqramm işləyərkən istifadəçi Yox düyməsini seçmişdirsə və ya formanı bağlayarsa MrCancel (ədədi qiyməti 2) qiyməti qaytarıldığından proqramadan çıxarıq.Bizim 3 dəfə şifrəni təyin etmək cəhdimiz başa çatdıqdan sonra da MrCancel qiyməti qaytarıldığından proqramdan çıxarıq.
Proqramı işlətdikdən sonra şifrəni(7897) 3 cəhddə düzgün daxil etməsək proqrama daxil ola bilmərik.

6.5.5. Menyu komponenti (TMain Menu)

Təyinatı. TMainMenu komponenti heç bir Windows əlavəsinin keçinəbilmədiyi proqrama menyu sətrinin əlavə olunması üçün nəzərdə tutulmuşdur.

Yaradılma üsulu: Proqrama menyu əlavə etmək üçün standard komponentlər panelindən TMainMenu komponentini seçib onu formanın istənilən yerində yerləşdirmək lazımdır. (şəkil 6.5)



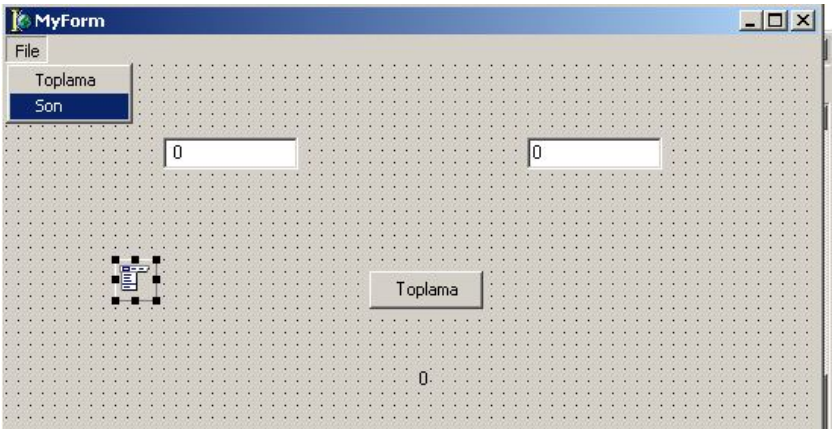
Şəkil 6.5. Formaya TMainMenu komponentinin əlavə olunması.

TEdit və TLabel vizual komponentlərindən fərqli olaraq,TMainMenu komponenti qeyri-vizual komponentdir. Bu o deməkdir ki, komponent formada böyük olmayan kvadrat şəklində görünsədə, yaradılmış proqramın pəncərəsində görünməyəcəkdir.

Menyu redaktorunu MainMenu1 obyekti üzərində mausun sol düyməsini iki dəfə vurmaqla çağırmaq olar.

Əvvəlcə menyu boş olur, obyektlər inspektorunun Caption(Başlıq) xassəsində menyunun birinci punktunu daxil edib, ENTER klavişini basırıq. Menyu redaktoru yenidən layihələndirilən menyuya qoşulacaq və ekranda birinci punkt görünəcəkdir.

Yenidən Enter düyməsi basılsa, Delphi 6, yenidən Caption xassəsinə qoşulacaq. Buraya növbəti adı daxil
edib(məsələnToplama). Enter düyməsi basılsa, menyunun
formalaşması təkrar olunacaq.



Şəkil 6.6 Layihələndirmə rejimində yaradılmış menyu proqramı.

Menyular arasında ayırıcı xət qoymaq üçün Caption xassəsində birinci mövqedə '-' (tire) işarəsi qoymaq lazımdır. Bir qayda olaraq axırda **Son** əlavə edilir. Yeni punktların əlavə edilməsi üçün INSERT, silinməsi üçün isə DELETE düymələrindən istifadə olunur. Layihələndirilən menyu üzrə hərəkət etmək üçün kursoru idarə klavişlərindən istifadə edilir. Menyu hazır olduqdan sonra redaktoru bağlamaq lazımdır.Menyunun strukturunu redaktə etmək üçün MainMenu1 obyektinə müraciət etmək lazımdır.

Menyudan istifadə. Punktlardan hər hansı birinin seçilməsi ilə hadisənin emaledicisini yaratmaq üçün menyu redaktorunda həmin punkt üzərində mausun sol düyməsini iki dəfə vurmaq lazımdır. (məsələn Son)

Delphi6 sistemi avtomatik olaraq yeni metod yaradacaqdır.

```
procedure MyForm.N1Click (Sender: TObject);
begin

end;
```

Menyunun punktlarını elan etmək üçün TMenuItem sinfi nəzərdə tutulmuşdur. N1-Son menyu punktunun proqram daxilində identifikatorudur. TMyForm sinfinin təsvirində o,

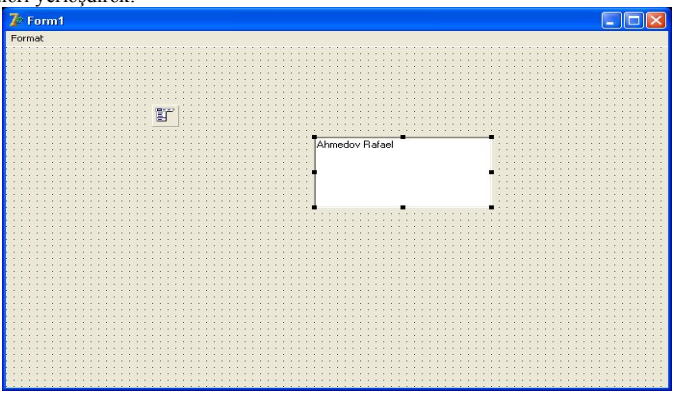
```
N1:TMenuItem;
```

şəklində göstərilir.

Son punktunun seçilməsi ilə əlavədən çıxmaq üçün N1Click metodunun reallaşması ilə TMyForm sinfinin Close metodunu çağırmaq lazımdır. TMyForm sinfi TForm sinfinin varisi sayılır.

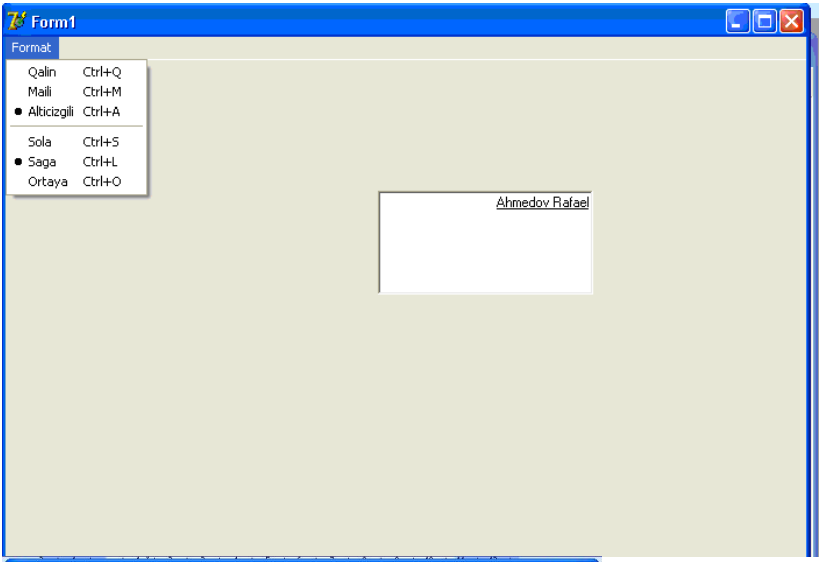
```
Procedure TMyForm.N1Click (Sender; TObject);
begin
close;
end;
```

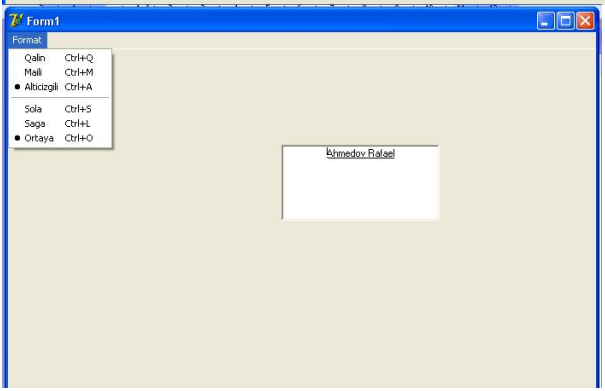
Misal olaraq Memo içindəki yazını qalın,maili , altı cizgili və solda ,sağda,ortada yerləşdirmək üçün bir menyu yaradaq. Bunun üçün forma üzərinə bir menyu və mətn sahəsi komponentləri yerləşdirək.




```
procedure TForm1.Qalin1Click(Sender: TObject);
begin
qalin1.Checked:=not qalin1.Checked;
IF qalin1.Checked THEN
Memo1.Font.Style:=Memo1.Font.Style+[fsBold]
else
Memo1.Font.Style:=Memo1.Font.Style-[fsBold];

end;

procedure TForm1.Maili1Click(Sender: TObject);
begin
maili1.Checked:=not maili1.Checked;
IF maili1.Checked THEN
Memo1.Font.Style:=Memo1.Font.Style+[fsItalic]
else
Memo1.Font.Style:=Memo1.Font.Style-[fsItalic];
end;

procedure TForm1.Alticizgili1Click(Sender: TObject);
begin
Alticizgili1.Checked:=not Alticizgili1.Checked;
IF Alticizgili1.Checked THEN
Memo1.Font.Style:=Memo1.Font.Style+[fsUnderline]
else
Memo1.Font.Style:=Memo1.Font.Style-[fsUnderline];
end;

procedure TForm1.Sola1Click(Sender: TObject);
begin
sola1.Checked:=not sola1.Checked;
memo1.Alignment:=taLeftJustify;
```

```
end;

procedure TForm1.Saga1Click(Sender: TObject);
begin
saga1.Checked:=not saga1.Checked;
memo1.Alignment:=taRightJustify;
end;

procedure TForm1.Ortaya1Click(Sender: TObject);
begin
ortaya1.Checked:=not ortaya1.Checked;
memo1.Alignment:=taCenter;
end;
```





Menyulardan sonradan istifadə etmək üçün qeydetmə

Hazırladığımız menyunu qeyd edib sonradan lazım gəldikdə yenidən istifadə edə bilərik.Hazırladığımız menyudan başqa proqramda da istifadə etmək istəyirsinizsə,menyu redaktoru pəncərəsində mausun sağ düyməsini sıxıb Save as Template səhifəsinə keçin.Əgər əvvəlcədən hazırladığımız menyudan istifadə etmək istəyirsinizsə,munyudan Insert From Template sətrini seçin.Burada yerləşən əmrlərin vəzifəsi aşağıdakından ibarətdir.

Insert əmri ilə istənilən yerə bir menyu yerləşdirilir

Delete əmri ilə kursor üzərində olduğu menyu silinir.

Create Submenu üzərində olduğumuz menyu üçün yeni altmenyu yaradılır

Select Menu forma üzərində yerləşdirilən menyulardan istəniləni seçilə bilər.

Save as Template yaratdığımız menyu qeyd ediləcək və lazım gəldikdə istifadə olunacaqdır.

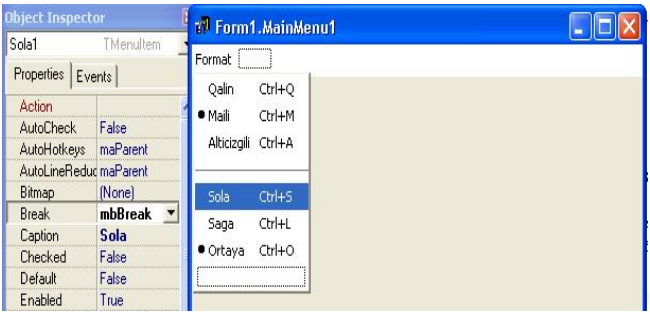
Insert From Template əvvəlcədən hazırlanmış menyulardan istifadə etmək üçün istifadə olunur.

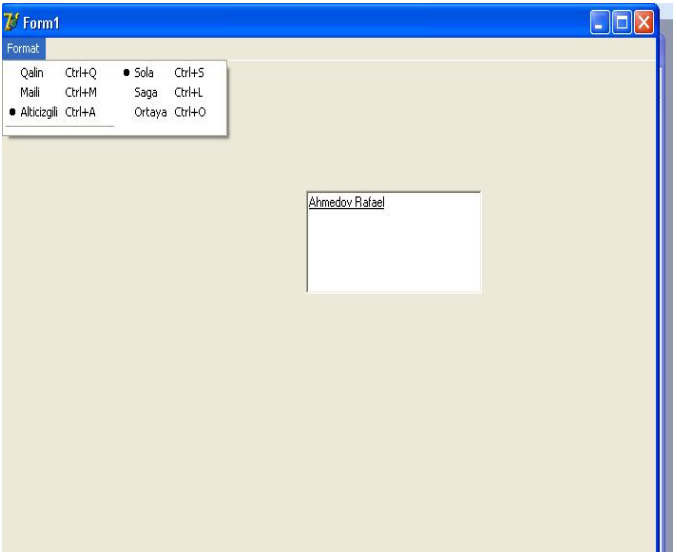
Delete Templates Insert Template pəncərəsindəki menyu işləmələrini silmək olar.

Insert From Resourse hər hansı proqramın menyuları menyu redaktoru formasına yerləşdirilir.

Iki sütunlu menyu yaratma

Əgər bir menyunun elementləri həddən artıqdırsa,onları iki sütunda yerləşdirmək olar.Ayırmaq istədiyimiz yerdəki menyu elementinin Break xassəsinə Object Inspector pəncərəsindən MbBreak qiymətini versək menyu həmin nöqtədən iki sütuna ayrılacaqdır.





Eyni işi əsas menyular üçün də etmək mümkündür.Əsas menyulardan birinin Break xassəsinə Object Inspector pəncərəsindən MbBarBreak qiymətini versək menyu həmin nöqtədən iki sütuna ayrılacaqdır.

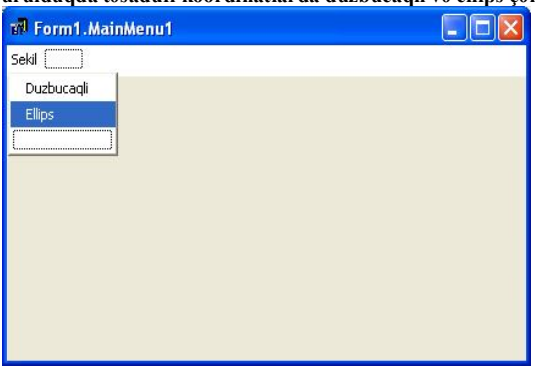
Hadisələr

OnDrawItem(Sender:Tobject;ACanvas:TCanvas;ARect:TRect;Selected:Boolean)

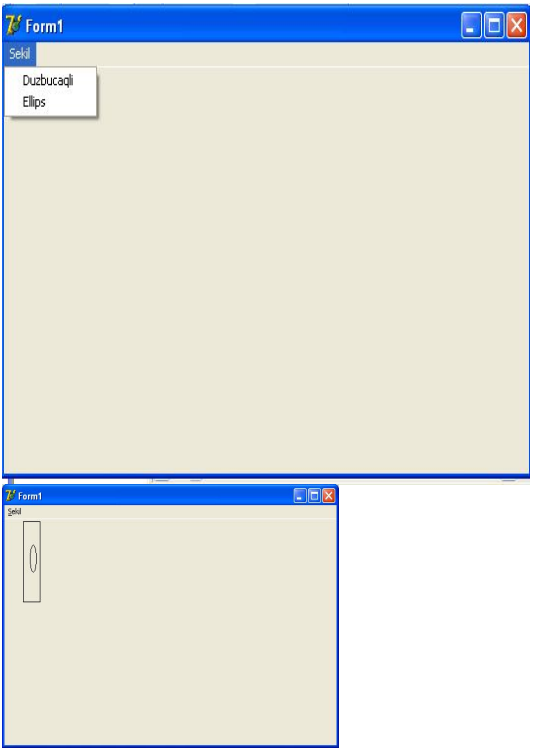
OwnerDraw xassəsi olan menyular ekranda göstərilərkən bu hadisə meydana gəlir.Bu hadisənin emaledicisinə yazacağınız kodla menyunun içərisində yazılacaq yazını özümüz tərtib edə bilərik.

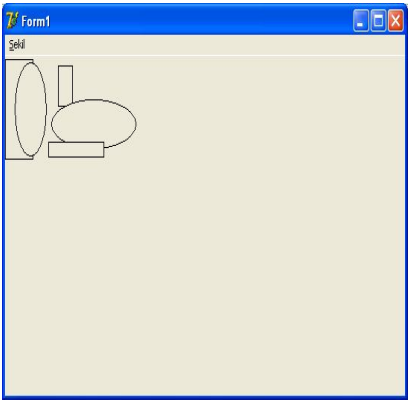
Rect parametri ilə menyunun koordinatlarını,Selected parametri ilə seçili olub olmadığını təyin edə bilərik. ACanvas parametri ilə şəkil çəkə bilərik. ACanvas parametri ilə şəkil çəkə bildiyimiz kimi yazı yaza bilər və ya istədiyimiz bir şəkli menyuya yerləşdirə bilərik.

Misal olaraq düzbucaqlı və ellips çəkə bilən menyu düzəldək.Menyuların üzərində vurulduqda təsadüfi koordinatlarda düzbucaqlı və ellips çəkilsin.



```
procedure TForm1.Duzbucaqli1Click(Sender: TObject);
begin
canvas.Rectangle(integer(random(left)+20),integer(random(top)+20),integer(r
andom(left)+30),integer(random(top)));
end;

procedure TForm1.Ellips1Click(Sender: TObject);
begin
canvas.Ellipse(integer(random(left)+10),integer(random(top)),integer(random(
left)),integer(random(top)));
end;
```

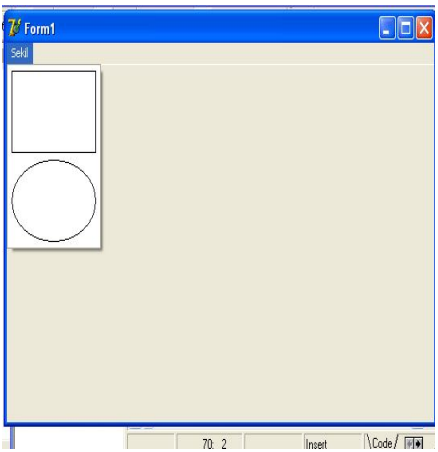




Indi isə menyularda mətn əvəzinə şəkil yerləşdirək.Bunun üçün menyuların OnDrawItem hadisəsinin emaledicisinə aşağıdakı proqram kodlarını yazaq.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
 mainmenu1.OwnerDraw:=true;
end;

procedure TForm1.Duzbucaqli1DrawItem(Sender: TObject; ACanvas:
TCanvas;
  ARect: TRect; Selected: Boolean);
begin
if selected then
Acanvas.Brush.Color:=clhighlight
  else
Acanvas.Brush.Color:=clmenu;
Acanvas.FillRect(Arect);
Inflaterect(arect,-3,-3);
Acanvas.Rectangle(arect.Left,arect.Top,arect.Right,arect.Bottom);
end;

procedure TForm1.Ellips1DrawItem(Sender: TObject; ACanvas: TCanvas;
  ARect: TRect; Selected: Boolean);
begin
if selected then
Acanvas.Brush.Color:=clhighlight
  else
Acanvas.Brush.Color:=clmenu;
Acanvas.FillRect(Arect);
Inflaterect(arect,-3,-3);
Acanvas.Ellipse(arect.Left,arect.Top,arect.Right,arect.Bottom);
end;

procedure TForm1.Duzbucaqli1MeasureItem(Sender: TObject; ACanvas:
TCanvas;
  var Width, Height: Integer);
begin
```

```
 width:=100;
 height:=80;
end;

procedure TForm1.Ellips1MeasureItem(Sender: TObject; ACanvas:
TCanvas;
 var Width, Height: Integer);
begin
 width:=100;
 height:=80;
end;
```



OnMeasure(Sender:Tobject;Acanvas:Tcanvas;var width,height:integer)

OwnerDraw xassəsi true olan menyular ekranda göstərilərkən bu hadisə meydana gəlir.Bu hadisənin emaledicisinə yazdığımız kodla menyunun ölçülərinin dəyişdirilməsini təmin etmək olar.

6.5.6. Mətn oblastı komponenti (TMemo)

Sadə mətn sətri (TEdit) komponenti ilə bir çox hallarda ötüşmək olmur. Istifadəçi böyük həcmli informasiya daxil etmək istəyirsə, ona bir neçə mətn sətri lazım gəlir. Bu halda TMemo komponentindən istifadə olunur. Mətn daxil edərkən yeni sətrə keçmək üçün bir qayda olaraq Enter klavişindən istifadə olunur. Lakin Windows-un dialoq pəncərəsində bu klavişdən daxil etməni sona çatdırmaq üçün istifadə edilir. ENTER klavişindən istifadə qaydası WantReturns xassəsinin qiyməti ilə təyin olunur. Əgər onun qiyməti true olarsa, onda Enter klavişi mətn oblastı daxilində növbəti sətrə keçməyə, əks halda isə daxil etmənin sona çatdığını və növbəti idarə elementinə keçidə xidmət edir. Bu zaman növbəti sətrə keçmək üçün Ctrl+Enter klavişilərinin kombinasiyasında istifadə olunur.

Bu komponentin əsas xassəsi Lines (sətir) TStrings tipinə malikdir. Bu xassədə istifadəçi tərəfindən daxil edilən sətirlərin siyahısı saxlanır. Bu sətirləri TStrings sinfində mümkün olan bütün metodlarla emal etmək olar. Məsələn, sətirləri faylda saxlamaq üçün:

Memol.Lines.Save ToFile (‘c:/memol.txt’);

Mətn oblastında sürüşdürmə düymələri varlığı ScrollBars xassəsinin köməyi ilə verilir.

Qiyməti	Mətn oblastının görünüşü
SsNone	Sürüşdürmə düymələri yoxdur
ssHorizontal	Üfiqi sürüşdürmə düyməsi vardır
SsVertical	Şaquli sürüşdürmə düyməsi vardır
SsBoth	Hər iki sürüşdürmə düyməsi vardır

Əgər üfqi sürüşdürmə düyməsi qoşulmuşdursa, Word Wrap xassəsinin qiyməti ləğv olunur. Bu xassə sağ sərhəddə çatdıqda avtomatik olaraq növbəti sətrə keçidi təmin edir. Lakin mətndə yeni sətrə heç bir simvol əlavə olunmayacaq keçid yalnız ekranda əks olunacaq.

Mətn oblastında mətn fraqmenti seçilərkən SelStart xassəsinə seçilmiş fraqmentin birinci simvolunun mövqeyi, SelLength xassəsinə isə seçilmiş simvoların sayı yazılır. Seçilmiş mətnə SelText (string tipi) xassəsi vasitəsilə müraciət etmək olar. Bütün mətnin seçilməsi üçün SelectAll, seçilmiş mətnin ləgv olunması üçün ClearSelection metodundan istifadə olunur.

Mətn oblastını təmizləmək üçün Clear, axırıncı dəyişikliyi ləgv etmək üçün Undo(dəyişəni özündə saxlayan) buferi təmizləmək və dəyişikliyi ləğv etməni aradan qaldırmağ üçün ClearUndo metodundan istifadə olunur.

Windows-un mübadilə buferi ilə işləmək üçün müəyyən qrup metodlar nəzərdə tutulmuşdur. Seçilmiş mətnin mübadilə buferinə yazılması üçün CopyToClip Board, mətnin kəsilməsi üçün CutToClipBoard, mətnin buferdən yapışdırılması üçün PasteFromClipBoard metodundan istifadə olunur.

6.5.7. Bayraqcıq komponenti (TCheckBox)

Təyinatı: Bu komponentdən iki mümkün variantdan birinin qeyd olunması üçün istifadə olunur. («qoşulmuşdur» və ya «qoşulmamışdır»).

Yaradılması: Bayraqcıq komponentinin nümunəsini formada yerləşdirdikdən sonra yaradılacaq elementə yazını Caption xassəsinin köməyi ilə yerləşdirmək olar. Yazının vəziyyəti Alignment xassəsinin köməyi ilə müəyyən edilir: taRighJustify qiyməti yazının sağ tərəfdə yerləşdiyini, taleft Justify isə yazının soldan yerləşdiyini göstərir. Bayraqcığın əsas xassəsi Checked adlanır. Onu proqramın layihələndirmə və işləmə mərhələsində dəyişdirmək mümkündür. Bu xassə əgər bayraqcıq qoşulmuşdursa true, qoşulmamışdırsa false qiymətini alır. Bəzən bayraqcıq «sanki qoşulmuşdur» kimi üçüncü bir vəziyyətdə də ola bilər ki, onun qoşulma əlaməti solğun rəngdir. Bu əlavə vəziyyət (məsələn, proqramın installsiyası zamanı onun bütün komponentlərinin installsiya olunmadığı bu vəziyyətin köməyi ilə çatdırılır).

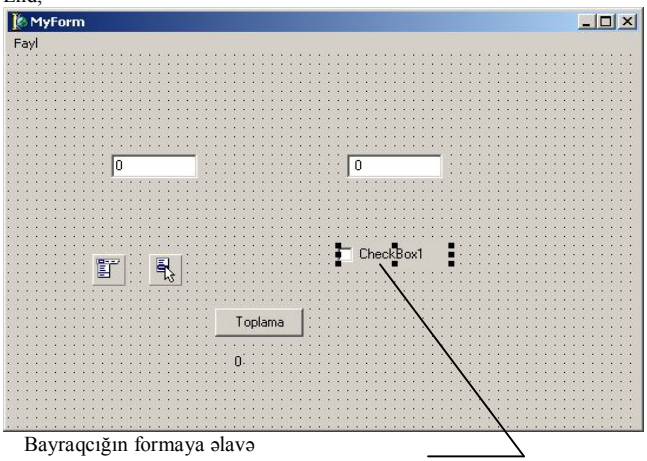
Əgər AllowGrayed xassəsi üçün true qiyməti verilmişdirsə, onda bayraqcıq üzərində siçanın sol düyməsini növbə ilə vursaq ardıcıl olaraq «qoşulmuşdur», «sanki qoşulmuşdur», «qoşulmamışdır» vəziyyətlərini alır. Cari vəziyyəti təyin etmək və ya mümkün variantlardan birini seçmək üçün State xassəsini yoxlamaq və ya dəyişdirmək lazımdır. OnClick hadisəsinin emaledicisinin köməyi ilə bayraqcığın vəziyyətinin dəyişməsini qeydə almaq olar.

State xassəsinin qiymətləri

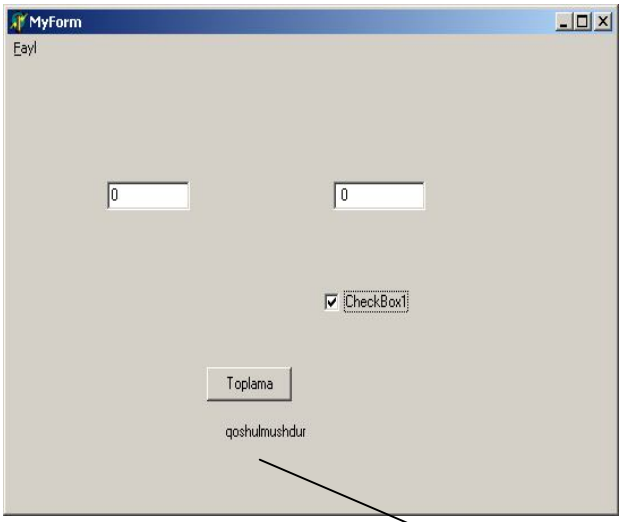
Qiymət	Bayraqcıq vəziyyəti
cbUncheked	qoşulmamışdır
cbGrayed	sanki qoşulmuşdur
cbChecked	qoşulmuşdur

Istifadəsi: Bayraqcığın vəziyyətinin dəyişməsi ilə onun cari vəziyyətini Label1 nişanına çıxaran misala baxaq. Bunun üçün forma üzərində uyğun komponenti verləşdirib, onun AllowGrayed xassəsinə true qiymətini mənimsədib, On Click hadisəsinin emal edicini formalaşdırıb ona bayraqcığın cari vəziyyətinin seçilməsi operatorunu əlavə edirik:

```
Procedure TForm 1.CheckBox1Click (Sender: Object);
Begin
   Case  checkBox1 State of
   CbUnchecked:Label1.Caption:= 'qoşulmamışdır';
   CbGrayed:Label1.Caption:= 'sanki qoşulmuşdur';
   CbChecked: Label1.Caption:= 'qoşulmuşdur';
End;
End;
```



Bayraqcığın formaya əlavə olunması.



Bayraqcığın vəziyyəti yazı şəklində təsvir olunmuşdur.

Şəkil 6.7 Proqram iş zamanı.

6.5.8. Çevirgəc komponenti (TRadioButton)

Təyinatı: Bayraqcıqdan fərqli olaraq TRadioButton komponenti bir neçə mümkün variantdan yalnız birinin seçilməsi üçündür. Onlar həmişə qrup şəklində istifadə olunurlar. Əgər istifadəçi onlardan birini seçirsə digər variantdan seçilmə götürülür.

Yaradılması: Çevirgəcləri qrup şəklində yaratmaq lazım gəldiyindən forma üzərində bir neçə TRadioButton komponenti yerləşdirək. Bu komponentlərdən yalnız biri seçilə bilər. Çevirgəc komponentinin bütöv xassələri bayraqcıq komponentinə analojidir. Alignment xassəsi Çevirgəcin sağda və ya solda yerləşdiyini göstərir. Checked xassəsi isə obyektin vəziyyətini təyin edir. Konkret çevirgəcin vəziyyətini izləmək üçün OnClick hadisəsini emal etmək olar.

Istifadəsi. Formada bir neçə Çevirgəc yerləşdirsək,proqram kompilyasiya və yerinə yetirildikdən sonra onlardan yalnız biri qalacaqdır. Layihələndirmə zamanı Çevirgəclərdən birini qoşulu vəziyyətdə qoyub, digərlərini susmaya görə qoşmamaq lazımdır.

Əgər dinamiki olaraq Çevirgəclərin vəziyyətini izləmək istəyiriksə, onda onlardan hər biri üçün OnClick hadisəsinin emal edicisini yaratmaq lazımdır. Çevirgəcin yeni vəziyyəti barəsində məlumatı, məsələn label1 nişanı vasitəsi ilə almaq olar.

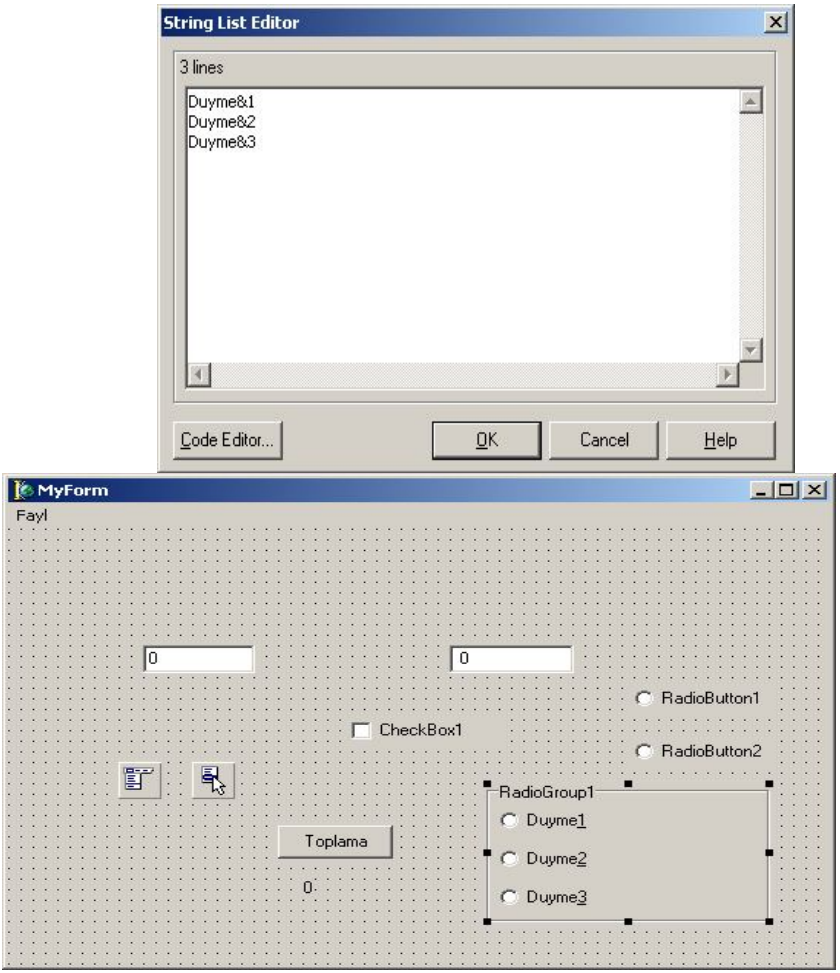
```
ProcedureTMyForm.RadioButton1Click (Sender:TObject);
begin
  If RadioButton1Checked then Label1.Caption:=’I-ci qoşulmuşdur’;
end;
  procedure TMyForm.RadioButton2Click (Sender: TObject);
begin
  if      RadioButton2.Checked          then      Label1.Caption:='II-ci qoşulmuşdur';
```

```
end;
```

6.5.9, Çevirgəc qrupu komponenti (TRadioGroup)

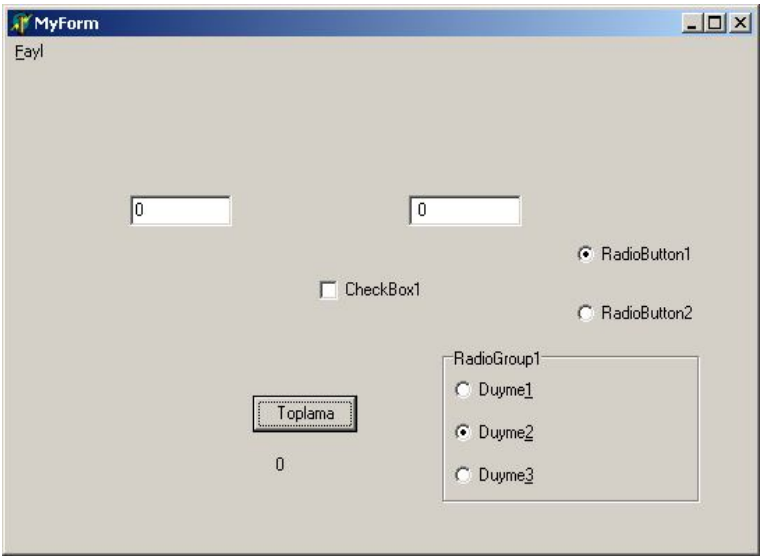
Təyinatı. Əgər proqramda bir neçə qrup Çevirgəcdən istifadə etmək tələb olunursa, aşağıdakı iki yanaşmadan birini tətbiq etmək olar. Birinci yanaşmada hər bir qrup üçün xüsusi obyekt (panel) seçilir və Delphi sistemi çevirgəclərin nə cür birləşdiyini buna əsasən başa düşür.

Ikinci yanaşma, Çevirgəclərin çevirgəc qrupunun işini təmin edən xassə və metodları özündə birləşdirən TRadioGroup komponentinin istifadəsinə əsaslanır . Forma üzərində TRadioGroup komponentini yerləşdirdikdən sonra onu təşkil edən çevirgəclər onların adlarının sadalanması ilə verilir. Bu adlar Tstring tipinə aid Items xassəsindən daxil edilir. Bir neçə sətir daxil olunduğundan onların daxil edilməsi üçün xüsusi redaktor nəzərdə tutulmuşdur. Bu redaktor Items xassəsi yerləşən sətirin sağ tərəfindən  xüsusi düyməsi üzərində mausun sol düyməsini vurmaqla çağrılır. Proqramın klaviaturanın köməyi ilə idarə etmək üçün adlarda hər hansı hərfin qarşısında $ işarəsi qoymaq olar. Redaktor pəncərəsində adları daxil etdikdən sonra OK düyməsini vursaq RadioGroup obyektinin forma üzərində görünüşü dəyişəcəkdir.



Şəkil 6.8 Iki qrup çevirgəclərin yaradılması.

Proqramı kompilyasiya edib yerinə yetirsək, onda iki qrup çevirgəc işləyəcəkdir. Birinci qrup Radio Button1 və RadioButton2 ayrı-ayrı elementlərdən təşkil olunmuş, ikinci qrup RadioGroup1 isə tam vahid obyekt kimi formalaşdırılmışdır.



Şəkil 6.9 Proqram iş zamanı. Iki qrup çevirgəclərin müxtəlif vasitələrdən istifadə edilərək yaranması.

Istifadəsi. RadioGroup komponenti özünün çevirgəcləri ilə birlikdə tam vahid təşkil edir və istifadəsi RadioButton komponentindən fərqlənir. Caption xassəsi çevirgəc üzərində olan yazını təyin etmir (bu yazılar Items xassəsində verilir), qrupun başlığını təyin edir (ilkin olaraq o RadioGroup1 adlanır). Columns xassəsi çevirgəclər təşkil olunan sütunların sayıdır. ItemIndex xassəsi (ilkin qiyməti-1) seçilmiş çevirgəcin nömrəsini göstərir (-1 ədədi heç bir çevirgəcin seçilmədiyini göstərir). Bu xassənin qiyməti avtomatik olaraq istifadəçi çevirgəclərdən birini seçdikdə dəyişir. Onu proqramdan da dəyişmək olar: Item Index xassəsinə yeni qiymət mənimsətsək, forma üzərində seçilmiş cari çevirgəcdə dəyişəcəkdir.

Yeni çevirgəcin seçilməsini dinamiki olaraq izləməyə qrupda OnClick hadisəsinin emaledicisinin köməyi ilə nail olmaq olar. Məsələn, cari çevirgəcin adını Label1 nişanı üzərində təsvir etmək üçün, Items (sətirlər siyahısı) xassəsinə müraciət edib, ItemIndex xassəsində yazılan elementi seçmək lazımdır. Əvəlcədən nömrəsi seçilmiş çevirgəcin olub olmadığını yoxlamaq (Item Index xassəsinin qiymətinin –1bərabər olub olmadığını) və ya layihələndirmə mərhələsində çevirgəcin ixtiyari birini seçmək lazımdır (məsələn ItemIndex xassəsinə 0 qiymətini mənimsətməklə).

```
Procedure TMyForm.RadioGroup1Click(Sender:Tobject);
Begin
 If RadioGroup1.ItemIndex > -1 then

Label1.Caption:=’Secilmishdir‘+RadioGroup1.Items[RadioGroup1.ItemIndex]
End;
```

6.5.10. Sürüşdürmə çubuğu (TScroll Bar).

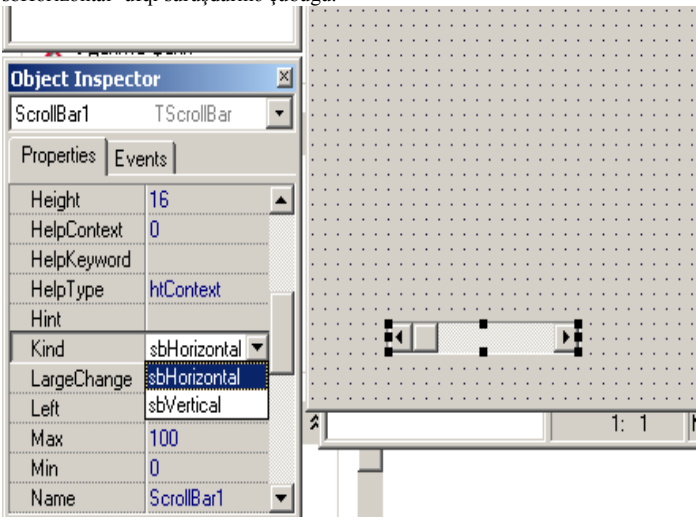
Üfqi və şaquli sürüşdürmə çubuqlarından verilənlər bazasının yazıları üzərində irəli və ya geriyə hərəkət etməklə yanaşı, hər hansı mətndə və ya şəkildə sürüşdürmə üçün istifadə olunur.

Xassələri:

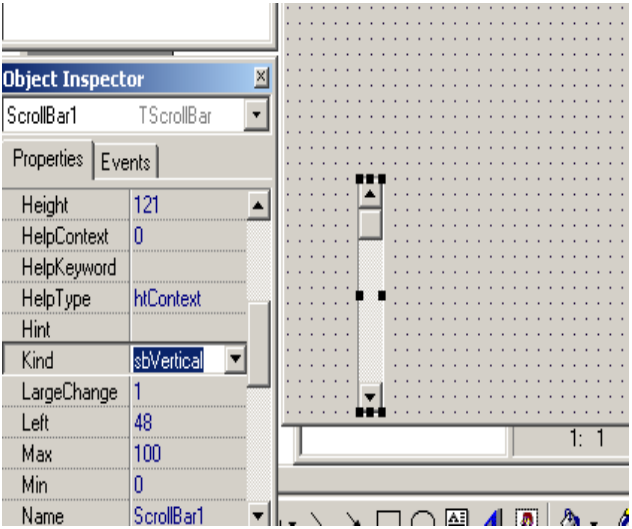
Kind– sürüşdürmə çubuğunun tipini müəyyən edir. O, iki qiymətdən birini alır.

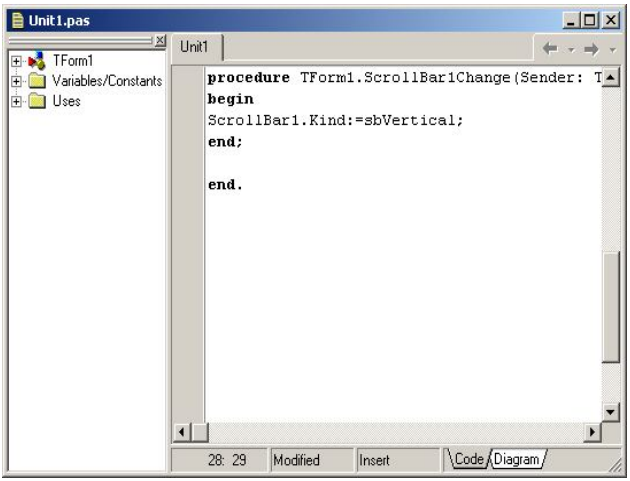
sbVertical- şaquli sürüşdürmə çubuğu.

sbHorizontal- üfqi sürüşdürmə çubuğu.



Min-Max xassələri sürüşdürmə çubuqlarının ala biləcəyi minimum və maksimum qiymətləri təyin edir.





Sürüşdürmə çubuğunun əhatə etdiyi qiymətlər diapazonu Min(minimal qiymət)və Max(maksimal qiymət) xassələri vasitəsi ilə verilir.Sürüşdürmə çubuğunun cari vəziyyəti Pozition xassəsinin köməyi ilə verilir(bu qiymət Min ilə Max diapazonunda dəyişməlidir)və müxtəlif üsullarla dəyişdirilə bilər.

- Sürüşdürmə çubuğunun uclarındakı iki ox ilə azaldılıb-artırıla bilər. Bu hadisə SmallChange adlanır.
- Şaquli sürüşdürmə çubuğu üzərində mausun düyməsini sıxaraq, bu qiymət dəyişdirilə bilər. Bu hadisə LargeChange adlanır.

- Sürüşdürmə çubuğunun üzərindəki qutucuq mausun köməyi ilə istənilən vüziyyətə gətirilərək qiymət azaldılıb-artırıla bilər. Bu hadisə ScrollChange adlanır.

Min,Max və Pozition xassələrinin qiymətlərini SetParams metodunun köməyi ilə təyin etmək olar.

```
Procedure SetParams(Aposition,Amin,Amax:Integer);
```

Misal olaraq, biz ScrollBar komponenti ilə Edit1 komponenti içərisində qiymətin dəyişdirilməsinə baxaq.

```
Procedure Tform1.ScrollBar1Change(Sender:TObject);
  Begin
Edit1.text:= İntToStr(ScrollBar1.Position);
   End;
```

Edit1 içərisindəki qiyməti deyil, Edit1-in Left xassəsinə ScrollBar komponentinin Position xassəsi vasitəsi ilə nəzarət etsək, Edit1 koordinatlarını dəyişdirə bilərik.

```
Edit1.Left:=ScrollBar1.Position;
```

6.5.11. TPopupMenu (Kontekst menyu)

Bu komponent MainMenu komponentinə oxşardır. Eyni qayda ilə PopupMenu yaradılır. MainMenu-dan fərqli olaraq formanın başlığında deyil, istənilən yerində açıla bilər.

Kontekst menyunun açılması üçün üç bir-birindən fərqli üsul vardır.

1.Kontekst menyu komponenti üzərinə qoyulduğu komponentin PopupMenu xassəsinə bu kontekst menyunun adı verilir. Kontekst menyunun bağlı olduğu komponent üzərində mausun sağ düyməsini basdıqda kontekst menyu açılır.

Formanın üzərində PopupMenu-nun aktiv hala gəlməsi üçün Object Inspector pəncərəsində formanın PopupMenu xassəsinə bu menyunun adı verilir. Bu iş proqram kodu vasitəsi ilə də görülə bilər:

```
Form1.PopupMenu:=PopupMenu1;
```

2.Komponentin MouseDown hadisəsinə proqram kodu yazılaraq mausun sağ düyməsinin basılıb basılmadığı yoxlanılır və PopupMenu1.Popup metoduyla menyu lazımi koordinatda aktiv hala gətirilir.

```
ProcedureTForm1.Edit2MouseDown(Sender:TObject;Button:
          TMouseButton;Shift:TShiftState; x,y:INTEGER);
  Var
    Yer:TPoint;
  Begin
```

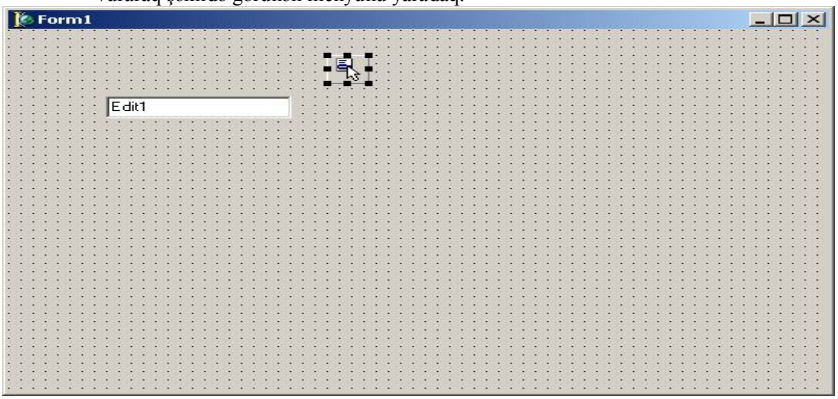
```
        If Button=mbRight then
          Begin
           Yer:=Edit2.ClientToScreen(Point(x,y));
            Popup Menu1.Popup(yer.x, yer.y);
          End;
      End;
```

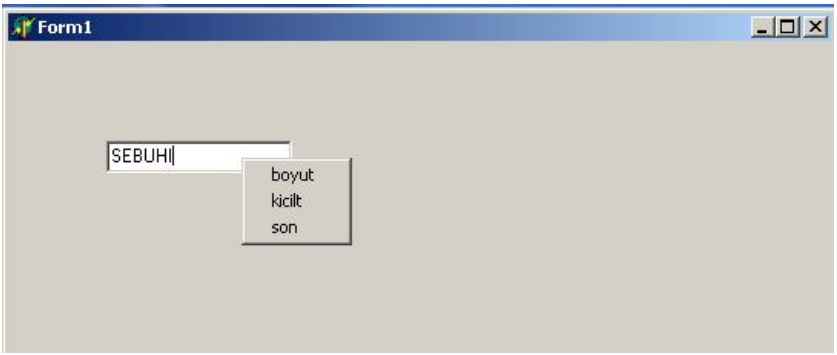
3.Komponentin OnContextPopup hadisəsində PopupMenu1.Popup metoduyla menyu lazımi koordinatlarda aktiv hala gətirilir.

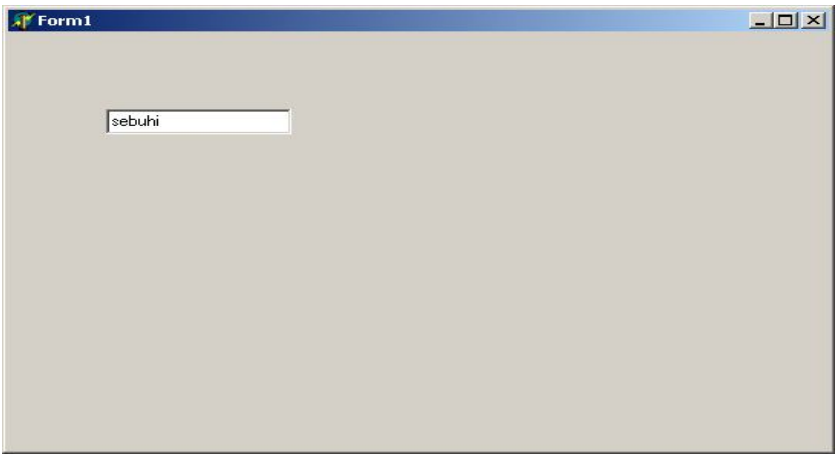
```
Procedure Tform1.Edit1ContextPopup(Sender:TObject; Mouse Pos: TPoint; var
                Handled: Boolean);
Var
   Yer:TPoint;
 Begin
  Yer:=Edit1.ClientToScreen (MousePos);
   Popup Menu1.Popup(yer.x, yer.y);
   Handled:=true;
End;
```

Nümunə: Edit mətn sətri içərisində mausun sağ düyməsini basdıqda Böyük hərfə çevir və kiçik hərfə çevir variantları açılan bir menyu yaradaq.

Formaya bir PopupMenu yerləşdirib, mausun sol düyməsini onun üzərində vuraraq şəkildə görünən menyunu yaradaq:







Menyuya aid Click hadisələrinə aşağıdakı proqram kodlarını yazaq.

```
Procedure Tform1.Form Create(Sender:TObject);
Begin
Edit1.PopupMenu:=PopupMenu1;
End;
Procedure Tform1.BoyutClick (Sender:TObject);
Begin
Edit1.Text:=Upper Case (Edit1.Text);
End;
Procedure Tform1.Kicilt Click (Sender:TObject);
Begin
```

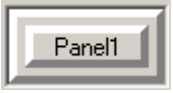
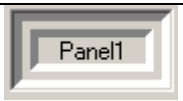
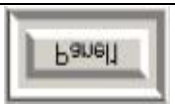
```
    Edit1.Text:=LowerCase(Edit1.Text);
    End;
```

Proqram istəldikdən sonra Edit1 üzərində mausun sağ düyməsini bassaq hazırladığımız menyu açılacaq və uyğun proqram kodunu işlədəcəkdir.

6.5.12. TPanel(Panel) komponenti

Bu komponentin üzərinə müxtəlif komponentlər yerləşdirilə bilər. Yəni bir neçə komponenti qruplaşdırmaq üçün nəzərdə tutulmuşdur.

Panel komponentinin BevelInner və BevelOuter xassələrinin köməyi ilə panelin daxili və xarici çərçivələrinin forması müəyyənləşir. Bu iki xassənin aldığı qiymətlər və görünüşü belədir.

		BevelOuter	
		BvLowered	BvRaised
BevelInner	BvRaised		
	BvLovered		

BevelWidth və BorderWidth xassələri ilə çərçivələrin eni verilir.
Caption panelin üzərindəki yazı bu xassənin köməyi ilə verilir.

6.5.13. Siyahı komponenti (TListBox)

Təyinatı: Siyahı komponentindən WINDOWS əlavələrində tez-tez istifadə olunur. Bu komponent siyahıdan bir və ya bir neçə sətri seçməyə imkan verir.

Yaradılması. Ilkin olaraq TListBox komponenti forma üzərində boş kvadrat şəklində görünəcək. Onun ölçülərini mausun köməyi ilə nizamlamaq olar.

Siyahı bir neçə sütundan ibarət ola bilər. Bu heçdə hər bir sütunun ayrıca siyahıdan ibarət olması deyil, sadəcə olaraq siyahının görünən hissəsində aşağı sətirlərdə olan siyahı elementlərinin növbəti sütunda görünməsi ilə əlaqədardır.

Sütunların sayı Columns xassəsində verilir.Əgər o ,0-dan böyükdürsə,onda hər bir sütuna siyahının ümumi eninin müəyyən hissəsi ayrılır.(Width xassəsinin qiyməti bölünsün sütunların sayı qədər)

Siyahı üç mümkün tipdən birinə məxsus ola bilər (Style xassəsi).

Cədvəl: Style xassəsinin qiymətləri

Qiyməti	Siyahı üslubu
LbStandard	Standart siyahı
LbOwnerDrawFixed	Siyahının hər bir elementi qeyd olunmuş hündürlüyə malikdir,lakin onun təsviri proqramçı tərəfindən təyin edilir.
LbOwnerDrawVariable	Proqram mətnində siyahının hər bir elementinin ölçüsü verilməlidir ki,bu da müxtəlif ölçülü siyahı elementləri yaratmağa imkan verir.

Əgər Style xassəsinin qiyməti lbStandard-a bərabər deyilsə, onda istifadəçi özü OnDrawItem hadisəsinin emaledicini formalaşdırmalıdır. Belə siyahı təkcə mətin deyil digər obyektləri də öz daxilinə ala bilər. Elementin hündürlüyünü ItemHeight xassəsinin köməyi ilə öyrənmək olar.

Əgər siyahının elementlərinin hündürlüyü müxtəlifdirsə, onda OnDrawItem hadisəsinin emalından əvvəl OnMeasureItem hadisəsini emal etmək lazımdır. Uyğun emaledicinin başlığında belə bir parametr təsvir olunur:

Var Height: Integer;

Bu parametrə, növbəti elementin ölçüsünü yazıb (onun nömrəsi Index parametrində göstərilir), elementin cari hündürlüyünü nəzərə almaqla OnDrawItem hadisəsini emal etmək lazımdır.

Bir qayda olaraq siyahının elementləri bir-bir seçilir, bir dəfəyə bir neçə elementin seçilməsinə MultiSelect (məntiqi tip) xassəsi imkan verir (əgər onun qiyməti True olarsa) .

SelCount xassəsi siyahının seçilmiş sətirlərinin sayını göstərir. Konkret elementin seçilib-seçilmədiyini yoxlamaq üçün, onun nömrəsindən istifadə edib Boolean tip massivdən ibarət Selected xassəsinə muraciət etmək lazımdır.ListBox.Selected[4] ifadəsinin qiyməti true olarsa, siyahıda 5-ci element seçilmişdir(hesabat 0-dan başlayır). Əgər bir neçə elementin seçilməsinə icazə verilmirsə, onda hansı elementin seçildiyini öyrənmək üçün yeganə seçilmiş elementin indeksini özündə saxlayan ItemIndex xassəsinə müraciət etmək lazımdır.

Başlanğıcda siyahıda seçilmiş element olmur. Bu o deməkdir ki, ItemIndex xassəsinin qiyməti –1-dir.

Istifadəsi: Siyahının məzmunu Items xassəsində saxlanılır (sətirlər siyahısı, Tstrings sinfi). Sətirləri layihələndirmə mərhələsində xüsusi redaktorun köməyi ilə verməklə yanaşı, proqram işləyərkən TStrings sinfinin Add metodunun köməyi ilə də vermək olar.

Məsələn, əgər siyahıya toplamanın nəticəsini yazmaq istiyiriksə, onda formaya yeni ElaveET(o,avtomatik olaraq Button2 adını alacaqdır) düyməsi yerləşdirib bu düymənin basılmasının emaledicisində (OnClick hadisəsi) Add metodunu çağırmaq lazımdır.

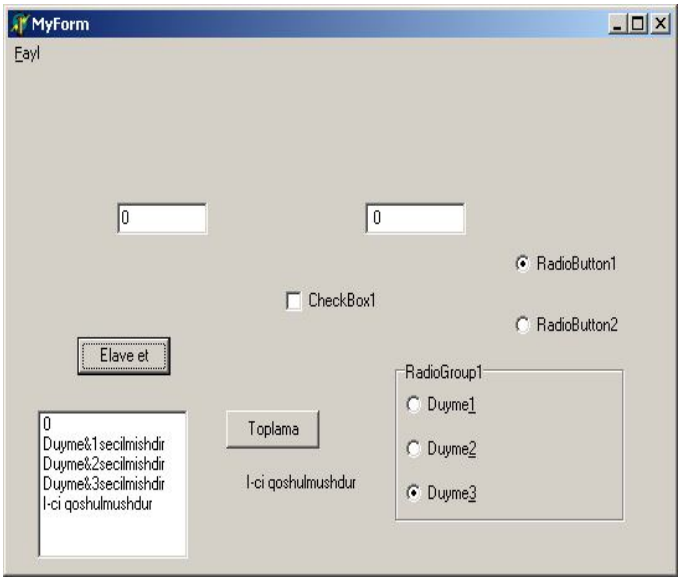
```
Procedure TmyForm.Button2.Click(Sender:TObject);
Begin
ListBox1.Items.Add(Label1.Caption);
End;
```

Siyahının elementlərini Sorted xassəsinə True qiyməti verməklə əlifbaya görə nizamlamaq olar. Bundan sonra yeni element daxil olarkən avtomatik olaraq nizamlanır.

Bütün siyahının məzmununu yox etmək üçün Clear metodundan istifadə olunur.

ListBox1.Clear;

Konkret elementin yox edilməsi üçün siyahının Items xassəsində dəyişiklik edilməklə,DeleteString metodundan istifadə edilir. ListBox.DeleteString (4);



Şəkil 6.10Aşagı sol tərəfdə nəticələrin siyahısının verilməsi

6.5.14. Siyahılı sahə komponenti (TComboBox)

Bu komponent seçilmiş elementi təsvir olunan köməkçi sahə qoşulmuş siyahı komponentinin bir variantıdır. Bu sahədən yeni elementin daxil edilməsi və ya lazımi elementin başlanğıc simvoluna görə cəld axtarışını təşkil etmək üçün istifadə etmək olar. TComboBox siyahılı sahə komponenti TListbox siyahı və TEdit mətn sətri komponentinin kombinasiyasından ibarət olduğundan onun əsas xassə və metodları bu iki komponentlərdən götürülmüşdür. Komponentin Style xassəsi ilə təyin olunan beş modifikasiyası vardır: csSimple, csDropDOWN, csDropDownList, csOwnerDrawFixed və csOwnerDrowVariable.

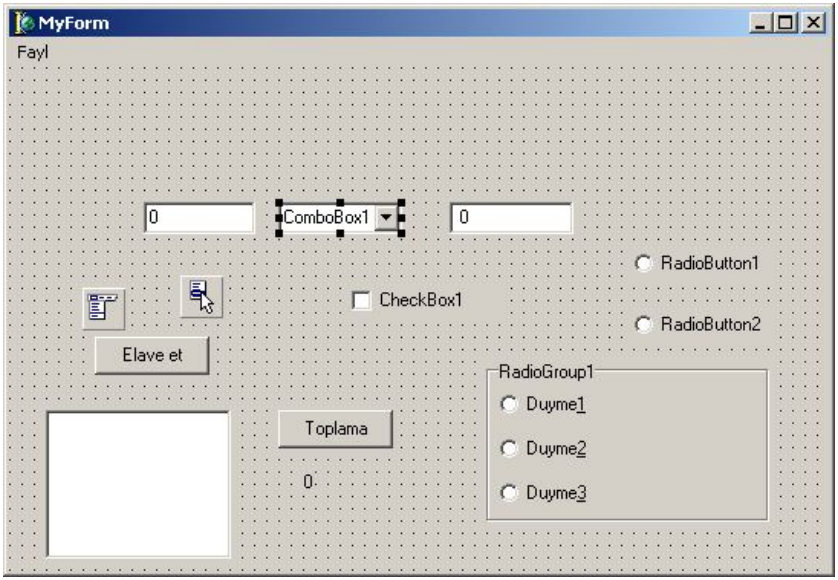
CSSimple Aşağıya doğru açılmayan bir ComboBox görünür. Istifadəçi siyahıya müraciət edə bilir. Aşağıya doğru açılmadığı üçün istifadəçi yuxarı və aşağı kursoru idarə düymələri ilə seçim edə bilər.

CSDropDownList Aşağıya doğru açılan bir ComboBox görünür. Istifadəçi siyahıya müraciət edə bilmir. Siyahıdakı bütün verilənlər eyni hündürlüyə malikdir.

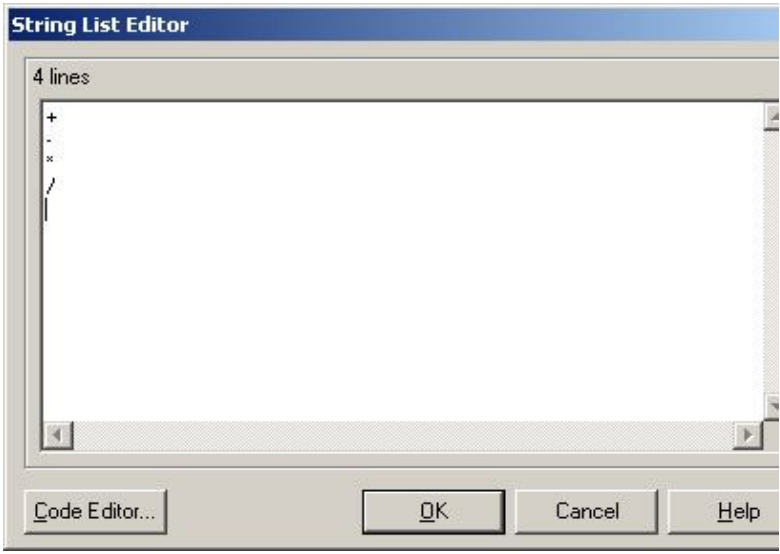
CSDropDown Aşağıya doğru açılan bir ComboBox görünür. Istifadəçi siyahıya müraciət edə bilir. Siyahıdakı bütün verilənlər eyni hündürlükdədir.

CSOwnerDrownFixed Aşağıya doğru açılan bir ComboBox görünür. Siyahıdakı verilənlərin hündürlükləri Item Meight xassəsi ilə dəyişdirilə bilər. Istifadəçi siyahıya müraciət edə bilmir.

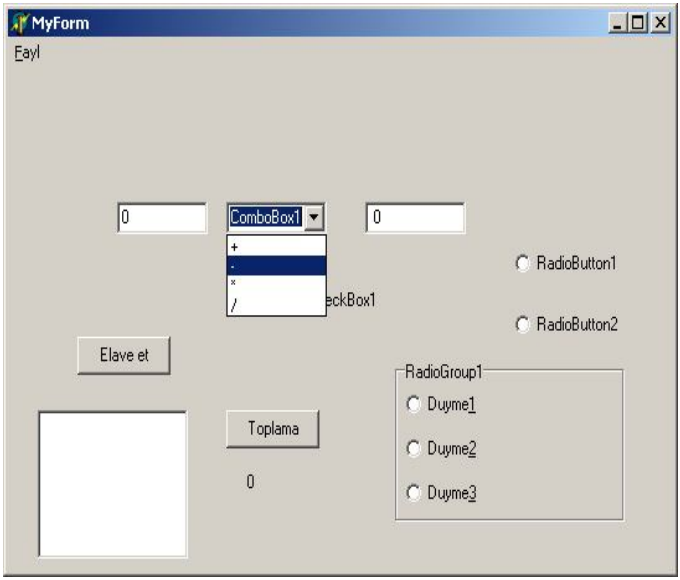
CSOwnerDrownVariable Aşağıya döğru açılan bir ComboBox görünür. Istifadəçi siyahıya müraciət edə bilmir. Siyahıdakı verilənlərin hündürlükləri müxtəlif ola bilər.



Şəkil 6.11 Siyahılı sahə komponentinin formaya yerləşdirilməsi



Şəkil 6.12 Redaktora siyahının əlavə olunması.



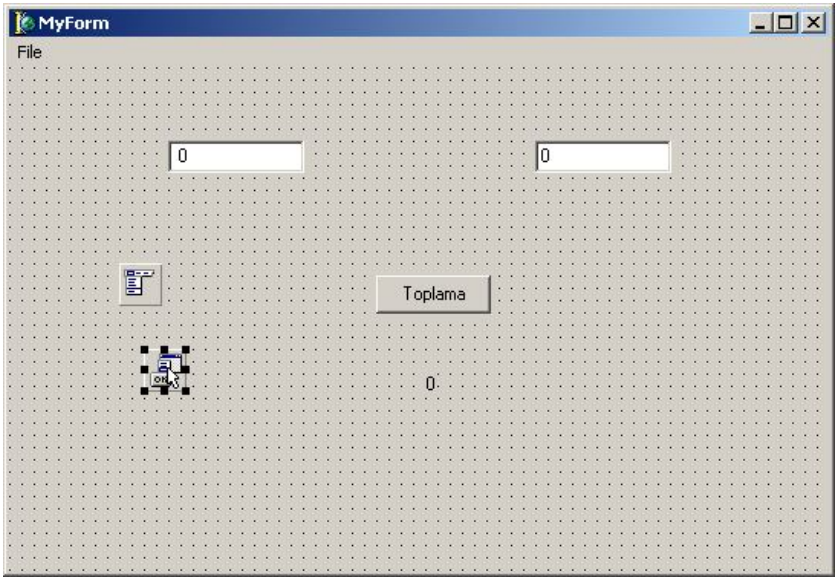
Şəkil6.13 Siyahıdan elementin seçilməsi.

6.5.15. Əməliyyatlar siyahısı komponenti(TActionList)

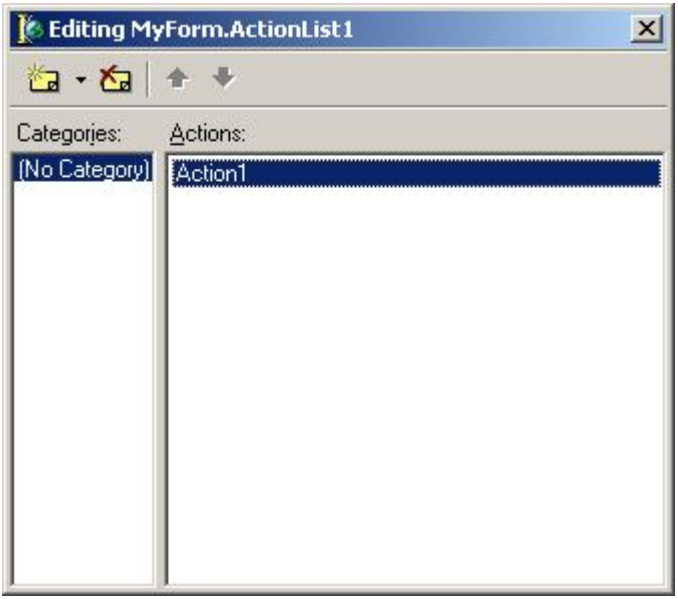
Bu komponent ilk dəfə Delphi 4 versiyasında daxil edilmişdir. Komponent proqram çoxlu sayda mümkün hadisələri emal etdikdə, və eyni əməliyyatlar müxtəlif istifadəçilər tərəfindən yerinə yetirildikdə , məsələn düyməyə vurmaq və ya menyular sətrindən əmrin seçilməsi istifadə olunur. Bu şəraitdə bütün vacib hadisələri mərkəzləşdirilmiş saxlamaq və hadisələrə reaksiyanı bir obyektdə formalaşdırmaq lazımdır.

Yaradılması. TActionList komponentini formada yerləşdirmək digər komponentlərə analojidir. Komponent görünməyən olduğundan onun üçün çox yer tələb olunmur.

Istifadəsi. ActionList1 obyekti üzərində mausun sol düyməsini iki dəfə vurduqda redaktor pəncərəsi açılır. Bu obyektlər məcmuundan ibarət standart redaktordur. Yeni əməliyyat daxil etmək üçün açılan kontekst menyudan NewAction düyməsinə vurmaq lazımdır. Categories bölməsində None sətri yaranır. Delphi 6 sistemində iki kateqoriya əməliyyatlar vardır: standart və istifadəçi tərəfindən təyin olunan əməliyatlar.



Şəkil 6.14: TActionList komponentinin formada yerləşdirilməsi



Şəkil 6.15 Əməliyyatlar siyahısının redaktə olunmas

Actions bölməsində Action1 sətri vardır. O, TAction sinfinin yeni obyektini təyin edir və onun üçün obyektlər inspektoru pəncərəsində bir sıra xassələri sazlamaq lazımdır. Hər şeydən əvvəl bu Caption xassəsidir. Başlığın qiyməti &Toplama sətri olacaqdır. Obyektin adını(Name xassəsi) daha çox uyğun gələn AddAction dəyişdirmək olar.

Nəhayət verilmiş obyektlə verilən əməliyyatı təyin etmək lazımdır. Bunun üçün OnExecute hadisəsinin emaledicisini formalaşdırmaq lazımdır. Orada emaledicinin iş məntiqi göstərilir:

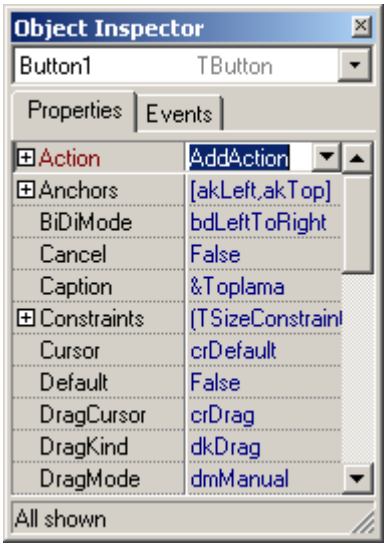
```
ProcedureTMyForm.AddActionExecute(Sender:TObject);
begin Label1.Caption:=InToStr(StrToInt(Edit1.Text)+StrToInt(Edit2.Text));
end;
```

Proqram kodu pəncərəsində Toplama düyməsinin ,MainMenu1 menyusunun Toplama sətrinin və AddAction obyektinin OnExecute hadisəsinin emaledicisindən ibarət altproqramlar olacaqdır.

```
procedure TForm1.oplama1Click(Sender: TObject);
begin
label1.Caption:=IntToStr(strtoint(edit1.Text)+strtoint(edit2.Text));
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
label1.Caption:=IntToStr(strtoint(edit1.Text)+strtoint(edit2.Text));
end;
procedure TForm1.AddActionExecute(Sender: TObject);
begin
 label1.Caption:=IntToStr(strtoint(edit1.Text)+strtoint(edit2.Text));
```

end;

Bundan sonra Obyektlər Inspektoru pəncərəsində Button1 düyməsinin və MainMenu1 menyusunun Torlama sətrinin Action xəssəsində AddAction vermək lazımdır.



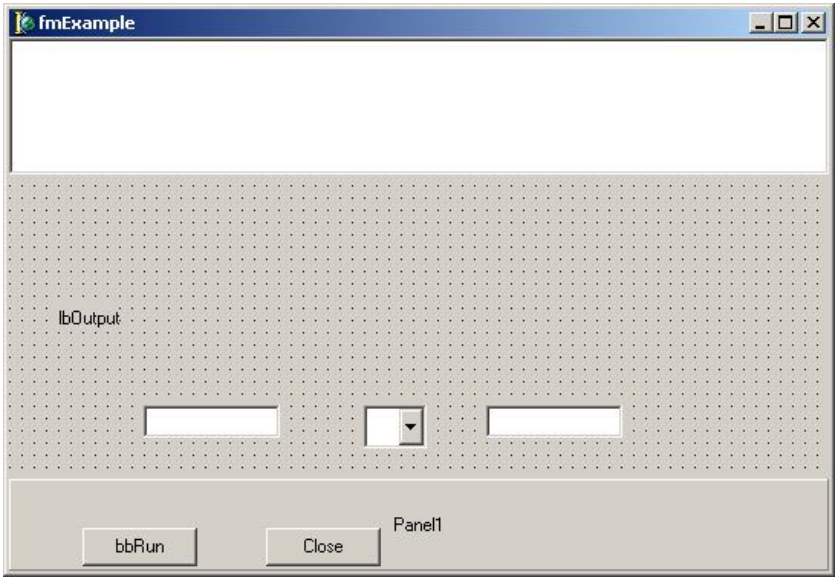
Şəkil 6.16Idarə elementinə əməliyyatın təyin olunması.

Bu işləri yerinə yetirdikdən sonra hər iki düymənin emaledicilərini proqram kodu pəncərəsindən götürmək olar. Bundan sonra proqramı yenidən kompilyasiya edib işə salmaq olar.

6.5.16 Kalkulyator yaradılması proqramı

Kalkulyator yaradılması proqramını verək.

Bunun üçün

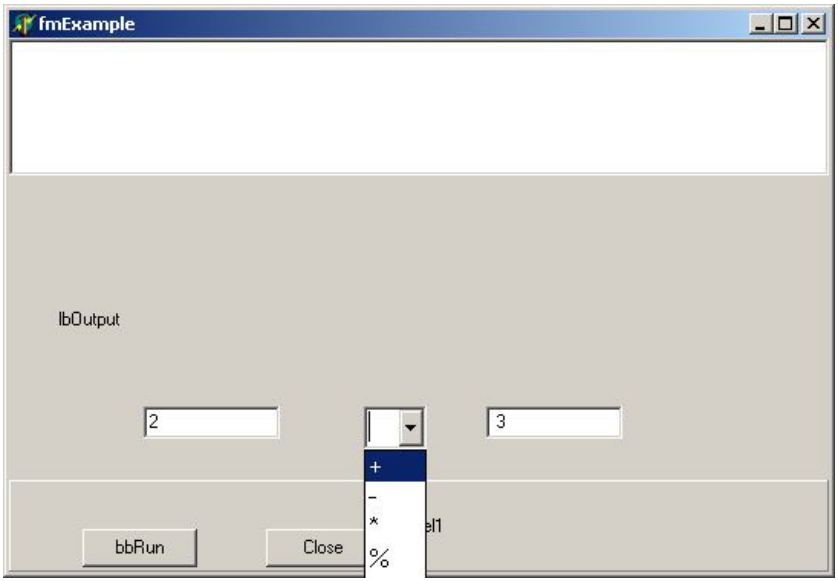


Şəkil 6.17 Kalkulyator proqramı üçün formanın görünüşü

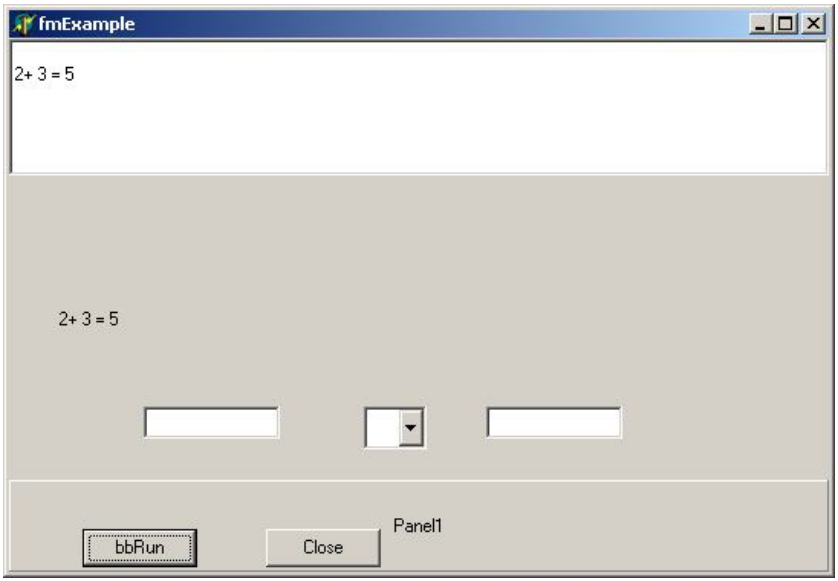
Proqram kodu

```
unit prog1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms,Dialogs, StdCtrls, ExtCtrls;

type
  TfmExample = class(TForm)
    mmOutput: TMemo;
    Panel1: TPanel;
    bbRun: TButton;
    Close: TButton;
    edInput1: TEdit;
    edInput2: TEdit;
    cbSign: TComboBox;
    lbOutput: TLabel;
    procedure bbRunClick(Sender: TObject);
  private
    { Private declarations }
  public
  { Public declarations }
  end;

var
```

```
  fmExample: TfmExample;

implementation

{$R *.dfm}

procedure TfmExample.bbRunClick(Sender: TObject);
var
x,y,z:real;
begin
if (edInput1.text='') or (edInput2.text='')or
(cbSign.ItemIndex<0)then exit;
try
x:=StrToFloat(trim(edInput1.Text));
except
ShowMessage('Sehv yazilish: '+edInput1.Text);
edInput1.SetFocus;
exit;
end;
try
y:=StrToFloat(trim(edInput2.Text));
except
ShowMessage('Sehv yazilish: '+edInput2.Text);
edInput2.SetFocus;
exit;
end;
case cbSign.ItemIndex of
0:z:=x+y;
1:z:=x-y;
2:z:=x*y;
3:z:=x*y/100;
4:try
z:=x/y;
except
z:=1.1e+38;
end;
end;
lbOutput.Caption:=trim(edInput1.Text)+''+
cbSign.items[cbSign.ItemIndex]+' '+ trim(edInput2.Text)+' = ';
if z>=1.1e38 then
lbOutput.Caption:=lbOutput.Caption+'sonsuzluq'
else
lbOutput.Caption:=lbOutput.Caption+floattostr(z);
mmOutput.Lines.Add(lbOutput.Caption);
edInput1.Text:='';
edInput2.Text:='';
cbSign.ItemIndex:=-1;


end;

end.
```



Şəkil 6.18 Proqram yerinə yetirildikdən və siyahılı sahə komponentindən əməl seçildikdən sonra pəncərənin görünüşü



Şəkil 6.19 Proqram yerinə yetirildikdən sonra pəncərənin görünüşü.

7. Object Pascal dili

7.1. Dilin elementləri

Əlifbası. Object Pascal dilinin əlifbası hərflərdən, rəqəmlərdən, onaltılıq rəqəmlərdən, xüsusi işarə, aralıqlar və ehtiyat sözlərdən təşkil olunmuşdur.

Hərflər-latın əlifbasının «a»-dan, «z»-ə və «A»-dan «Z»-ə qədər hərfləri və aşağı xət (-) işarəsindən ibarətdir. Dildə böyük və kiçik hərflər arasında (əgər onlar işarə və sətir ifadələrinə aid deyillərsə) heç bir fərq yoxdur.

Rəqəmlər-0-dan 9-a qədər ərəb rəqəmləri.

Hər bir onaltılıq rəqəm 0-dan 15-ə qədər qiymət alır. Birinci 10 rəqəm 0-9 ərəb rəqəmləri, qalan altısı isə A-F və ya a-f latın hərfləri ilə işarə olunur.

Object Pascal-ın xüsusi işarələri aşağıdakılardır:

+ - * / = , ‘ . ; : « » [ ], ( ) ,{ }, ^, @ , $, #

Xüsusi işarələrə həmçinin aşağıdakı işarələr cütü də aiddir:

< >, <=, >=, :=, (*,*), (. , .), //

Proqramda bu işarələr münasibət əməllərinin işarələri və ya kommentariya ayırıcıları kimi istifadə olunurlarsa, onlar arasında boşluq işarəsi qoymaq olmaz. (. və .) işarələrinin əvəzinə { və } işarələrindən istifadə etmək olar.

Dilin əlifbasında kodları 0-la 32 arasında yerləşən aralıqlar xüsusi yer tutur. Bu işarələrə identifikatorlar, sabitlər, ədədlər, ehtiyat sözlərinin ayırıcıları kimi baxmaq olar.

Object Pascal-da aşağıdakı ehtiyat sözlər vardır:

and	exports	mod	shr
array	file	nil	string
as	finalization	not	then
asm	finally	object	threadvar
begin	for	of	to
case	function	or	try
class	goto	out	type
const	if	packed	unit
constructor	implementation	procedure	until
destructor	in	program	uses
dispinterface	inherited	property	var
div	initialization	raise	while
do	inline	record	with
downto	interface	repeat	xor
else	is	resourcestring	
end	label	set	
except	library	shl	

Bunlardan əlavə *at* və *on* sözləri də xüsusi qiymətə malikdirlər(proqramın mətnində onların yazılma yerindən asılı olaraq).

Ehtiyat sözlərdən identifikator kimi istifadə etmək olmaz. Standart direktivlər proqramda müəyyən standart elanla bağlıdırlar. Onlara aşağıdakılar aiddir:

absolute	implements	read
Abctract	index	readonly
Assembler	message	register
Automated	name	reintroduce
Cdecl	near	requires
Contains	nodefault	resident
Default	override	safecall
Dispid	package	stdcall
Dynamic	pascal	stored
Export	private	virtual
External	protected	write
Far	public	writeonly
Forward	published	

Ehtiyat sözlər kimi standart direktivlər də Delphinin proqram kodu pəncərəsində yazılardan qalın şriftlə seçilirlər.

Private, protected, public, published və automed açar sözləri sinfin elanı daxilində ehtiyat sözləri, xaricində isə standart direktiv hesab olunurlar.

7.2. Identifikatorlar.

Object Pascalda identifikatorlar sabitlər, dəyişənlər, nişanlar, tiplər, obyektlər, siniflər, xassələr, prosedurlar, funksiyalar, modullar, proqram və yazı sahələrinin adlarından ibarətdir. Identifikator həmişə hərflə başlayır və sonra hərflər və rəqəmlər gələ bilər. Qeyd edək ki, aşağı xət (-) hərf sayıldığından identifikator onunla başlaya bilər və yalnız o işarələrdəndə ibarət ola bilər. Aralıq və xüsusi işarələr identifikatora daxil ola bilməz. Identifikatorlar istənilən uzunluqda ola bilər.

Identifikatorların düzgün yazılışına misallar:

```
A
ALPHA
Date_27_21
Sebuhi_6_MEKTEB
_Sebuhi
```

Identifikatorların səhv yazılışına misallar:

```
1Proqram      // rəqəmlə başlayır
Sebuhi$88     // xüsusi işarə var
My Prog       // aralıq işarəsi var
Do            // ehtiyat söz
```

7.3. Sabitlər

Object Pascalda sabit olaraq tam, həqiqi, onaltılıq ədədlər, məntiqi sabitlər, işarələr, işarələr sətri, çoxluqlar konstruktoru və NIL qeyri-müəyyən göstərici əlaməti götürülə bilər.

Tam ədədlər işarə və ya işarəsiz adi qaydada yazılır və –2 147 483 648-dən +2 147 483 647-ə qədər qiymət ala bilər. Qeyd etmək lazımdır ki, əgər tam sabit bu intervaldan kənara çıxırsa onda kompilyator avtomatik olaraq onu həqiqi tipə çevirir.

Həqiqi ədədlər işarəli və ya işarəsiz onluq nöqtədən istifadə etməklə və eksponensial formada verilir. Eksponensial hissə e və ya E işarələri ilə başlayıb, sonra «+» və ya «-» işarəsi və onluq tərtib gələ bilər. e(E) işarəsi onluq tərtibi nəzərdə tutur və mənası «vurulsun 10-cu qüvvətdən» deməkdir.

Misal.

O 5.51E3 - 5,51 vurulsun 10-cu qüvvətdən 3;

O- 88 E-2 - –88 vurulsun 10-cu qüvvətdən-2;

Əgər həqiqi ədədin yazılışında onluq nöqtə iştirak edirsə, onda ondan əvvəl heç olmasa bir rəqəm olmalıdır. Əgər ədəd eksponensial formada verilirsə, onda e(E) işarəsindən sonra heç olmasa bir onluq rəqəm verilməlidir.

Onaltılıq ədəd əvvəlində $(dollar) işarəsi yazılan onaltılıq rəqəmlərdən təşkil olunur. Onaltılıq ədədlərin dəyişmə diapazonu 00000000-dan FFFFFFFF-ə qədərdir. Məntiqi sabit ya False (yalan), ya da True (doğru) qiymətini alır.

Işarə sabiti – kompyuterin klaviaturasında olan apostrof işarəsinə alınmış istənilən işarədir:

O ‘r' - r işarəsi;

O 'f’- f işarəsi;

Əgər apostrof işarəsini yazmaq lazımdırsa, o iki dəfə yazılır, məsələn – (‘’) bu apostrof işarəsidir.

Işarəni onun kompüterdə daxili təsvirinin köməyi ilə əvvəlində # (diez) işarəsi olmaqla da vermək olar. Məsələn

O #97- a işarəsi;

O #90-Z işarəsi;
O #13-sətrin sonu işarəsi (CR)
Sətir tipli sabit – apostrof işarəsi arasında işarələrin istənilən ardıcıllığı (CR sətrin sonu simvolundan başqa).
Məsələn,
'Sebuhi'
Əgər sətirdə apostrof işarəsini göstərmək lazımdırsa, o iki dəfə yazılır:
'Me''lum'
Işarələr sətri boş da ola bilər, belə ki, apostrof işarələri arasında heç bir işarə olmaya da bilər. Sətiri hər bir işarənin kodunun qarşısında işarəsini qoymaqla da yazmaq olar. Məsələn, aşağıdakı hər iki sətir ekvivalentdir.
#83#121#109#98#11#108
'Symbol'
Nəhayət sətirdə bir hissəni apostrof içərisində, digər hissəni isə kodların köməyi ilə vermək olar. Bu yolla sətirə ixtiyari idarəediji işarə yerləşdirmək olar, o cümlədən CR işarəsi (kodu 13) məsələn
#7'Sehv'#13'Istenilən düyməni basın'#7
#7 kodu ilə təsvir olunan işarə klaviaturada yoxdur; Əgər kompüter səs kartı ilə təmin olunmayıbsa, bu işarə kompüterin dinamikini qısa səs siqnalı verməyə məcbur edir.
Çoxluqlar konstruktoru-kvadrat mötərizə arasına alınmış çoxluq elementlərinin siyahısıdır, məsələn;
[1,2, 4…7, 15]
[blue, red]

Object Pascal-da sabitlərin elanı zamanı operandları əvvəlcədən elan olunmuş sabitlərdən obyekt və tiplərin adlarından və onların aşağıdakı funksiyalarından ibarət olan ifadələrdən istifadə oluna bilər.

Abs	odd	sizeof
Chr	ord	succ
Hi	pred	swap
Length	ptr	trunc
Lo	round	

Məsələn,

```
Const
     MaxReal = MaxInt  div  SizeOf(real);
      NumChars = ord('z')-ord('a')+1;
       Ln10=2.302585092994;
Ln10R=1/ln10;
```

Dəyişənə başlanğıc qiymətin mənimsədilməsi
və statik dəyişən təsviri

Dəyişənə başlanğıc qiymətin mənimsədilməsi və bu dəyişənin qiymətinin prosedurun çalışması bitdikdən sonra saxlanması üçün Const kimi dəyişən təsvir olunur.

```
Const
Dəyişənin_adı:Tipi= Ilk_qiyməti;
```

Diqqətlə baxsaq görərik ki, sabitin təsvirindən fərqli olaraq dəyişənin adı ilə ilk qiyməti arasında dəyişənin tipi verilir.Bu təsvir heç də sabit təsviri deyildir.
Misal :Düyməsinin neçə dəfə basıldığını təyin edən proqram yazaq.

```
Procedure Tform1.Button1Click(Sender:Tobject);
 Var
   X:integer;
```

```
Begin
 x:=x+1;
 Label1.Caption:='bu duymeye'+IntToStr(x)+'defe basdiniz';
End;
```

Proqramı işlətsək hər düyməni sıxdıqda 1-ə bərabər olmayan eyni nəticəni görəcəyik. Çünki dəyişənə ilk qiymət mənimsənilməmişdir və x-in qiyməti təsadüfi bir ədəddir.Düyməyə hər dəfə sıxlıqda eyni nəticəni görməyimizin səbəbi isə dəyişənin lokal təyin olunması və buna görə də proqramın işi başa çatdıqdan sonra nəticənin yaddaşdan silinməsidir.

Dəyişənə ilk qiymət mənimsədilməsi və hər sonrakı addımda bu qiymətin yaddaşda saxlanması üçün proqramı aşağıdakı kimi yazmaq lazımdır.

```
{J+}//const bir deyisene qiymet menimsedilmesi ucun bu direktiv vacibdir
procedure Tform1.Button1Click(Sender:Tobject);
  Const
   X:integer=0;
  Begin
   X:=x+1;
   Label1.Caption:='bu duymeye'+IntToStr(x)+'defe basdiniz';
  End;
```

Artıq proqramımız düyməyə neçə dəfə basıldığını düzgün göstərəcəkdir.

Diqqətlə baxsaq görərik ki, ilk qiymətin mənimsədilməsi hər düymə sıxıldıqda təkrarlanmır.Belə olsa idi hər dəfə eyni nəticəni alardıq.Ilk qiymət bir dəfə mənimsədilir və prosedur hər dəfə işlədikdə dəyişənin bir öncəki qiymətindən istifadə olunur.

7.4. Ifadələr

Proqramın yerinə yetirilən hissəsinin əsas elementləri sabitlər, dəyişənlər və funksiyalardan ibarətdir. Bu elementlərdən hər biri öz qiyməti və müəyyən verilənlər tipinə aid olması ilə xarakterizə olunurlar. Mötərizələr,əməliyyat işarələri və bu elementlərin köməyi ilə ifadələr təşkil olunur ki, bu da faktiki olaraq yeni qiymətin alınmasından ibarətdir. Xüsusi halda ifadə yalnız bir elementdən, yəni sabit, dəyişən və ya funksiyaya müraciətdən ibarət ola bilər. Bu ifadənin tipi elementin tipi ilə üst -üstə düşməlidir. Ümumi halda isə ifadə bir neçə elementdən (operanddan) və əməl işarələrindən ibarət olduğundan, onun tipi operandların tipi və onlara tətbiq olunan əməllərdən asılıdır.

Ifadə yazılarkən ayrı-ayrı operand və əməl işarələri bir-birindən ixtiyari sayda aralıq işarələri ilə ayrıla bilərlər.

Ifadələrə aid nümunələr:

```
A
57
COS(t)
D<5
Not  K  and  (t=q).
```

7.5. Əməliyyatlar

Object Pascalda aşağıdakı əməliyyatlar təyin olunmuşdur.

O unar əməliyyatlar – **not , @**
O multiplikativ əməliyyatlar - ***, /, div, mod, and, shl, shr;**
O additiv əməliyyatlar - +,-,or,xor;
O münasibət əmələri - **=, <>, <, >, <=, >=, in**

Əməliyyatların yerinə yetirilmə üstünlüyü yuxarıda göstərilən qaydada azalır, belə ki, ən yüksək üstünlük dərəcəsinə unar əməliyyatlar (+,-, not, @) və ən aşağı üstünlük dərəcəsinə münasibət əməliyyatları malikdir. Bir neçə eyni üstünlük dərəcəsinə malik əməliyyatların soldan-sağa yerinə yetirilməsi məcburi deyil və bu ardıcıllıq kompilyatorun proqram kodunu optimallaşdırması şərtindən asılıdır. Məntiqi ifadələr hesablanarkən eyni üstünlük dərəcəsinə malik əməliyyatlar soldan sağa yerinə yetirilir. Project Options pəncərəsinin (Project ▶ Options əmrinin köməyi ilə açılır) Compiler bəndində Compiler Boolean eval bayraqcığı qoşulmuşdursa, bütün münasibət əmələri yerinə yetiriləcək, qoşulmamışdırsa, yalnız hesablamanın nəticəsini birqiymətli təyin etmək üçün zəruri olan əməliyyatlar yerinə yetiriləcək. Müxtəlif tip operandlar üzərində əməllərin yerinə yetirilməsi aşağıdakı cədvəldə verilmişdir:

Əməliyyat	Əməl	Operandın tipi	Nəticənin tipi
Not	Inkar	Məntiqi	Məntiqi
Not	Inkar	Ixtiyari tam	Operandın tipi
@	Ünvan	Ixtiyari	Göstərici
*	Vurma	Ixtiyari tam	Ən kiçik tam
*	Vurma	Ixtiyari həqiqi	Extended
*	Çoxluqların kəsişməsi	Çoxluq	Çoxluq
/	Bölmə	Ixtiyari həqiqi	Extended
Div	Tam bölmə	Ixtiyari tam	Ən kiçik tam
Mod	Qalığın tapılması	Ixtiyari tam	Ən kiçik tam
And	Məntiqi və	Məntiqi	Məntiqi
And	Məntiqi və	Ixtiyari tam	Ən kiçik tam
Shl	Sola sürüşmə	Ixtiyari tam	Ən kiçik tam
Shr	Sağa sürüşmə	Ixtiyari tam	Ən kiçik tam
+	Toplama	Ixtiyari tam	Ən kiçik tam
+	toplama	Ixtiyari həqiqi	Extended
+	Çoxluqların başı	Çoxluq	Çoxluq
+	Sətirlərin zəncirləri	Sətir	Sətir
-	Çıxma	Ixtiyari tam	Ən kiçik tam
-	Çıxma	Ixtiyari həqiqi	Extended
Or	Məntiqi və ya	Məntiqi	Məntiqi
Or	Məntiqi və ya	Ixtiyari tam	Ən kiçik tam
=	Bərabər	Ixtiyari sadə və ya sətir	Məntiqi
<>	Bərabər deyil	Ixtiyari sadə və ya sətir	Məntiqi
<	Kiçik	Məntiqi	Məntiqi
<=	Kiçik və ya bərabər	Məntiqi	Məntiqi
>	Böyük	Məntiqi	Məntiqi
>=	Böyük və ya bərabər	Məntiqi	Məntiqi

Object Pascal-da aşağıdakı məntiq əmələri təyin olunmuşdur.

O **not** məntiqi inkar
O **and** məntiqi Və
O **or** məntiqi və ya

O **xor** və ya-nı istisna edən

Məntiq əməlləri tam və məntiqi tip operandlara tətbiq oluna bilərlər. Əgər operandlar tam ədədlərdisə, onda məntiq əməllərinin nəticəsi də bitləri aşağıda göstərilmiş qaydada təşkil olunmuş tam ədəd olar.

Operand 1	Operand 2	Not	and	or	Xor
1	-	0	-	-	-
0	-	1	-	-	-
0	0	-	0	0	0
0	1	-	0	1	1
1	0	-	0	1	1
1	1	-	1	1	0

Tam ədədlər üzərində hər iki sürüşdürmə əməliyyatı da məntiq əməllərinə aiddir.

O **i shl j** -i-nin məzmunu j vahid sola sürüşdürülür; azad olmuş aşağı mərtəbələr sıfırlarla doldurulur.

O **i shr j** -i-nin məzmunu j vahid sağa sürüşdürülür; azad olmuş yuxarı mərtəbələr sıfırlarla doldurulur.

Məntiqi verilənlər üzərində məntiq əməllərinin nəticəsi aşağıda göstərilən qaydada məntiqi tip olur:

Operand1	Operand2	not	and	or	x or
True	-	false	-	-	-
False	-	true	-	-	-
False	False	-	false	false	False
False	True	-	false	true	True
True	False	-	false	true	True
True	True	-	True	true	False

In münasibət əməli iki operanda tətbiq olunur. Birinci operand istənilən tərtibli tipə aid olan ifadə, ikinci isə bu tip elementlərdən təşkil olunmuş çoxluq və ya çoxluq tipli identifikator ola bilər. Əgər sol operand çoxluğa aiddirsə, əməliyyatın nəticəsi **true** olacaqdır. Digər proqramlaşdırma dillərindən fərqli olaraq Object Pascalda məntiq əməllərinin üstünlük dərəcəsi münasibət əməllərindən yuxarı olduğundan şərt ifadələrinin formalaşdırılması zamanı mötərizələrdən istifadə etmək lazımdır.

(a>b) and (b<>0)

7.6. Operatorlar

Operator dedikdə, dildə xüsusi şəkildə təşkil olunmuş cümlə nəzərdə tutulur. Hər bir operator bir və ya bir neçə ardıcıl sətirdə yerləşə bilər və bir-birlərindən nöqtə vergüllə ayrılırlar.

7.6.1. Mənimsətmə operatoru

Mənimsətmə operatoru mənimsətmə işarəsi (:=) ilə birləşdirilən (iki hissədən ibarətdir). Mənimsətmə operatorunun işarələri (: və =) həmişə bitişik yazılır, onlar arasında boşluq işarəsi yazmaq olmaz. Mənimsətmə operatorunun sol tərəfində dəyişənin identifikatoru və ya xassə, sağ tərəfində isə həmin tip ifadə olmalıdır, nümunələr:

Var a, b, c, d:real;

```
        S:string;
        C:=5;
A:=c+3;
        B:=a/2;
        S:= 'Ahmedov Sebuhi';
        Label1.caption:=FloatToStr (b);
```

7.6.2. Mürəkkəb operator

Mürəkkəb operator - operator mötərizələri ilə (begin-başlanğıc, end-son) əhatə olunmuş Object Pascal-ın operatorlar qrupundan ibarətdir. Sintaksis nöqteyi nəzərindən mürəkkəb operatora neçə və hansı operatorları daxilinə almasından asılı olmayaraq bir operator kimi baxılır. Bu operatorların ixtiyarisi yenidən mürəkkəb operator ola bilər. Mürəkkəb operatorların bir-birinin daxilində yerləşməsinə heç bir məhdudiyyət qoyulmur.

```
Begin
…
Begin
….
Begin
….
End;
End;
End;
```

7.6.3. Məntiqi operator

Məntiqi operator proqramda budaqlanma nöqtələrini təyin edir. Bu operator **if**(əgər) **then** (onda) **else** (əks halda) ehtiyyat sözlərinin köməyi ilə iki formada yazılır:
If *şərti_ifadə* then *operator_1* else *operator_2*;
If *şərti _ifadə* then *operator*;

Operator yerinə yetirilərkən əvvəlcə *şərti_ifadə* hesablanır; əgər o, doğrudursa (True qiymətini alırsa) birinci operator yerinə yetirilir, əks halda (False qiymətini alırsa) birinci operator ötürülür və ikinci operator yerinə yetirilir. Ikinci formada *şərti_ifadə* doğru qiymət alırsa, then sözündən sonra gələn operator yerinə yetirilir, əks halda ötürülür.

Şərti opretorlarda olan operatorlar istənilən tip, o cümlədən şərti operator da ola bilər. Ikinci formadan gördüyümüz kimi şərti operatorda else hissəsi olmaya da bilər ki, bu da ümumilikdə şərti operatorun traktorkasında birqiymətliliyin pozulmasına gətirə bilər ki,bu çatışmazlığı aşağıdakı yolla aradan qaldırmaq olar: *operator_2*-nin **else** hissəsi yuxarıdan ona ən yaxın olan **then**-ə aid hesab olunur.

7.6.4. Dövrlər

7.6.4.1. For dövr operatoru

Operatorun formatı:
For *dövr_parametri*: =*ilkin_qiymət* To *son_qiymət* do *operator*;
Burada **for** (üçün) **to** (qədər) **do** (icra etmək) - ehtiyat sözlər;
Dövr_parametri - istənilən nizamlı tip lokal dəyişən; *ilikin_qiymət*,*son_qiymət* –həmin tip ifadə; *operator*-Object Pascalın istənilən operatoru. Operator aşağıdakı qaydada işləyir: əvvəlcə dövr parametrinə ilkin qiymət mənimsədilir, sonra son qiymət hesablanır. Əgər dövr parmetrinin qiyməti son qiymətdən kiçik və ya bərabərdirsə, operator yerinə yetirilir. Bundan sonra dövr parametrinin qiyməti bir vahid artırılır və yenidən *dövr_parametri* <=*son_qiymət*

şərti yoxlanılır. Dövr parametrinin qiyməti son qiyməti aşana qədər dövr davam edəcəkdir. **Do** sözündən sonra gələn operatorda *dövr_parametri*-ni və *son_qiymət*-ini dəyişmək olmaz.

Operatorun digər formatı:

For *dövr_parametri* :=*ilkin_qiymət* downto *son_qiymət operator*;

Bu formatda *dövr_parametri* ardıcıl olaraq *son_qiymət*-dən kiçik olana qədər bir vahid azaldılır.

7.6.4.2. While dövr operatoru

Operatorun formatı:

While *şərt* do *operator*;

Burada While (hələ ki), **do** (icra etmək)-ehtiyat sözlər; *şərt*-məntiqi tip ifadə; *operator*-object pascal-da istənilən operatordur (o cümlədən mürəkkəb operator).

Bu operator nə qədər ki, *şərt* true qiymət alır yerinə yetiriləcəkdir. Bu operatora çox vaxt ön şərtli dövr operatoru da deyilir.

7.6.4.3. Repeat dövr operatoru

Operatorun formatı:

Repeat
operatorlar;
Until *şərt*;

Burada **repeat** (təkrar etmək) və *until* (o vaxta qədər hələ ki) ehtiyat sözlər; *operatorlar*-Object Pascalın istənilən operatorları; *şərt*-məntiqi tip ifadədir. Bu dövr heç olmasa bir dəfə yerinə yetirildikdən sonra məntiqi ifadənin qiyməti hesablanır: əgər o **false** (yalan) olarsa, dövr təkrar olunur. Bu operatora son şərtli dövr operatoru da deyilir.

7.6.5. Continue və break proseduraları

Delphi-də parametrsiz Continue və break prosedurlarının köməyi ilə dövr operatorunun növbəti yerinə yetirilməsini təxirə salmaq və ya ondan vaxtından əvvəl çıxmaq olar: continue-dövrün gövdəsində bu proseduraya müraciətdən sonra gələn bütün operatorlar ötürülür və dövrün növbəti addımına keçilir;

Misal:1-dən 100-ə qədər cüt ədədləri Continue operatorundan istifadə etməklə toplamalı.

```
Var
    i,s:integer;
Begin
  S:=0;
 For i:=1 to 100 do
Begin
```

```
  If odd(i) then
    Continue;
S:=S+i;
End;
Show Message(IntToStr(s));
End;
```

break-bu prosedura dövrün yerinə yetirilməsini dayandırır və idarə dövr operatorundan sonra gələn operatora verilir.
Misal: ad[1000] adlardan ibarət sətir tipli massivi verilmişdir. for operatorunun köməyi ilə Sebuhi adı tapıldıqda Break operatoru ilə dövrdən çıxmağı təmin edən proqram yazaq:

```
Procedure Tform1.ButtonClick(Sender:TObject);
Var ad:array [1…1000] of string;
      I:integer;
Begin
  For i:=1 to 1000 do
    If ad[i]= 'Sebuhi' then
       Break;
    If i< 1000 then
      ShowMessage('Axtarılan ad'+IntToStr(i)+ 'saylıdır')
   Else
      Show Message ('Tapılmadı');
   End;
```

7.6.6. Exit və Halt prosedurları

Exit proseduru verilmiş blokdan çıxmağa imkan verir.Blok prosedurdan, dövr operatoru və ya şərt blokundan ibarət ola bilər.

Exit prosedurundan istifadə etməklə N!-ı hesablayan alt proqram yazaq

```
function factorial(n:integer): integer;
var i,r: integer;
begin
if n=0 then
begin
Result:=0;
Exit
end;
r:=1;
for i:=1 to n do
r:=r*i;
Result:=r
end;
```

Halt proseduru proqramdan çıxışı təmin edir.

7.6.7. Seçmə operatoru

Seçmə operatorunun formatı:

Case *seçmə_açarı* **of** *seçmə_siyahısı* (else operatorlar) **end;**
Burada case (variant), of(olan), else(əks halda), end(son)-açar sözləri; *seçmə_açarı*-tərtib tipli ifadə; *seçmə_siyahısı* bir və ya bir neçə aşağıdakı görünüşdən ibarət konstruksiyadır:
seçmə_sabiti: operator;
Seçmə_sabiti – seçmə_açarı tipinə uyğun sabit, operator isə Object Pascal-ın istənilən operatoru ola bilər.
Seçmə operatoru aşağıdakı qaydada işləyir:
Əvvəlcə *seçmə_açarı* ifadəsinin qiyməti hesablanır. Sonra isə seçmə sabitləri içərisində həmin qiymət ardıcıl olaraq axtarılır. Əgər həmin qiymət tapılarsa, onda onun qarşısındakı operator yerinə yetirilər. Bundan sonra seçmə operatoru öz işini qurtarır. Əgər seçmə sabitləri içərisində seçmə açarına uyğun qiymət tapılmazsa, onda idarə ELSE sözündən sonrakı operatora verilir.
Seçmə siyahısının istənilən operatorunun qarşısında bir deyil, bir-birindən vergüllə ayrılan bir neçə seçmə sabiti ola bilər. Nümunə:

```
Var
   C:char;
   S:string;
Begin
….
Case  c   of
‘y’, ‘Y’:  S:=’Yox’
‘h’, ‘H’: S:=’he’
 else
  S:= ‘Ne he,ne yox’ ;
End;
```

7.6.8. Keçid operatoru

Operatorun formatı:

goto ***nişan***;

Burada goto (keçməli) ehtiyat söz; ***nişan***-əvvəlcədən elan olunmuş nişandır. Nişan olaraq identifikatorlardan və işarəsiz tam ədədlərdən istifadə oluna bilər.
Bu operator heç bir şərt yoxlamadan, idarəni proqramın istənilən nişanlanmış sətrinə ötürür. Nişanlar əvvəlcədən proqramın **Label** bölməsində göstərilir. Nişan iki nöqtə (:) ilə operatordan ayrılır.
Yadda saxlamaq lazımdır ki, goto operatoru vasitəsi ilə modullara, prosedura və funksiyaya idarəni göndərmək, onlardan çıxmaq olmaz. Nümunə:

```
Label
      rr;
begin
   ….
   goto rr;     //keçid operatorunun yerinə yetirmə nəticəsində
   ….              //bu operatorlar ötürüləcək
   rr:continue;  //və idarə bu operatora ötürüləcək
   ….
End;
```

7.6.9. Birləşdirmə operatoru

Object Pascalda (siniflər, interfeyslər, yazılar və s.) hər hansı əlamətləri özündə birləşdirən mürəkkəb tiplərdən geniş istifadə olunur. Bu əlamətlərə (sahə, metod, xassə və s.) müraciət etmək üçün mürəkkəb obyektin adı göstərilən identifikator, ayırıcı nöqtə və əlamətin adı göstərilir. Nümunə:

```
MyDataModule. MyQuery.SQL. Clear;
MyDataModule.MyQuery.SQL.Add; (select*from books);
```

Bu halda **with** birləşdirmə operatorundan istifadə etmək əlverişlidir.

```
With MuDataModule. MyQuery.SQL do
Begin
Clear;
Add (select * from books);
End;
```

7.7. Alt proqrama müraciət

Object Pascal-da iki tip altproqram vardır: proseduralar və funksiyalar. Prosedura heç bir xüsusi qiymətə malik olmur, funksiya isə hər hansı tipə aid qiymətə malik olur. Altproqrama müraciət etmək üçün onun adı və («,») işarələri arasında müraciət parametrləri verilə bilər (altproqram parametrsiz də yazıla bilər). Nümunələr:

```
Var
Q: string;
 …
delete (S,1,1)
S:=FloatToStr(Cos (Pi/3));
```

Birinci operatorda delete prosedura s çağrılır və ona üç parametr verilir: sətir tipli dəyişən (s), dəyişəndə ləğv olunacaq. Birinci işarənin nömrəsi və ləğv olunacaq işarələrin sayı (prosedura s sətrində birinci işarəni ləğv edəcəkdir).

Ikinci operatorda isə ardıcıl olaraq üç funksiyaya müraciət olunur. Əvvəlcə pi funksiyası öz həqiqi qiymətini qaytarır. Pi=3,141592653…, sonra s funksiyası arqumentin Pi/3 qiymətində hesablanır, sonunda isə cos funksiyasının qaytardığı həqiqi qiymət FloatToStr funksiyasının köməyi ilə sətir tipə çevrilir.

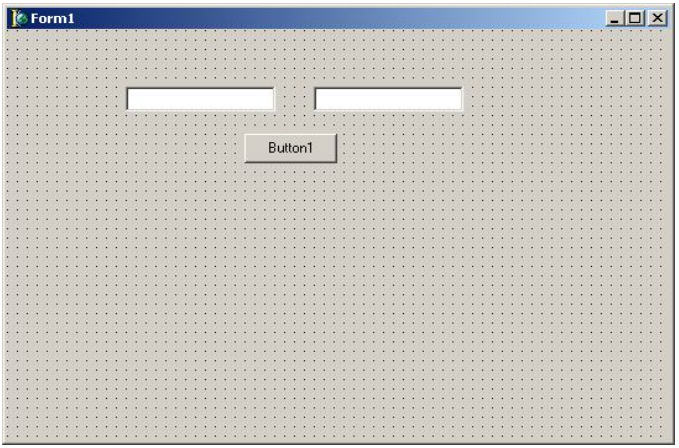
7.7.2 Prosedurlar

Prosedurlar ümumi şəkildə aşağıdakı kimi təsvir olunur.

```
Procedure prosedurun_adı(Giriş_parametrləri: tipi);
  Sabit,dəyişən,tiplərin təsviri
    Begin
      Proqram kodları;
    End
   [Exit;]
End
```

Misal:Prosedurdan istifadə edərək x tam tipli dəyişənin 5 mislini tapan proqram tərtib etməli.

Misalı həll etmək üçün File ▶New ▶Application əmri ilə yeni bir proyekt yaradaq.Forma üzərinə Edit1,Edit2 və Button1 komponentləri yerləşdirək.



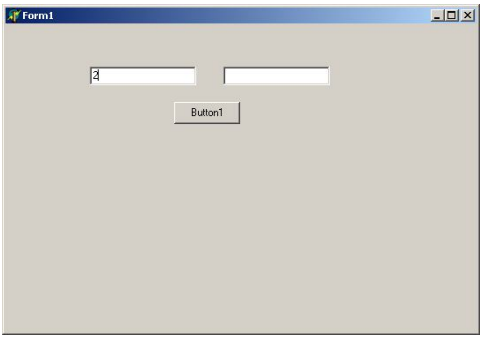
File ▶New ▶Unit əmri ilə proyektə yeni bir modul əlavə edək.Yeni yaratdığımız modula proqram kodunu aşağıdakı kimi yazaq.Yeni yaratdığımız modulu proyektə qoşmaq üçün standart alətlər panelində  yerləşən düyməsindən istifadə edirik.

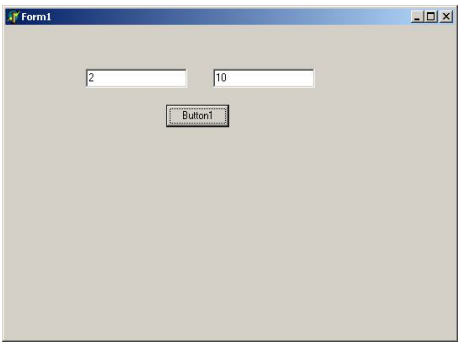
```
unit Unit2;
interface
  procedure unitt2(var y:integer);
implementation
uses Unit1;
 procedure unitt2( var y:integer);
 begin
 y:=y*5;
 end ;
end.
```

Button1 düyməsinin vurulması hadisəsinin emaledicisinə aşağıdakı prqram kodunu yazırıq.

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms,Dialogs, StdCtrls;
type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Button1: TButton;
    Edit2: TEdit;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
```

```
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
uses Unit2;
{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
var x :integer;
begin
x:=strtoint(edit1.Text);
unitt2(x);
edit2.text:=inttostr(x);
end;

end.
```





7.7.3.Funksiyalar

Prosedurlar tələb olunan işi görər və geriyə heç bir qiymət qaytarmaz.Funksiya isə tələb olunan işi görər və geriyə qiymət qaytarar.

Funksiya aşağıdakı şəkildə təsvir olunar.

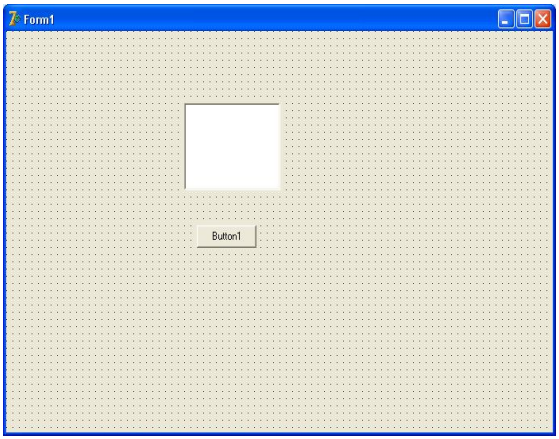
Function Funksiyanın_adı(Giriş_parametrləri: tipi):Funksiyanın_tipi;
Sabit ,dəyişən, tiplərin təsviri
Begin
Proqram kodları;
[Exit;]
Funksiyanın_adı:=Qiymət;
End;

Prosedurun təsbirindən fərqli olaraq funksiya geriyə qiymət qaytardığı üçün bu qiymətin tipi Funksiyanın_tipi parametri ilə verilir.

Geriyə qaytarılan qiymət ya Funksiyanın_adı ya da Rezult parametrinə verilir.

Function Funksiyanın_adı(Giriş_parametrləri: tipi):Funksiyanın_tipi;
Sabit ,dəyişən, tiplərin təsviri
Begin
Proqram kodları;
[Exit;]
Rezult:=Qiymət;
End;

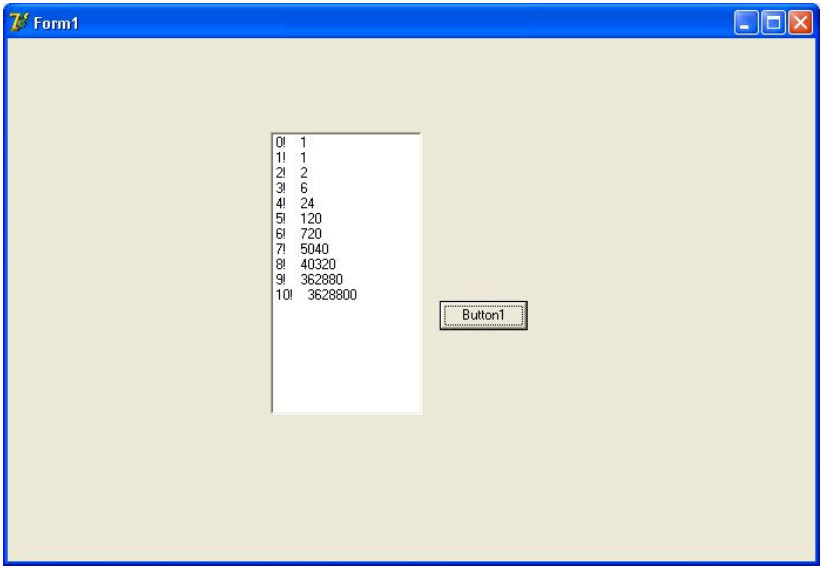
Misal Funksiyanın köməyi ilə N!-ı hesablayan program yazaq.




```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;
```

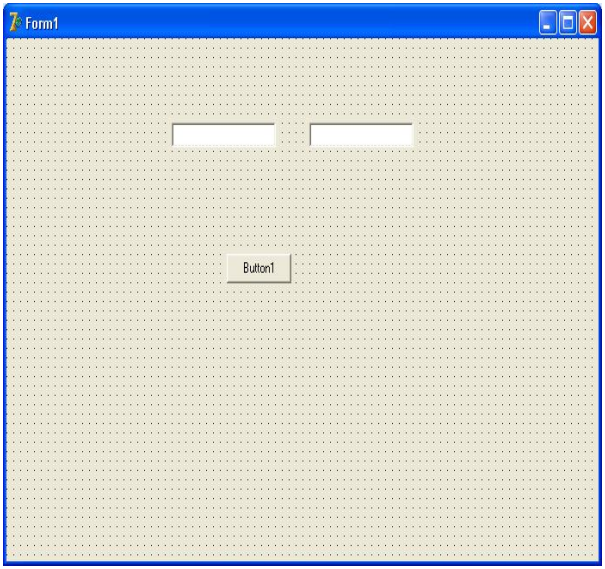
```
type
 TForm1 = class(TForm)
   Button1: TButton;
   ListBox1: TListBox;
   procedure Button1Click(Sender: TObject);
 private
   { Private declarations }
 public
   { Public declarations }
 end;
var
 Form1: TForm1;
implementation
{$R *.dfm}
Function Fak(x:integer):LongInt;
var
 netice:LongInt;
 i:integer;
begin
 if x<0 then
   begin
     Fak:=-1;
     Exit;
   end;
   netice:=1;
   if x=0 then
    netice:=1
   else
    for i:=1 to x do
    netice:=netice*i;
 Fak:=netice;
end;
procedure TForm1.Button1Click(Sender: TObject);
var
n,i:integer;
begin
n:=strtoint(InputBox('Eded daxil etme','Bir eded daxil et',''));
ListBox1.Clear;
For i:=0 to n do
ListBox1.Items.Add(IntToStr(i)+'!    '+IntToStr(Fak(i)));
```

```
end;
end.
```



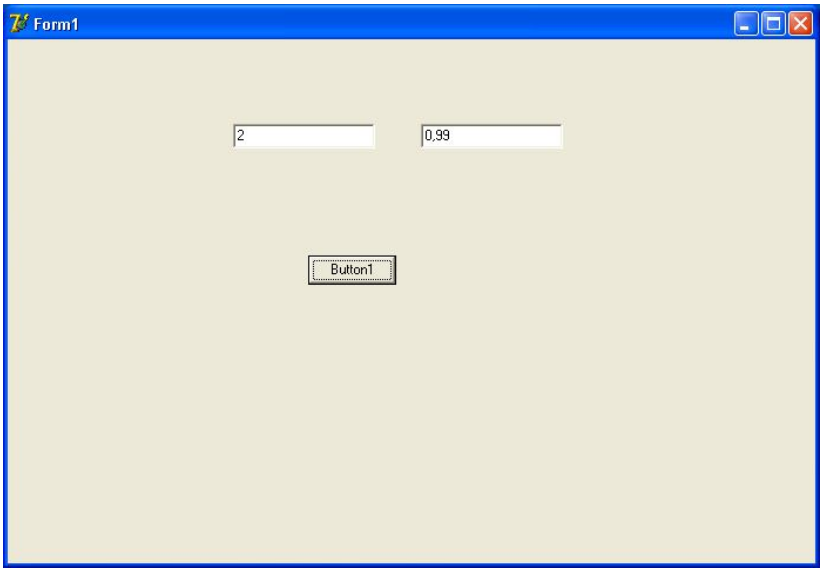
Funksiya geriyə yalnız bir qiymət qaytarır.Bəs iki və daha çox qiymət qaytarmaq tələb olunursa nə olacaqdır Bunun üçün daxil edilən parametrlərin müəyyən bir qiyməti qaytarmasına nail olmaqdan ötrü onlar təsvir olunarkən onların önünə Var sözü əlavə olunmalıdır.Təbii ki bu halda funksiyadan istifadə məcburi deyil bir prosedurun köməyi ilə də bu işi görmək mümkündür.

Misal Həqiqi ədədin tam və ohluq hissələrini ayıran bir prosedur yazaq.




```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;
type
  TForm1 = class(TForm)
    Button1: TButton;
    Edit1: TEdit;
    Edit2: TEdit;
   procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.dfm}
```

```
procedure kesrtam(s:real;var t:longint;var v:real);
            begin
            t:=Trunc(s);
            v:=Frac(s);
            end;
procedureTForm1.Button1Click(Sender: TObject);
           var
            x:Longint;
            y:real;
           begin
           kesrtam(2.99,x,y);
           edit1.text:=IntToStr(x);
           edit2.text:=FloatToStr(y);
           end;

           end.
```



7.7.4. Prosedur və Funksiya təsviri

Bəzi hallarda bir prosedur və ya funksiyanı çağırmamışdan öncə onun varlığını kompilyatora bildirmək lazımdır.Bir prosedur bə ya funksiya təsvir olunmamışdan əvvəl çağrılırsa kompilyator Undeclared İdentifier səhvini verəcəkdir.

Fərz edək ki, x və y adlı iki prosedurumuz vardır.Onların aşağıdakı sıra ilə yazıldıqlarını qəbul edək:

```
Procedure x;
  Begin
    ...
  End
Procedure y;
  Begin
    ...
  End;
```

Bu vəziyyətdə y prosedurundan x proseduru çağrıla bilər.

```
Procedure x;
  Begin
    ...
  End
Procedure y;
  Begin
    x; //y prosedurundan x proseduru çağrılır
    ...
  End;
```

Ancaq x prosedurundan y proseduru çağrıla bilməz.Çünki x proseduru y prosedurundan əvvəl təsvir olunmuşdur və bu prosedurun varlığı kompilyatora bildirilməlidir.

```
Procedure x;
  Begin
  y; //x prosedurundan y proseduru çağrılır
    ...
  End
Procedure y;
  Begin


    ...
  End;
```

Bu vəziyyət də əvvəlcədən y proseduru təsvir olunmalıdır.Ancaq y proseduru da x prosedurunu çağırırsa, hansını əvvələ keçirsək ,nəticə eyni olacaqdır.

```
Procedure x;
  Begin
  y; //x prosedurundan y proseduru çağrılır
    ...
  End
Procedure y;
  Begin
    x; //y prosedurundan x proseduru çağrılır
    ...
  End;
```

Bu vəziyyətdən çıxmaq üçün x posedurunda y proseduru çağrılmamışdan əvvəl y prosedurunun varlığını forward sözü ilə kompilyatora bildirmək lazımdır.

```
Procedure y; forward; //y prosedurunun varlığı bildirildi
```

```
Procedure x;
 Begin
  y;
   ...
 End
Procedure y;
  Begin
    x;
    ...
  End;
```

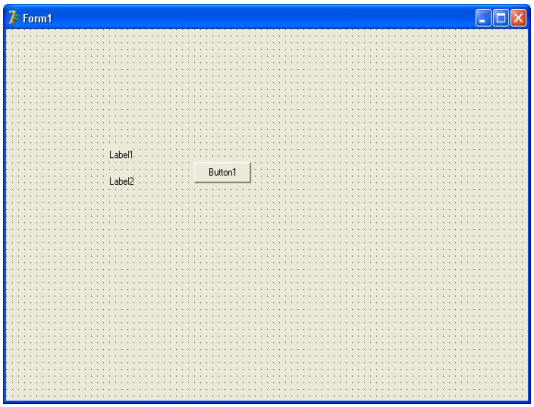
7.7.5 Eyni adlı birdən çox funksiyadan istifadə etmə

Delphi eyni adlı birdən artıq prosedurdan istifadə etməyə imkan verir.Bunun üçün prosedurun təsvirinin sonuna **overload** sözünü yazmaq lazımdır.

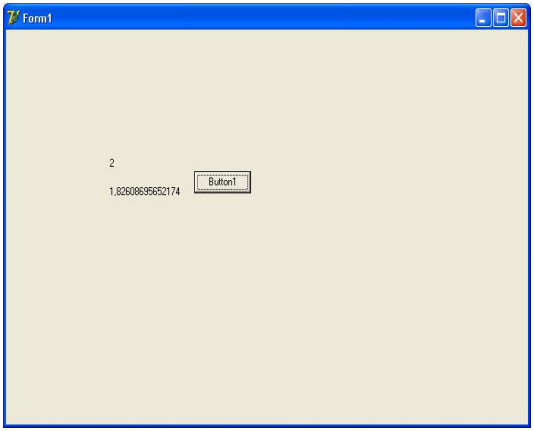
Buna nədən ehtiyac olduğu ilə maraqlana bilərsiniz.Bunun iki səbəbi vardır.

1.Bir funksiyaya ayrı-ayrı tiplərdə verilənlər daxil edilə bilər.

```
Function bol(x,y:real):real;overload;
 Begin
   Result:=x/y;
 End;
Function bol(x,y:integer):integer;overload;
 Begin
   Result:=x div y;
 End;
```




```
procedure TForm1.Button1Click(Sender: TObject);
begin
Label1.Caption:=inttostr(bol(4,2));
Label2.Caption:=floattostr(bol(4.2,2.3));
end;
```



2.Bəzən də prosedurda daxil edilən parametrlərin sayı müxtəlif olur.Istifadəçinin istədiyi parametri istədiyi qədər yaza bilməsi üçün **yaz** adlı bir prosedur hazırlayaq.Prosedurumuz aşağıdakı şəkildə olacaqdır.

```
Procedure yaz(metn:string;eded:integer);overload;
Var
 I:integer;
 Begin
   For i:=1 to   eded  do
     Form1.Canvas.TextOut(0,i*20,metn);
   End;
```

Əgər istifadəçi eded parametrini yazmırsa,bir dəfə yazacaqdır.Bu halda parametrin olmadığı yenibir prosedur yazılmalıdır.

```
   Procedure yaz(metn:string);overload;
Var
 I:integer;
 Begin
     Form1.Canvas.TextOut(0,0,metn);
   End;
```

Hər iki prosedurun adı eyni yaz olmasına baxmayaraq hansı prosedurun işləməsi prosedurun necə çağrılmasından asılıdır.

Əgər **yaz** prosedurunu iki parametr ilə çağırsanız ilk prosedur çağrılır.Tək parametr ilə çağrılarsa,ikinci prosedur işləyəcəkdir.

```
Yaz('Sebuhi',5)//Birinci prosedur  işləyərək  5 dəfə  yazar
Yaz('Qazax');//  Ikinci prosedur işləyərək 1 dəfə  yazar
```

Parametri sabit qiymətli prosedur təsviri

Prosedur parametrlərinə sabit qiymətlər mənimsədilərək,həmin qiymətlərlə prosedur işləyə bilər.Yuxarıdakı **yaz** adlı prosedurun köməyi ilə istifadəçi parametri istədiyi sayda yazdıra bilir.

```
Procedure yaz(metn:string;eded:integer);
```

Burada istifadəçi metn parametrinə yazdıracağı mətni,eded parametri ilə isə neçə dəfə yazacağını verir.Istifadəçi ikinci parametri vermirsə,prosedurun mətni bir dəfə yazacağını qəbul edək.Bu halda proseduru aşağıdakı şəkildə təsvir etmək lazımdır.

```
Procedure yaz(metn:string;eded:integer=1);
```

Əgər istifadəçi yaz prosedurunu parametrsiz çağırarsa,eded parametri 1 olaraq qəbul ediləcəkdir.

```
Procedure yaz(metn:string;eded:integer=1);
    Var
     I:integer;
     Begin
       For i:=1 to    eded  do
         Form1.Canvas.TextOut(0,i*20,metn);
       End;
```

Istifadəçi proseduru eded parametri olmadan çağırarsa,onun qiyməti 1 qəbul ediləcəkdir.

```
  yaz('Rafael',5);//eded parametri 5-dir
  yaz('Rafael');//eded parametri verilmədiyi üçün 1 qəbul edilir
```

Bu vəziyyət overload **olaraq təsvir olunmuş prosedur ilə qarışdırılmamalıdır.** overload **olaraq təsvir olunmuş prosedurlarda eyni prosedurdan iki ədəd təsvir edirdik.Parametrin verilib verilməməsindən asılı olaraq bu və ya digər prosedur işləyirdi.Burada isə yalnız bir təsvirdən söhbət gedir və parametr verilmirsə ,onun sabit qiymətindən istifadə olunacaqdır.**

7.7.6. Local və qlobal dəyişənlər

Əgər dəyişəni bir prosedurun Var blokunda təsvir etsək, o dəyişənə yalnız həmin prosedur müraciət edə bilir və buna lokal dəyişən deyilir.

```
Procedure A;
Var A1: integer;
Begin
…..
end;
Procedure B;
Var B1: integer
Begin
……
end;
```

Yuxarıdakı proqram fraqmentində və A proceduru B1 dəyişəninə, nə də B proceduru A1 dəyişəninə müraciət edə bilmir. Çünki onlar procedura daxilində lokal dəyişənlərdir. Hətta bu dəyişənlərin adları eyni olsa belə onlara müraciət mümkün deyil.

```
Procedure A;
Var A1:integer;
Begin
……
end;
Procedure B;
Var A1:integer
Begin
……
end;
```

Yuxarıdakı proqram fraqmentində hər iki dəyişənin adı A1 olsa da, onlar bir-birindən asılı deyillər.

Iki procedurun eyni dəyişənə müraciəti. Hər iki procedurun eyni dəyişənə müraciət edə bilməsi üçün onları procedurların Var blokunda deyil Unit-in Var blokunda təsvir etmək lazımdır.

Proqramın kod pəncərəsində aşağıdakı kimi Var bloku görərik:

```
Var
  Form 1:Tform;
Impemention
{$R*.DFM}
…….
```

Dəyişəni buradakı Var blokunda təsvir etsək, Unit içindəki procedurlar bu dəyişənə müraciət edə bilər. Yəni dəyişən qlobal olur.

```
Var
  Form 1: TForm;
  K:integer;
  implementation
    {$R*.DFM}
  Procedure A;
     Var A1:integer
  Begin
  ……..
  end;
  procedure B;
    Var B1: integer;
  Begin
   ……..
  end;
```

Yuxarıdakı kimi K dəyişəni UNIT –in var blokunda təsvir olunursa, həm A, həm də B proceduru bu dəyişənə müraciət edə bilir. Əgər procedurların biri bu dəyişənin qiymətini dəyişirsə, digər procedur dəyişilmiş qiymətdən istifadə edir.

Iki UNIT-in eyni dəyişənə muraciəti

Bu dəyişəni UNIT-in Var blokunda təsvir etsək, o dəyişənə həm bu UNIT həm də proqramdakı digər UNIT-lər tərəfindən müraciət oluna bilər. Digər UNIT-lərdə müraciətin mümkünlüyü üçün USES sətrində UNIT adı yazılmalıdır.

```
// UNIT 1
Var
   Form1: Tform1;
    K:integer; //UNIT1 daxilində tanınan dəyişən
Implementation
```

```
   {$R*.DFM}
Procedure A;
  Var A1:integer;
```

```
Begin
……
end;
Procedure B;
Var B1:integer;
Begin
……
end;
```

UNIT2-də K-dəyişəninə müraciət edə bilmək üçün aşağıdakı kimi yazılmalıdır:

```
//Unit2
implementation
USES UNIT1;
{$R*.DFM}
Procedure C;
   Var C1: integer;
Begin
K:=…//UNIT1-dəki dəyişən tanınır
End;
```

Iki UNIT-in eyni procedura müraciəti. Bir dəyişənə başqa bir UNIT içəridən müraciət etdiyimiz kimi, bir UNIT içərisində tanınmış procedura da digər UNIT-dən müraciət etmək olar.

Yuxarıdakı misalda yazılmış A və B procedurlarına baxaq. B procedurunun UNIT2-də tanınması üçün aşağıdakı kimi yazılmalıdır.

```
//UNIT1
Var
   Form1:TForm;
   K:integer;
Procedure B;
Implementation
{$R*.DFM}
Procedure A;
  Var A1:integer
Begin
…….
End;
Procedure B;
Var B1:integer;
Begin
……
end;
```

Bu cür təsvirdən sonra B proseduruna UNIT2-də müraciət oluna bilməsi üçün UNIT1-i UNIT2-nin USES sətrində yazmaq lazımdır.

```
// UNIT2
implementation
USES UNIT1;
 {$R*.DFM}
Procedure C;
   Var C1:integer;
Begin
B(); //UNIT1-dəki B procedurunu çağırır
```

```
End;
```

UNIT2-də bu adda bir procedur varsa, UNIT1-dəki proceduru çağırarkən, procedurun adının əvvəlinə UNIT-in adını yazmaq lazımdır.

```
//UNIT2
implementation
USES UNIT1;
{$R*.DFM}
Procedure B;
Begin
…..
end;
Procedure C;
Var C1:integer;
Begin
B( ); //bu UNIT-dəki b proceduru
UNIT1.b( ); //UNIT1-dəki b proceduru
End;
```

Altproqramda lokal dəyişən əvvəlcədən elan olunmuş qlobal dəyişənlə üst-üstə düşə bilər. Bu halda lokal dəyişən qlobal dəyişənin üstünü örtərək ona müraciət etməyə imkan vermir. Məsələn

```
Procedure TForm1.button1Click(Sender:TObject);
Var
        i:Integer;
Procedure P;
var
i:Intecer;
begin
        label1.caption:=Int ToStr(I);
        end;//P
        begin
                i:=1
                P
                end
```

Bu proqramın ekrana nə çıxaracağım araşdıraq. P prosedurunda i dəyişəninin qiyməti təyin olunmamışdır, baxmayaraq ki, eyni adlı i qlobal dəyişəni 1-ə bərabərdir. Local dəyişən qlobal dəyişəni «gizlədir» və nəticədə ekrana ixtiyari qiymət çıxarılır.Əgər P prosedurundan aşağıdakı təsviri götürsün, ekrana i dəyişəninin qiyməti olaraq 1 çıxarılacaqdır.

```
                var
                        i:integer;
```

Beləliklə, eyni adlı local və qlobal dəyişənlər tamamilə müxtəlif dəyişənlərdir.

7.8. Proqramın kodlaşdırılması qaydaları

Proqram kodunun əsas hissəsi alt proqramlarda yerləşdiyindən onların tərtibinə aid qaydalarla tanış olaq. Istənilən altproqram başlıqdan, təsvir bölməsindən və operatorlar bölməsindən ibarət olur.

```
Procedure MyProcedure; //Başlıq
```

```
// təsvirlər bölməsi
begin
   //operatorlar bölməsi;
end;
```

Təsvir bölməsində dəyişənlər (təsvir olunur), sabitlər, altproqramlar və operatorlar bölməsində istifadə olunan nişanlar təsvir olunur.

Təsvirlər (seksiyalarda) **var** (dəyişənlər seksiyası), **const** (sabitlər seksiyası), **label**(nişanlar seksiyası) ehtiyat sözləri ilə başlayan seksiyalarda qruplaşdırılır. Seksiyaların yazılma ardıcıllığı və onların miqdarı heç bir əhəmiyyət kəsb etmir.

Dəyişənlərin təsviri onların tipinin, sabitlərin təsviri onların qiymətlərinin göstərilməsindən, nişanların təsviri onların adlarının label seksiyasında sadalanmasından ibarətdir.

```
Var
  K:Integer;
  A,b:Real;
  List1, List2: TString List;
Const
      S='Object Pascal';
      Q='Ahmedov Sebuhi';
Label
     Aa,21,t3;
```

Bir sətirdə ixtiyari sayda təsvir yerləşdirmək olar. Eyni tipli dəyişənləri siyahi şəklində qruplaşdırmaq olar.

Qlobal dəyişənləri (altproqramdan kənarda təsvir olunan dəyişənlər) onların təsviri zamanı tipdən sonra bərabər (=) işarəsi qoymaqla inisiallaşdırmaq olar:

```
Var
   Q:string =’Ahmedov Sebuhi';
```

Lokal (altproqramda təsvir olunan dəyişənlər) dəyişənləri bu yolla inisiallaşdırmaq olmaz.

Proqramda tipli sabitlərdən istifadə etməyə icazə verilir.

```
      Const
          Q:String='Ahmedov Sebuhi';
```

Tipsiz sibitlərdən fərqli olaraq, tipli sabitləri proqramın yerinə yetirilməsi zamanı dəyişdirmək olar. Tipli sabitlərin qiymətini Delphi-nin 6-cı versiyasından başqa bütün versiyalarında dəyişdirmək olar.

Operatorlar bir və ya bir neçə sətirdə yerləşir və (;) işarəsi ilə qurtarır. Operatorlar keçid operatoru müstəsna olmaqla yazıldıqları ardıcıllıqla yerinə yetirilirlər. Bir sətirdə bir neçə operator yerləşə bilər. Bu zaman onlar soldan sağa yerinə yetirilir. Object Pascalda komentariyalar vermək üçün {…}, (*…*) və // işarələrindən istifadə olunur.

```
{bu komentariyadır}
(* bu da komentariyadır*)
// cari sətrin bütün işarələri komentariyadır.
(**) və {} işarələri kəsişə bilməz, lakin bir-birinin daxilində yerləşə bilərlər.
(*…{ …*)…} // belə yazmaq olmaz
(*…{ …}…*)// belə yazmaq olar
```

Əgər komentariya {$ və ya (*$ işarəsi ilə başlayırsa, belə komentariya kompilyatorun direktivi hesab olunur. Bunun köməyi ilə kompilyatorun sazlanmasını dinamik olaraq dəyişdirmək və şərti kompilyasiyanı həyata keçirmək olar.

7.9. Tiplər

Object Pascal ciddi tipləşdirilmiş alqoritmik dildir. Bu o deməkdir ki, dildə susma ilə təyin olunan tip yoxdur. Verilənlərin tipi uyğun dəyişənin tutduğu yaddaş sahəsinin uzunluğunu və dəyişənlər üzərində mümkün əməliyyatları təyin edir.

Object Pascalda **Integer, Boolean, Real** əvvəlcədən təyin olunmuş tiplərdir. Bununla bərabər dildə istifadəçi tipləri-massivlər, yazılar, siniflər yaradılması üçün vasitələr vardır. Istifadəçi tipləri kompilyatora əvvəlcədən tanış deyil və onlar yalnız **type** seksiyasında və modulun interfeys hissəsində təsvir olunduqdan sonra təyin olunurlar. Dildə istifadəçi tiplərini dəyişənlərin təsviri bölməsində təsvir etməyə icazə verilir.

```
Var
MyArray: array [1..5] of Integer;
        Q: record
              Ad:string;
               BirthDay: TDateTime;
             End;
```

Fərz edək ki, iki massiv elan olunmuşdur.

```
        Var
          A:array [1..5] of Byte;
          B:array [1..5] of Byte;
```

Aşağıdakı operatorun translyasiyası zamanı kompilyator dəyişənlərin tiplərinin uyğun gəlmədiyini göstərəcək.

```
A:=B;
```

Massivləri aşağıdakı kimi elan etmək lazımdır.

```
Type
    MyArray:array [1..5] of Byte;
Var
     A:MyArray;
     B:MyArray;
```

Və ya:

```
Var
  A,B:array [1..5] of Byte;
```

7.9.1. Nizamlı tiplər

Nizamlı tiplər sonlu sayda mümkün qiymətlər alması ilə fərqlənirlər. Bu qiymətləri müəyyən qaydada nizamlamaq olar və deməli, qiymətlərin hər birinə qarşı hər hansı tam ədəd qoymaq olar(onların sıra nömrəsi olan).

Nizamlı tiplərə tam, məntiqi, işarə, sadalanan və diapazon tip aiddir. Onların hər birinə X ifadəsinin sıra nömrəsini qaytaran Ord(x) funksiyası tətbiq oluna bilər. Tam tiplər üçün Ord(x)=x. Ord(x) funksiyasının məntiqi, işarə və sadalanan tiplərə tətbiqi 0 və 2 diapazonunda (məntiqi tip), 0 və 255 diapazonunda (işarə tip), 0 və 65535 diapazonunda (sadalanan tip) müsbət tam ədəd verir. Diapazon tipinə bu funksiyanın tətbiqi bu tipin xassələrindən asılıdır.

Nizamlı tipə aşağıdakı funksiyaları tətbiq etmək olar.

O Pred(x) - Ord(x)-1 sıra nömrəsinə uyğun gələn, nizamlı tipin özündən əvvəl gələn qiymətini qaytarır.

Ord(Pred(x))=Ord(x)-1

O Succ(x) - Ord(x)+1 sıra nömrəsinə uyğun gələn, nizamlı tipin özündən sonra gələn qiymətini qaytarır.

Ord(Succ(x))=Ord(x)+1

Məsələn, fərz edək ki, proqramda aşağıdakı dəyişən təyin olunmuşdur.

```
Var
   C:Char;
Begin
  C:= '7';
End;
```

Onda Pred(c) funksiyası '6' işarəsini, Succ(x) funksiyası isə '8' işarəsini qaytaracaqdır.

7.9.2. Tam tip

Tam tiplərin mümkün qiymətləri diapazonu onların daxili təsvirlərindən, yəni bir, iki, dörd və ya səkkiz bayt yer tutmasından asılıdır. Aşağıdakı cədvəldə tam tiplərin adları, onların daxili təsvirləri baytlarla və mümkün qiymətləri diapazonu verilmişdir.

Tipin adı	Uzunluğu, bayt	Qiymətlər diapazonu
Byte	1	0…255
Short Int	1	-128…+127
Small Int	2	-32768…+32767
Word	2	0…65535
Integer	4	-2 147 483 648…+2147 483 647
Long Int	4	-2 147 483 648…+2147 483 647
Long Word	4	0…4 294 967 295
Int 64	8	-2^{63}… + 2^{63}-1
Cardinal	4	0… 4 294 967 295

Prosedura və funksiyalarda tam tipli parametrlərdən istifadə edərkən tiplərin bir birinin «daxilinə alması» nəzərə almaq lazımdır. Belə ki, Word tip istifadə olunan yerdə Byte tipdən istifadə etmək olar (əksinə olmaz).

Tam tipli parametrlərə tətbiq oluna bilən prosedura və funksiyaların siyasi aşağıdakı cədvəldə verilmişdir. b, s, w, i, l hərfləri ilə Byte, ShortInt, Word, Integer və LongInt tiplərinə aid olan ifadələr işarə olunmuşdur.

X-bu tiplərdən birinə aid ifadədir; vb, vs, vw, vl, vx hərfləri ilə uyğun tiplərə aid dəyişənlər işarə olunmuşdur.

Altproqram	Nəticənin tipi	Təsviri
abs(x)	x	x-in modulunu qaytarır
Chr(b)	char	koda görə işarəni gaytarır
Dec(vx(,1))	-	vx-in qiymətini i vahid azaldır, i parametri olmadıqda 1 vahid azaldır.
Inc(vx(,i))	-	vx-in qiymətini i vahid artırır, i parametri olmadıqda 1 vahid artırır.
Hi(w)	Byte	Arqumentin yuxarı baytını qaytarır
Hi(I)	Byte	Arqumentin yuxarı baytını qaytarır
Lo(I)	Byte	Arqumentin aşağı baytını qaytarır
Lo(W)	Byte	Arqumentin aşağı baytını qaytarır
Odd(l)	Boolean	Arqument təkdirsə, True qiymətini qaytarır
Random(W)	Parametrin tipi	0-(w-1) arasında müntəzəm paylanmış təsadüfi ədədi qaytarır.
Sqr(x)		Arqumentin kvadratını qaytarır
Swap(i)	Integer	Sözdə baytların yerini dəyişir
Swap(w)	Word	Sözdə baytların yerini dəyişir

Tam ədədlər üzərində əməllər apararkən nəticənin tipi operandların tiplərinə uyğun gəlir, əgər operandlar müxtəlif tam tipə aiddirsə, hər iki operandın tipinin daxil olduğu ümumi tipə aid olacaq. Məsələn, Short Int və Vord tipli operandların nəticəsi Integer tipə aid olacaq.

Burada ən vacib şərt ən kiçik yaddaş tutumlu dəyişən tipini seçməklə lazımi məqsədə çatmaqdır. Məsələn, insanın yaşından istifadə olunan proqramda Byte tipli dəyişən

Yaddaşda 1bayt yer tutduğundan bu iş üçün ideal dəyişən tip olacaqdır. Burada bir faktı da nəzərə admaq lazımdır ki, LongInt,Integer və Cardinal tam tipli dəyişənlərdə dəyişən təyin olunmuş sərhəddi aşdıqda istifadəçiyə xəta mesajı göndərilir və proqramın işi dayandırılır. Digər 4 tam tipli dəyişənlərdə isə məsələ tam başqa çür qoyulur.Belə ki,Shortint dəyişəni –128 ilə+127 arasında dəyişir əgər bu tipli dəyişənə +128(+129) qiyməti mənimsədilərsə,onda xəta mesajı əvəzinə bu tip dəyişənə –128(-127) ədədi qiyməti mənimsədilir.SmallInt tam tipli dəyişəninə +32768(+32769) ədədi qiyməti mənimsədilərsəbxəta mesajı əvəzinə bu dəyişənə –32768(-32767)qiyməti mənimsədilir.Yaxud bu tip dəyişənə –32769(-32770)ədədi qiyməti mənimsədilərsə,+32767(+32766)ədədi qiyməti geriyə qaytarılır.Byte tipli dəyişən də (0 ilə 255 arası)dəyişənə 256(257)ədədi qiyməti mənimsədilərsə,geriyə 0(1) ədədi qiyməti qaytarılır.Yaxud bu tipdəyişənə-1(-2)ədədi qiyməti mənimsədilirsə,geriyə 65535(65534)qiyməti qaytarılacaqdır.

Word tipli dəyişəndə dəyişənə 65536(65537)ədədi qiyməti mənimsədilirsə,geriyə 0(1) qiyməti qaytarılacaqdır.Yaxud bu tip dəyişənə -1(-2) ədədi qiyməti mənimsədilirsə,geriyə 255(254) qiyməti qaytarılacaqdır.

Buradan bu nəticəyə gəlmək olar ki,bu dörd tam tipli dəyişənlərin öz limit qiymətlərini aşması qiymətlərin dairəvi olaraq təkrarlanması ilə müşahidə olunur.

. -1 .

. .

-2 0

. .

. .

-128 1

. .

. .
127 2
. .
. .
126

7.9.3. Məntiqi tip

Məntiqi tipə Boolean, ByteBool, Bool, WordBool və LongBool aiddir. Standart Pascalda ancaq Boolean tipi təyin olunmuşdur. Qalan tiplər Object Pascalda Windows ilə həmrəyliyi təmin etmək üçün daxil edilmişlər. Boolean və ByteBool tipləri hər biri bayt, Bool və Word Bool iki bayt, LongBool dörd bayt yer tutur. Məntiqi tipin qiymətləri əvvəlcədən elan olunmuş sabit qiymətlərdən True (doğru) və ya False (yalan) ola bilər. Onlar üçün aşağıdakı qayda doğrudur:

```
Ord(False)=0
Ord(True)<> 0
Succ(False)=True
Pred(True)=False
```

Məntiqi tip nizamlı tipə aid olduğundan ondan for dövr operatorunda istifadə etmək olar.

```
Var
  l:Boolean;
begin
  for l:=False to True do…
```

Bir faktı yadda saxlamaq lazımdır ki, Delphi-nin 32 mərtəbəli versiyası üçün Boolean tip üçün Ord(True)=+1, digər tiplər üçün isə Ord(True)=-1. Ona görə də bu tip operatorlardan istifadə etdikdə bir qədər ehtiyatlı olmaq lazımdır.

Məsələn, Delphi 6 versiyasında ShowMessage ('---') operatoru aşağıdakı for dövr operatorunda bir dəfə də olsun yerinə yetirilməyəcək.

```
Var
   L:Bool;
   K:Integer;
Begin
   For L:=False to True do
        Show Message ('---');
End;
```

Əgər dövr parametrinin tipinin dəyişib Boolean etsək, dövr işləyəcək və məlumat iki dəfə ekrana çıxacaq.

7.9.4. Işarə tipi

Işarə tipinin qiymetleri komputerin klaviaturasinin butun ishareleri choxlugu ola biler. Her bir ishareye 0-255 diapazonunda tam eded uygun gelir. Bu eded isharenin daxili tesvir koduna uygun gelib, ord funksiyasinin komeyi ile qiymeti qaytarilir.

Windows-da kodlashdirma uchun ANSI (bu kodu teklif eden amerika standartlashdirma institutunun adina uygun-American National Standard Institute) kodundan istifade olunur. Klaviaturanin kodlari 0-la 127 arasinda yerleshen isharelerinin siyahisi ashagidaki cedvelde verilmishdir.

ANSI kodlashdirilmasi

kod	Ishare	Kod	Ishare	kod	ishare	kod	Ishare
0	NUL	32	BL	64	@	96	*
1	SOH	33	!	65	A	97	A
2	STX	34	“	66	B	98	B
3	ETX	35	#	67	C	99	C
4	EQT	36	$	68	D	100	D
5	ENQ	37	%	69	E	101	E
6	ACK	38	&	70	F	102	F
7	BEL	39	‘	71	G	103	G
8	BS	40	(	72	H	104	H
9	HT	41	)	73	I	105	I
10	LF	42	*	74	J	106	J
11	VT	43	+	75	R	107	K
12	FF	44	,	76	L	108	L
13	CR	45	-	77	M	109	M
14	SO	46	.	78	N	110	N
15	SI	47	/	79	O	111	O
16	DEL	48	0	80	P	112	P
17	DC1	49	1	81	Q	113	Q
18	DC2	50	2	82	R	114	R
19	DC3	51	3	83	S	115	S
20	DC4	52	4	84	T	116	T
21	NAR	53	5	85	V	117	U
22	SYN	54	6	86	V	118	V
23	ETB	55	7	87	W	119	W
24	CAN	56	8	88	X	120	X
25	EM	57	9	89	Y	121	Y
26	SUB	58	:	90	Z	122	Z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	/	124	/
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	-	127	

Işarə tipli dəyişənlər üzərində aşağıdakı funksiyalar təyin olunub

Ord(x) - x işarəsinin ANSI kodunu verir
Chr(k) - kodu k olan ANSI işarəsini verir
UpCase(CH) - Char tipli funksiya; əgər CH kiçik latın hərfdirsə, onu böyük hərfə çevirir, əks halda CH işarəsinin özünü qaytarır.
Misal:ord(‘B')=66; chr(66)= 'B';
Işarə tipli c arqumenti üçün Pred(c) və Succ(c) funksiyaları təyin olunur.

Bu funksiyalar ANSI cədvəlində (c) işarəsindən müvafiq olaraq əvvəl və sonra gələn işarəni təyin edir.

Işarə tipli c_1,c_2 dəyişənləri üzərində <,<=,>,>=,<>,= müqayisə aparmaq mümkündür:

Məsələn: 'A'< 'B'

Kodları 128-255 arasında dəyişən işarələr müxtəlif şriftlər üçün dəyişir.

AnsiChar tipi

Char tipi ilə eynidir.

WideChar tipi

Bu tip 2 baytlıq işarə tipidir və uzaq şərq xalqlarının dillərində istifadə olunur.

Pchar tipi

Sıfr terminallı sətir adlandırılan bu tip sətir tipidir.Bu tip əsasən daha çox Windows API–lərində istifadə olunur.Bu tipdəki sətirlərin sonunda #0 işarəsi qoyulur.Bu işarə sətrin bitdiyi yeri göstərir. Digər sətir tipləri ilə PChar tipi arasında əməliyyatlar aparmaq olar.

```
Procedure Tform1.FormCreate(Sender:Tobject);
 Var
  Ad:string;
  Soyad:Pchar;
  Begin
    Ad:=’Sebuhi’;
    Soyad:=’Ahmedov’;
     ShowMessage(ad+soyad);
  End;
```

Əgər String və ya AnsiString tipində təyin olunmuş dəyişəni Pchar tipində bir dəyişənə mənimsətmək istəyiriksə, Pchar tip dəyişməsindən istifadə etmək olar. MessageBox Windows API-sindəki parametrlər Pchar tipindədir.Bu parametrlərə sətir tip olaraq elan olunmuş dəyişənləri aşağıdakı kimi mənimsətmək lazımdır.

```
Var
  Str1,str2:string;
  Begin
    MessageBox(0,Pchar(str1),Pchar(str2),mb_ok);
  End
```

Pchar tipini sətir tipinə çevirmək üçün String(p) tip çevirməsindən istifadə etmək olar.

7.9.5. Sadalanan tip

Sadalanan tip onun ala biləcəyi qiymətlərin sadalanması yolu ilə verilir. Hər bir qiymət hər hansı identifikatorla işarə olunur və dairəvi mötərisə ilə əhatə olunmuş siyahıda yerləşdirilir. Məsələn,

```
Type
  Colors=(red, white, blue);
```

Sadalanan tipin tətbiqi proqramı əyaniləşdirir. Məsələn, əgər, proqramda ilin aylarından verilən kimi istifadə olunursa, onda aşağıdakı proqram fraqmenti proqramı daha da əyaniləşdirir:

```
Type
    Ay_Tip=(Yan, Fev, Mar, Apr, May, Iyun, Iyul, Avq, Sen, Okt, Noy,
Dek);
```

```
Var
  Ay:Ay_Tip;
Begin
…..
 if Ay:=Iyun then
lboutput.Caption:= ‘Nabrana getmek vaxtıdır';
….
End;
```

Hər bir sadalanan tipin elementləri daxili nömrələri ilə təşkil edilir. Nömrələnmə 0-dan başlayır. Sadalanan tipə daxil olan elementlərin sayı 65 536-dən çox ola bilməz.

Sadalanan tipdən istifadə olunması uyğun dəyişənin ala biləcəyi qiymətlərə nəzarətin mümkünlüyü nöqteyi-nəzərdən proqramın etibarlığını artırır. Məsələn, fərz edək ki, aşağıdakı sadalanan tiplər verilmişdir.

```
Type
   Colors=(black, red, blue);
Ordinal=(one, two, three);
Days=(Monday, Tuesday, Wednesday);
```

Elementlərinin sayı və daxili təsvir nöqteyi nəzərindən hər üç tip ekvivalentdir:

```
    Ord(black)=0….ord(blue)=2
    Ord(One)=0….ord(three)=2
    Ord(Monday)=0…ord(Wednesday)=2
```

Lakin fərqlər də vardır. Aşağıdakı dəyişənləri təyin edək:

```
     Var
       Col:colors;
        Num:ordinal;
         Day:days;
```

Bu halda aşağıdakı operatorlar yol veriləndir:

```
       Col:=black;
        Num:=Succ(two);
         Day:=pred(Tuesday);
```

Lakin aşağıdakı operatorlar yol verilməzdir:

```
         Col:=one;
          Day:=black;
```

Yuxarıda qeyd etdiyimiz kimi sadalanan tipin elementləri ilə tam ədədlər çoxluğu arasında Ord(x) funksiyası ilə təyin olunan qarşılıqlı birqiymətli uyğunluq vardır.

Object Pascalda tərsinə çevirmə də mümkündür:

Word tipli ifadənin qiymətini sadalanan tipə çevirmək olar, əgər tam ifadənin qiyməti sadalanan tipin elementlərinin maksimal sayını aşmırsa.

Məsələn, yuxarıda elan olunan tiplər üçün aşağıdakı mənimsətmə operatorları ekvivalentdir:

```
Col:=black;
Col:=colors[0];
```

7.9.6. Diapazon tipi

Diapazon tipi özünün baza tipinin altçoxluğu olub, baza tipi olaraq diapazon tipindən başqa istənilən nizamlı tip götürülə bilər. Diapazon tipi baza tip daxilində özünün aşağı və yuxarı sərhədlərinin verilməsi ilə təyin olunur:

Min_qiymət..max_qiymət

Misal: type
Digit= '0'..'9';
Dig1='27'..'57';
Diapazon tipi təyin edərkən aşağıdakı qaydaları əsas götürmək lazımdır.
O Iki nöqtə (..) işarəsinə bir işarə kimi baxılır, ona görədə onlar arasında boşluq işarəsi qoymaq olmaz;
O Dipazonun sol sərhəddi sağ sərhəddini aşmamalıdır.

Diapazon tipi elementlərinin sayı nəzərə alınmaqla özünün baza tipinin bütün xassələrinə malikdir. Xüsusi halda dəyişəni aşağıdakı şəkildə təyin edək:

```
Type
   Days=(mo, tu, we, th, fr, sa, su);
WeekEnd=Sa…Su;
Var
W:WeekEnd;
 Begin
 …..
 w:=sa;
 …..
 end;
```

Ord(w) funksiyası 5 qiymətini qaytaracaq, Pred(w) funksiyası isə səhvə gətirəcək.
Object Pascal-ın standart kitabxanasında diapazon tiplə işləmək üçün aşağıdakı funksiyalar nəzərdə tutulur:
O High(x) - x dəyişəninin daxil olduğu diapazon tipin maksimal qiymətini qaytarır;
O Low(x) - diapazon tipin minimal qiymətini qaytarır.

7.9.7. Real tiplər

Single - 4 baytlıq onluq həqiqi tip olub, $1,5x10^{-45}$ilə $3,4x10^{38}$ aralığında qiymətlər ala bilər. Vergüldən sonra 7-8 rəqəmi qiymətli hesab olunur.

Real48 - 6 baytlıq onluq həqiqi tip olub, $2,9x10^{-39}$ ilə $1,7x10^{38}$ aralığında qiymətlər ala bilər. Vergüldən sonra 11-12 rəqəmi qiymətli hesab olunur.

Real və Double-8 baytlıq onluq həqiqi tiplər olub, $5,0x10^{-324}$ ilə, $1,7x10^{308}$ aralığında qiymətlər ala bilər.

Vergüldən sonra 15-16 rəqəmi qiymətli hesab olunur.

Delphi 4 versiyası ilə birlikdə 48 bitlik hər tipi 64 bitə çatdırılmış və əvvəlki tip Real 48 adlandırılmışdır. Real və Double tipi arasında heç bir fərq yoxdur.

```
Var
        r1:real; //64 bitlik həqiqi tip
        r2:real48;// 48 bitlik həqiqi tip
        d:double;//r1 ilə eyni tipdəndir
```

Extended - 10 baytlıq onluq həqiqi tipdir. $3,4x10^{-4932}$ ilə $1.1x10^{4932}$ arasında qiymətlər ala bilər. Vergüldən sonra 19-20 rəqəm qiymətli hesab olunur.

Comp - 8 baytlıq onluq həqiqi tipdir. $-2^{63}+1$ ilə $2^{63}-1$ aralığında qiymətlər ala bilər. Comp tipi 2-nin qüvvətlərini ala bilən həqiqi tipdir.

Currency - 8 baytlıq onluq həqiqi tip olub, -922337203685477.5808 ilə +922337203685477.5807 aralığında qiymətlər ala bilər.

7.9.8. Sətir tipi

Object Pascalda məbulərin emalı üçün aşağıdakı tiplərdən istifadə olunur:
- qısa sətir Short String və ya String (N), harada ki, N 255;
- uzun sətir String;

- O geniş sətir Vide String;
- O sıfr terminallı sətir;

Bu tiplərin hamısı üçün ümumi bir xüsusiyyət vardır ki, hər bir sətirə işarələrinin sayı aşağıdakı qaydada dəyişən bir ölçülü işarə massivi kimi baxıla bilər: String (N) üçün sətrin uzunluğu 0-la N arasında, String və PChar üçün 0-la 2qbayt arasında dəyişəbilər. Standart Pascal dilində ancaq qısa String [N] sətrindən istifadə olunur. Yaddaşda belə sətir üçün (N+1) bayt ayrılır, birinci bayta sətrin cari uzunlugu, işarələr özləri isə 2-ci baytdan başlayaraq yerləşdirilirlər. Sətrin uzunluğu üçün bir bayt ayrıldığından, qısa sətrin maksimal uzunluğu 255-i aşa bilməz. Maksimal uzunluqlu qısa sətri təsvir etmək üçün ShortString (String[255] ekvivalentdir) tipi təyin olunmuşdur. Windovs-da #0 işarəsi ilə məhdudlanmış işarələr ardıcıllığından ibarət sıfr terminallı sətirlərdən geniş istifadə olunur.

Delphi-nin 32 mərtəbəli versiyalarında Activex texnoloqiyasına əsaslanan komponentlərə uyğunluğu təmin etmək üçün WideString standart tipi təyin olunmuşdur. Bu tipin String tipindən fərqi hər bir işarə üçün yaddaşda bir deyil, iki bayt ayrılmasıdır.

Sətir tiplərin təsvirinə aid misallar:

```
Var
SsS:string [25];// uzunluğu 250 işarəyə qədər olan qısa tip
SsMax:ShortString// uzunluğu 255 işarəyə qədər olan qısa tip
StS:string //uzun sətir
SvS:WideString;//geniş sətir
PcS:Pchar;//sıfır terminallı sətirə istinad
AcS:array[0..1000] of Char; //uzunluğu 1000 işarəyə qədər 0 terminli sətir ssS dəyişəninin elanı zamanı kompilyator onun yerləşməsi üçün 250+1=251 bayt yer ayırır və birinci 0-cı baytda sətrin cari uzunluğunu yerləşdirir. Aşağıdakı proqram fraqmentinə baxaq:
    Procedure fmExample.bb Run Click(Sender:TObject);
    Var
      SsS:String[25];
    Begin
      SsS:= 'Ahmedov Sebuhi';
      SsS[5]:= 't'; //işarələr sətirdə 1-dən nömrələnirlər
      LbOutput.Caption:=ssS;/ 'Ahmetov Sebuhi' alınacaq
    End;
```

Bu fraqment yerinə yetirilərkən ssS dəyişəninə Ahmedov Sebuhi işarələr sətri mənimsədilir, birinci bayt 14 qiyməti (işarələrin sayı) alacaqdır.

Ikinci operator yerinə yetirildikdən sonra 5 indeksli işarə (baytların indeksasiyası 0-dan başlayır, birinci baytda cari uzunluq yerləşdiyindən, sətirdə birinci işarənin indeksi 1 olacaq) t işarəsi ilə əvəz olunacaq.

StS uzun sətirli dəyişən üçün yaddaşla işləmə mexanizmi tamamilə fərqlənəcək:

Aşağıdakı proqram fraqmentinə baxaq:

```
Procedure TfmExample.bbRunClick(Sender:TObject);
Var
  StS, stSS: String;
Begin
    StS:= 'Ahmedov Sebuhi';
    StSS:=stS;
    StS:= 'Elaci-'+stS;
    StS(10):= 't'; //Işarələr 1-dən başlayaraq nömrələnir
    LbOutput.Caption:=stS;//Elaci-Ahmetov Sebuhi çıxacaq
  End;
```

7.9.9. Strukturlaşdırılmış Tiplər

Strukturlaşdırılmış tiplər (Object Pascal-da onlar dörddür: massivlər, yazılar, çoxluqlar və fayllar) bu tipləri təşkil edən elementlər çoxluğu ilə xarakterizə olunurlar. Öz növbəsində hər bir element strukturlaşdırılmış tipə məxsus ola bilər ki, bu da tiplərin bir-birini daxilinə alması barədə danışmağa imkan verir. Object Pascal-da tiplərin bir-birinin daxilinə olması dərinliyinə heç bir məhdudiyyət qoyulmur, lakin onların ixtiyarının daxili təsvirinin ümumi ölçüsü 2 bayt-aşmamalıdır.

Strukturlaşdırılmış tipi təsvir etməmişdən əvvəl packed ehtiyat sözü yazıla bilər ki, bu da kompilyatora strukturlaşdırılmış tipin obyektlərinə ayrılan yaddaş sahəsinə qənaət etməyə imkan verir.

7.9.9.1. Massivlər

Massivlər elementlərinin sayına və strukturuna görə nizamlanmış və nömrələnmiş bircinsli verilənlər yığımıdır.

Massivin tipini ümumi şəkildə aşağıdakı kimi təsvir etmək olar.

Tipin_adı=array[indekslər_tipinin/tiplər_indeksinin_siyahısı] of tip;

burada ***tipin_adı***-düzgün identifikator; **array, of**-ehtiyat sözlər; ***tiplər_indeksinin_siyahısı***-bir-birindən vergül ilə ayrılan bir və ya bir neçə indeks tiplərinin siyahısı.

Əhatə edən mötərizələr sintaksisin tələbi, ***tip***-Object Pascal-ın ixtiyari tipi ola bilər. ***Indexlər_tipi*** olaraq 2 baytdan yuxarı tutuma malik olmayan istənilən nizamlı tip götürülə bilər (LongWord və Int64 tiplərindən başqa). Dəyişəni massiv olaraq bu dəyişənin bilavasitə təsviri zamanı da vermək olar.

Məsələn

```
var
  a,b:array[1..10] of real;
```

Bir qayda olaraq *indexlər_tipi* olaraq indexslərin dəyişmə sərhədləri verilməklə məhdud tipdən istifadə olunur. Massivin tipinin ümumi yazılışında *of* ehtiyat sözündən sonra gələn tip Object Pascal-ın istənilən tipi, o cümlədən massiv ola bilər:

Məsələn:

```
type
  matris=array[0..5] of array [-2..2] of array [char] of byte;
```

Bu yazılışı daha kompakt

```
type
  matris=array [0..5,-2..2, char] of byte;
```

yazılışı ilə əvəz etmək olar. Strukturlaşdırılmış tiplərin bir-birinin daxilində olmasına heç bir məhdudiyyət qoyulmadığından massivin ölçüsünə də heç bir məhdudiyyət qoyulmur.

Kompüterin yaddaşında massivin elementləri bir-birinin ardınca elə yerləşir ki,kiçik ünvandan böyük ünvana keçdikdə birinci olaraq ən sağda olan indeks dəyişir. Fərz edək ki, massiv aşağıdakı qaydada verilmişdir:

```
Var
  a:packed array [1..2,1..2] of Byte;
  begin
  a[1,1]:=1;
  a[2,1]:=2;
  a[1,2]:=3;
  a[2,2]:=4;
  end;
```

Onda yaddaşda bir-birinin arxasınca 1,3,2,4 qiymətli baytlar yerləşəcək.

Object Pascal-da bir mənimsətmə operatorunun köməyi ilə bir massivin elementlərini digər eyni tipli massivin elementlərinə mənimsətmək olar.

Məsələn

```
var
  a,b: array [1..10] of Byte;
begin

a:=b

end;
```

Bu proqram fraqmenti yerinə yetirildikdən sonra A massivinin bütün elementləri B massivinin uyğun elementləri aldığı qiymətləri alacaqdır.

Massivlər üzərində münasibət (müqayisə) əməlləri təyin olunmamışdır.

Məsələn

```
if a=b then
```

yazmaq olmaz.

Iki massivin elementlərini müqayisə etmək olar.

Məsələn

```
var
  a,b:array[1..5] of byte;
eq:boolean;
  i:byte;
begin
  ….
eq:=true;
  for i:=1 to 5 do
   if a[i]<>b[i] then
        eq:=false;
   if eq then
    …..
   end;
```

Bir proqram içərisində eyni xassəyə malik birdən artıq dəyişən təsvir etdikdə onları tək-tək deyil massiv şəklində təsvir etmək sərfəlidir.Bir qrupda olan tələbələrin yaş və boylarının orta qiymətini verən proqram yazaq.Bir qrupda 30 nəfər tələbə varsa,hər tələbəyə aid dəyişəni təkrar-təkrar təsvir etmək əvəzinə onları bir massiv şəklində təsvir etmək əlverişlidir.

Əyani olaraq massivi VAR təsvir blokunda aşağıdakı kimi göstərmək olar.

telebe:Array[1..30] of integer; Burada telebe integer tipində bir dəyişən olub,1-dən 30-a qədər qiymət almaqdadır.

Buna görə də

```
telebe1:integer;
telebe2:integer;
telebe3:integer;
  …….
telebe30:integer;
```

və ya

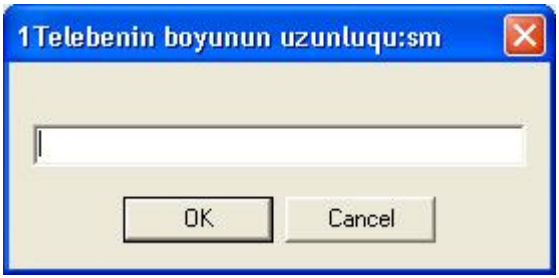
telebe1, telebe2, telebe3,…, telebe30:integer;şəklində 30 dəyişən təsvir etmək əvəzinə, massivin köməyi ilə bir sətir ilə 30 ayrı-ayrı dəyişənlər təsvir olunur.

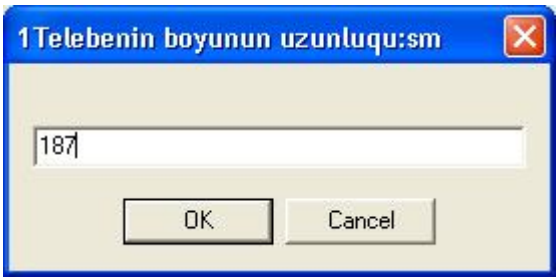
Yeni bir proyekt yaradaraq ,forma üzərinə Button1,Button2 düymələri yerləşdirərək,onların OnClick hadisəsinin emaledicisinə aşağıdakı proqram kodlarını yazırıq.

```
procedure TForm1.Button1Click(Sender: TObject);
var
```

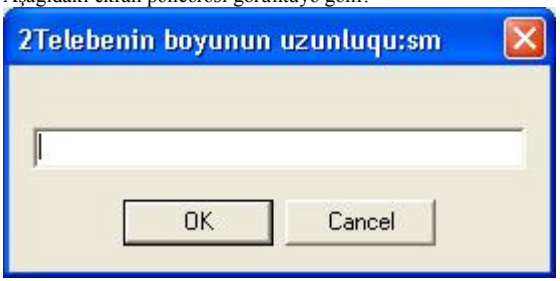
```
  telebe:array[1..10] of integer;
  boycem,tel:integer;
  ort:string[5];
begin
boycem:=0;
 for tel:=1 to 10 do
 begin
 telebe[tel]:=StrToInt(InputBox(IntToStr(tel)+
 'Telebenin boyunun uzunluqu:sm','',''));
 boycem:=boycem+telebe[tel];
 end;
 ort:=FloatToStr(boycem/10);
ShowMessage('Qrupun telebelerin orta boyu:'+ort+'sm-dir');

end;

procedure TForm1.Button2Click(Sender: TObject);
var
  telebe:array[1..10] of integer;
  yashcem,tel:integer;
  ort:string[5];
begin
yashcem:=0;
 for tel:=1 to 10 do
 begin
 telebe[tel]:=StrToInt(InputBox(IntToStr(tel)+
 'Telebenin yashi:','',''));
 yashcem:=yashcem+telebe[tel];
 end;
 ort:=FloatToStr(yashcem/10);
ShowMessage('Qrupun telebelerinin orta yashi:'+ort+'-dir');

end;

end.
```

Proqramı işə saldıqdan sonra Button1 düyməsi üzərində mausun sol düyməsini sıxdıqda aşağıdakı ekran pəncərəsi görüntüyə gəlir.

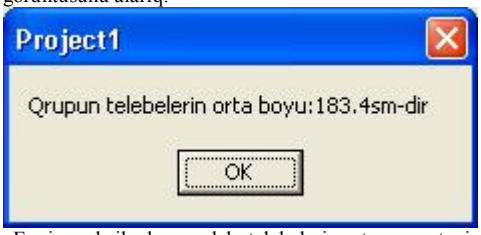




Birinci tələbənin boyunun uzunluğunu daxil edib OK düyməsini sıxdıqdan sonra Aşağıdakı ekran pəncərəsi görüntüyə gəlir.



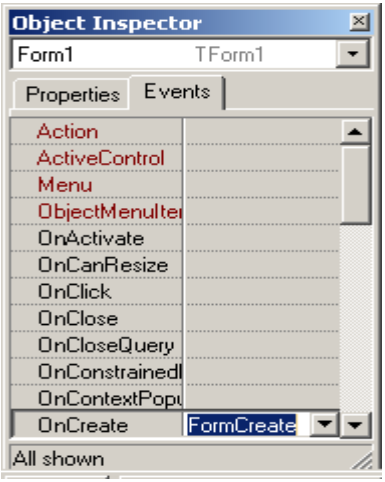
Bu qayda ilə bütün tələbələrin boyunun uzunluğunu daxil etdikdən sonra aşağıdakı ekran görüntüsünü alarıq:

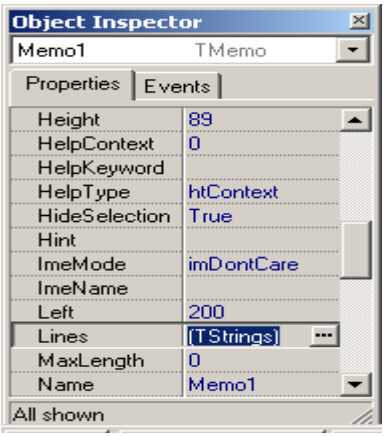


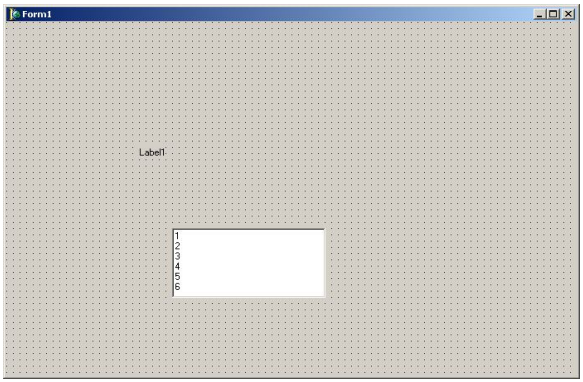
Eyni qayda ilə də qrupdakı tələbələrin orta yaşını təyin etmək olar.

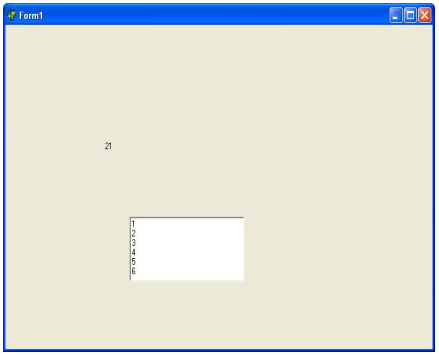
Massivlərin elementlərinin qiymətlərinin daxil edilməsi bir sıra hallarda çətinliklər törədir.Massivin elemetlərinin daxil edilməsinə aid misallara baxaq.

Misal 1. Birölçülü massivin elemetlərinin cəmini hesablayan proqram yazaq.Bunun üçün forma üzərində Memo1, Label1 komponentlərini yerləşdirək. Memo1 komponentinin Lines xassəsinin TStrings sətrində mausun sol düyməsini iki dəfə vuraraq mətn oblastı komponenti redaktorundan massivin elementlərini daxil edirik.Proqram modulunu Form1 forma konstruktorunun OnCreate hadisəsinin emaledicisinə yazırıq. Sonra isə daxil edilmiş elementləri C diskində rr1.doc faylına yazırıq.Sonra isə proqram vasitəsi ilə həmin elementləri fayldan oxuyub cəmləyirik.





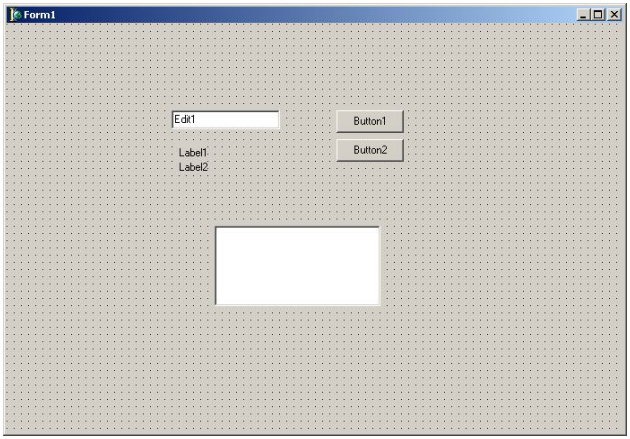


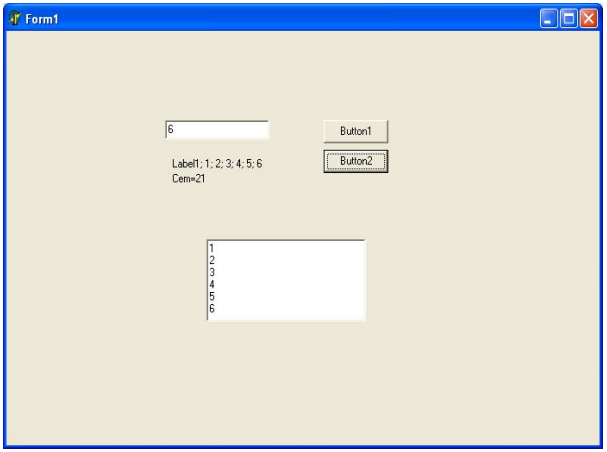



```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Memo1: TMemo;
    Label1: TLabel;
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
```

```
 end;

var
 Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
const
  n=6;
  var
    f:textfile;
    a:array[1..n] of real;
    a1:array[1..n] of string;
    i:integer;
    s:real;
begin
memo1.Lines.SaveToFile('c:\rr1.doc');
assignfile(f,'c:\rr1.doc');
reset(f);
i:=1;
s:=0;
while not eof(f) and(i<=n) do
begin
readln(f,a1[i]);
a[i]:=strtofloat(a1[i]);
s:=s+a[i];
inc(i);
end;
close;
label1.Caption:=floattostr(s);

end;

end.
```

Indi isə həmin misalı başqa üsulla həll edək.Bunun üçün forma üzərində Edit1, Memo1, Label1, Button1, Button2, Label2 komponentlərini yerləşdirək.



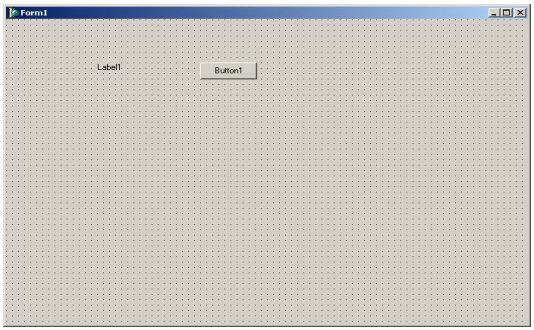


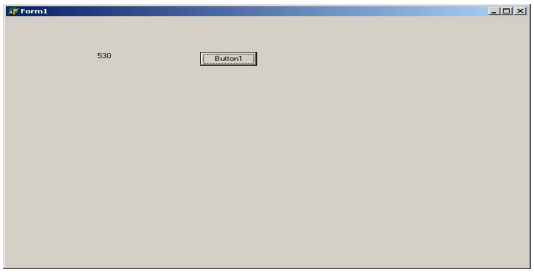

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;
```

```
type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Memo1: TMemo;
    Label1: TLabel;
    Button1: TButton;
    Button2: TButton;
    Label2: TLabel;
    procedure Edit1KeyPress(Sender: TObject; var Key: Char);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  i:integer;

   implementation

{$R *.dfm}

procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
if ord(key)=13 then
begin
edit1.SetFocus;
end;
 end;

procedure TForm1.Button1Click(Sender: TObject);
begin
memo1.Lines.Add(edit1.Text);
edit1.SetFocus;
end;

procedure TForm1.Button2Click(Sender: TObject);
var
   a:array[0..5] of integer;
   s,i:integer;

begin
s:=0;
for i:=0 to 5 do
begin
     a[i]:=strtoint(memo1.Lines[i]);
     s:=s+a[i];
     label1.Caption:=label1.Caption+'; '+inttostr(a[i]);
     label2.Caption:='Cem='+inttostr(s);
```

```
end;
end;

end.
```

Misal 2. T(2,5) matrisinin elementlərinin cəmini hesablayan proqram tərtib edin.






```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Label1: TLabel;
    procedure Button1Click(Sender: TObject);
```

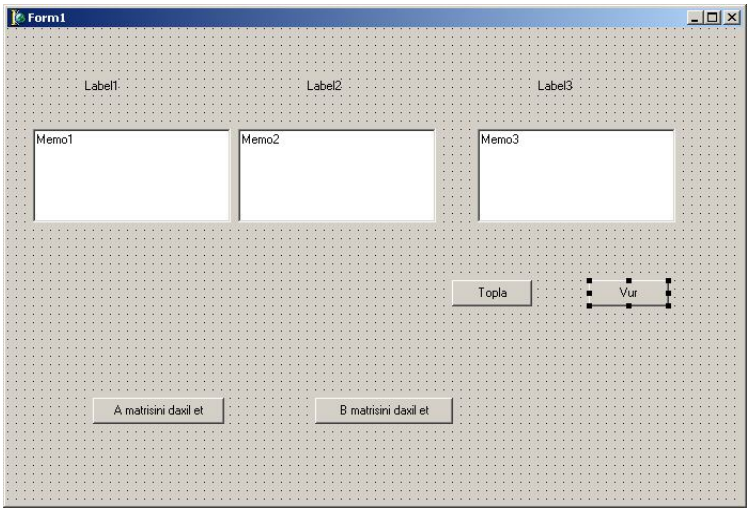
```
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
type t1=array[1..2,1..5] of byte;
const
   t:t1=((1,2,3,4,5),(101,102,103,104,105));
    var
      i,j,s:integer;
begin
s:=0;
   for i:=1 to 2 do
   for j:=1 to 5 do
   s:=s+t[i,j];
 label1.caption:=inttostr(s);
end;

end.
```

Misal 3. T(2,5) matrisinin hər hansı elementini ekrana çıxaran proqram tərtib etməli.


```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;
```

```
type
  TForm1 = class(TForm)
    Button1: TButton;
    Label1: TLabel;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
type t1=array[1..2,1..5] of byte;
const
   t:t1=((1,2,3,4,5),(101,102,103,104,105));
begin
 label1.caption:=inttostr(t[1,4]);
end;

end.
```

Misal 4.Iki 3x3 ölçülü matrisin toplanması və vurulmasını yerinə yetirən proqram yazaq.

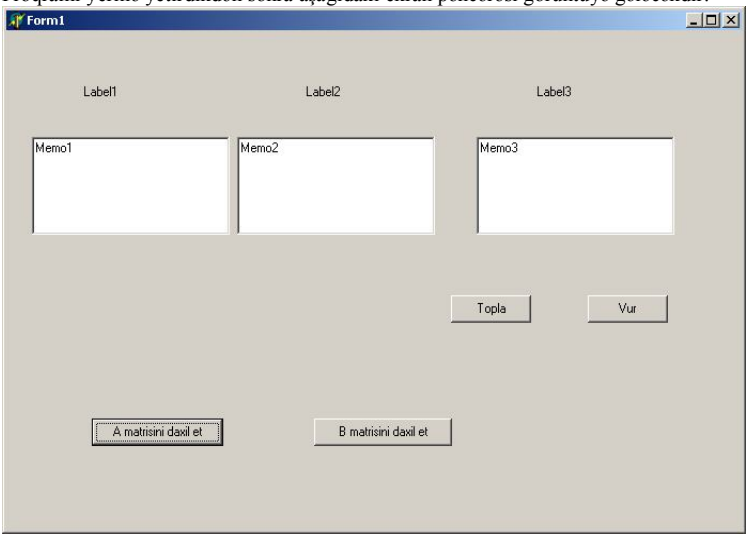


```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;
type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Memo1: TMemo;
    Memo2: TMemo;
    Memo3: TMemo;
    Button3: TButton;
    Button4: TButton;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
  a,b,c:array[1..3,1..3] of integer;
implementation
{$R *.dfm}
procedure TForm1.FormCreate(Sender: TObject);
begin
Memo1.ReadOnly:=True;
Memo2.ReadOnly:=True;
Memo3.ReadOnly:=True;
end;
procedure TForm1.Button1Click(Sender: TObject);
var  //A matrisni daxil et  duymesi
i,j:integer;
begin
Memo1.Text:='';
For i:=1 to 3 do
begin
for j:=1 to 3 do
begin
a[i,j]:=StrToInt(InputBox ('A matrisi',IntToStr(i)+','+IntToStr(j)+
 '.elementinin qiymeti :','1') );
Memo1.Text:=Memo1.Text+'   '+IntToStr(a[i,j]);
```

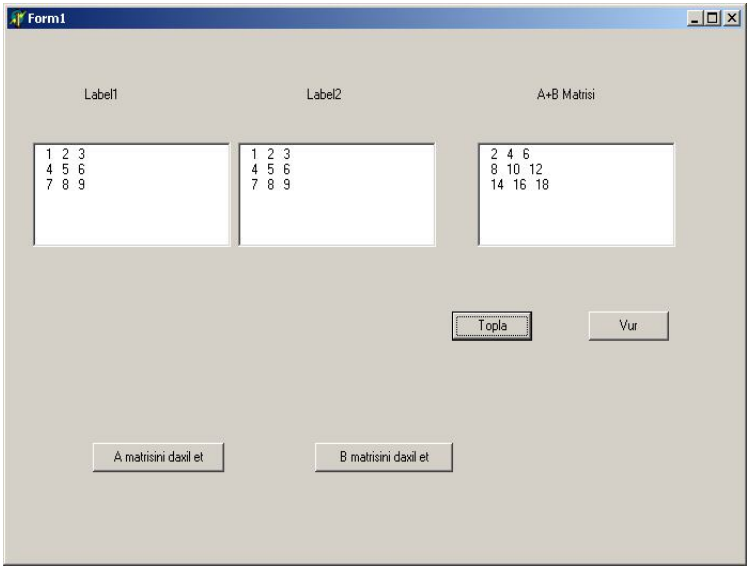
```
end;

Memo1.Text:=Memo1.Text+#13#10 {satir evveli}
end;
end;

procedure TForm1.Button2Click(Sender: TObject);
var  //B matrisni daxil et duymesi
i,j:integer;
begin
Memo2.Text:='';
For i:=1 to 3 do
begin
for j:=1 to 3 do
begin
b[i,j]:=StrToInt(InputBox ('B matrisi', IntToStr(i)+','+IntToStr(j)+
'.elementin qiymeti :','1') );
Memo2.Text:=Memo2.Text+'   '+IntToStr(b[i,j]);

end;
Memo2.Text:=Memo2.Text+#13#10 {satir evveli}
end;
end;

procedure TForm1.Button3Click(Sender: TObject);
Var// Topla  duymesi
i,j:integer;
begin
Label3.Caption:='A+B Matrisi';
Memo3.Text:='';
For i:=1 to 3 do
begin
For j:=1 to 3 do
begin
c[i,j]:=A[i,j]+B[i,j];
Memo3.Text:=Memo3.Text+'   '+IntToStr(c[i,j]);
end;
Memo3.Text:=Memo3.Text+#13#10
end;  end;

procedure TForm1.Button4Click(Sender: TObject);

Var//Vur duymesi
i,j,k,x:integer;
begin
Label3.Caption:='AxB Matrisi';
Memo3.Text:='';
For i:=1 to 3 do
For j:=1 to 3 do
begin
x:=0;
for k:=1 to 3 do
```

```
x:=x+A[i,k]*B[k,j];
c[i,j]:=x;
end;
For i:=1 to 3 do
Begin
For j:=1 to 3 do
Memo3.Text:=Memo3.Text+'  '+IntToStr(c[i,j]);
Memo3.Text:=Memo3.Text+#13#10
end;
end;

end.
```

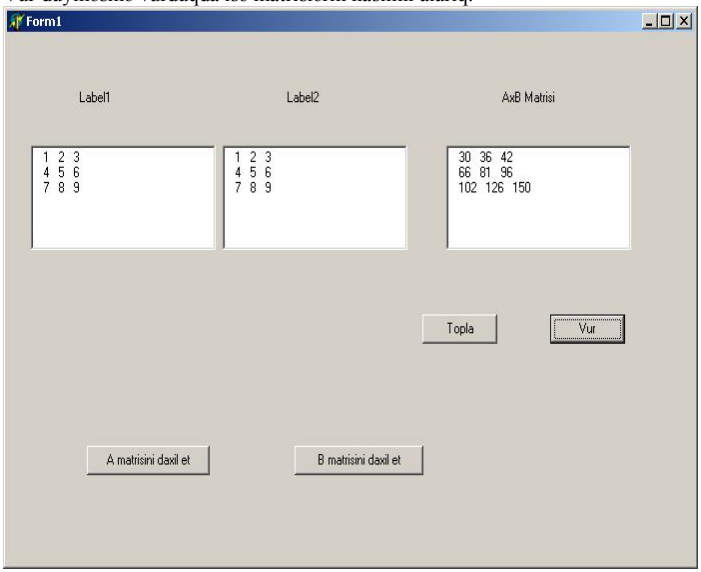
Proqramı yerinə yetirdikdən sonra aşağıdakı ekran pəncərəsi görüntüyə gələcəkdir.



Matrislərin qiymətlərini daxil etdikdən sonra Topla düyməsinə vurduqda matrislərin cəmini alarıq.



Vur düyməsinə vurduqda isə matrislərin hasilini alarıq.



7.9.9.2. Dinamiki Massivlər.

Delphi 4-cü versiyasından başlayaraq dinamiki massivlər daxil edilmişdir. Bu tip massivləri elan edərkən indekslərin sərhəddi göstərilmir:

```
var
        A:array of integer;
        B:array of array of char;
        C:array of array of array of real;
```

Bu misallarda A massivi birölçülü, B massivi ikiölçülü, C massivi isə üçölçülüdür. Dinamiki massiv yaddaşının bölünməsi və hər bir indeksin sərhəddinin göstərilməsi proqram yerinə yetirilərkən Setlength funksiyasının köməyi ilə inisiallaşdırma vasitəsi ilə həyata keçirilir:

```
Setlength(A,3);
```

Belə ki, dinamiki massivdə üç tam tipli ədədi yerləşdirməkdən ötəri yer ayrılır. Istənilən indeks üçün aşağı sərhəd 0-a bərabər olduğundan, A massivi üçün yuxarı sərhəd 2 olacaqdır.

Faktiki olaraq dinamiki massivin identifikatoru massivin yerləşdirilməsi üçün ayrılmış yaddaş sahəsinin birinci baytının göstəricisinə müraciət edir. Ona görə də bu yaddaşın təmizlənməsi üçün identifikatora NIL qiyməti mənimsətmək kifayətdir. (yaddaşın azad olunmasının digər üsulu Finalize prosedurunun köməyi ilə həyata keçirilir):

```
var
   A,B:array of integer;
begin
// yaddaş ayırırıq
  SetLength(A,10);
  SetLength (B,20);
// Massivlərdən istifadə
……
// yaddaşı təmizləyirik
  A:=NIL;
  Finalize (B);
end;
```

Artıq inisiallaşdırılmış dinamiki massivin hər hansı ölçüyə görə uzunluğunun dəyişdirilməsi üçün yeni massivin yerləşdirilməsi üçün yer ayrılır,sonra köhnə massivin elementləri yeni massivə köçürülür, bundan sonra əvvəlki massivə ayrılan yaddaş azad olunur.

Çox ölçülü massivlərdə əvvəlcə birinci ölçünün uzunluğu, sonra ikinci, üçüncü və s. təyin olunur. Məsələn,

```
var
  A:array of array of Integer //iki ölçülü massiv
  begin
  //birinci ölçü üçün uzunluğu təyin edirik
    Setlength (A,3);
  //hər bir sütunun uzunluğunu təyin edirik
  SetLength (A[0],3);
  SetLength ((A[1],3);
 SetLength (A[2],3);
 …….
 end;
```

Standart Pascal (həmçinin Object Pascal-ın) adi massivindən fərqli olaraq dinamiki massivlərdə ikinci və sonra gələn indekslər müxtəlif uzunluğa malik ola bilərlər. Əvvəlki misalda 3x3 ölçülü kvadrat massiv təyin olunmuşdur. Eyni qayda ilə bir üçbucaq matris də düzəldə bilərik.

```
SetLength (A,3);
```

```
//hər bir sütunun uzunluğunu veririk
SetLength (A[0],3);
SetLength (A[1],4);
SetLength (A[2],5);
```

Çoxölçülü dinamiki massivlərdə hər bir element ixtiyari (N-l) ölçüyə (N ölçülərin sayıdır) görə dinamiki massivdir və deməli, inisiallaşdırılmalıdır. Məsələn, həqiqi kubik 3x3x3 ölçülü masivi aşağıdakı kimi inisiallaşdırmaq olar.

```
var
  A:array of array of array of real;
     i,j: integer;
begin
   Set Length (A,3);
for i:=0 to 2 do
          begin
            Set Length (A[i],3);
          end;
          ….
          end;
```

7.9.9.3. Yazılar.

Yazı-qeyd olunmuş sayda yazının sahələri adlanan komponentlərdən təşkil olunmuş verilənlər strukturudur. Massivlərdən fərqli olaraq yazının komponentləri müxtəlif tipli ola bilər. Yazının bu və ya digər komponentinə (sahəsinə) müraciət etmək üçün onlar adlandırılır.

Bəzi tip məsələlərin həllinin proqramlaşdırılması prosesində bəzən qeyri bircins, yəni müxtəlif tipli, komponentlərdən ibarət obyektlərlə müxtəlif əməliyyatlar aparmaq lazım gəlir. Məsələn, tutaq ki, şəhərin 30 müəssisəsi üçün aşağıdakı məlumatlar toplanıb.

1) müəssisənin adı-ən azı 20 işarə, sətir tipli.
2) Işçilərin sayı-tam ədəd, tam tipli
3) Ekoloji nəzarətin olması faktı- Bul sabiti, məntiqi tipli
4) Havaya buraxılan zərərli qazların miqdarı tam ədəd, tam tipli

Havaya ən çox zəhərli qaz buraxan müəsisənin adını, orada çalışan fəhlələrin sayını və ekoloji nəzarətin olması faktını ekrana çıxaran proqram tərtib etmək tələb olunur.

Əvvəlcə bu proqram massiv tipli dəyişənlərdən istifadə etməklə tərtib etməyə cəhd edək. Bu məqsədlə 4 ədəd 30 elementli massiv: müəssisənin adını özündə saxlayan və elementlərin tipi string[20] olan massiv, işçilərin sayını özündə saxlayan və elementlərinin tipi Integer olan massiv, ekoloji nəzarətin olması faktını özündə saxlayan Boolean tipli massiv, havaya buraxılan zəhərli qazların miqdarını özündə saxlayan və elementlərinin tipi integer olan massiv.

Bu zaman hər bir müəssisəyə aid olan məlumat eyni indeksli massivlərdə yerləşir. Tələb olunan müəssisəni tapmaq üçün əvvəlcə havaya buraxılan qazların miqdarını özündə saxlayan massivdə ən böyük element və onun indeksini tapmaq, sonra isə bu indeksli elementləri çapa vermək lazımdır. Pascal dilində bu tipli proqramlar tərtib etmək üçün eyni tip dəyişənlərdən ibarət massivlərlə yanaşı, müxtəlif tipli dəyişənləri eyni ad altında birləşdirərək yazı (Record) tipindən geniş istifadə olunur. Yazı tipli dəyişənlər iki şəkildə-sadə və variantlı təsvir oluna bilərlər. Sadə təsvirə baxaq.

```
TYPE
        <tipin_adı>=RECORD
        D1:tip;
```

```
                D2:tip;
                …..
                DK:tip;
                END;
        var
                x:<tipin_adı>;
```

Burada RECORD azərbaycan dilində “yazı” deməkdir. RECORD və END işçi sözlərdir. x-dəyişəni yazı tiplidir, D1….,DK x dəyişənin komponentləridir. “Tip” olaraq yazı tipi də daxil olmaqla ixtiyari tip götürülə bilər.

Yuxarıdakı misalı aşağıdakı kimi yazmaq olar:

```
TYPE
        MUES=RECORD
        M_adi:string [20];
        I_sayi:integer;
        ENEZ:BOOLEAN;
        GAZ_m:integer;
end;
var
But_Mues:array [1..30] of mues;
```

Burada But_Mues hər bir elementi 4 komponentdən ibarət massivdir.

Göründüyü kimi istənilən müəssisənin göstəriciləri ancaq bir yazı tipli dəyişəndə cəmlənmişdir. Qeyd etdiyimiz kimi tip olaraq yazı tipidə göstərilə bilər. Məsələn, tutaq ki, müəssisənin göstəricilərinə onun iyirmi sexinin göstəricilərini də daxil etmişik. Hər sexin öz göstəriciləri var və aşağıdakılardır.

1) Sexin adı-iyirmi işarəyə qədər,işarə tipli
2)buraxılan məhsulun miqdarı-tam ədəd
3)işçilərin sayı-tam ədəd

Onda müəssisənin göstəriciləri sexin göstəriciləri də nəzərə alınmaqla aşağıdakı kimi təsvir oluna bilər.

```
        TYPE
                Se_type=record
                S_adi:string[20];
                M_miq:integer;
                S_is_s: integer;
        end;
        var
        mues: record
m_adi:string[20];
        Se:array[1..20] of Se_type;
I_sayı: integer;
                enez: boolean;
                gaz_m:integer;
        end;
        but_mues:array[1..30] of mues;
```

But_mues massivinin 720 komponenti var. Hər bir dəyişənə müraciət etmək üçün onun daxil olduğu yazı tipli dəyişənin adı yazılır, sonra nöqtə qoyulur və komponentin adı yazılır. Komponentlərin adı ayrıca yazıla bilməz.

Məsələn, proqramda

```
                write(m_adi); yazsaq səhv olacaq.
```

Çünki onun üçün **m_adi** dəyişəni deyil, mues yazı tipli dəyişənin **m_adi** komponenti mövcuddur və bu belə yazılır:

mues.m_adi

Yazı tipli dəyişənlər variantlar şəklində də təsvir oluna bilər. Tutaq ki, s1 yazı tipli dəyişəni həqiqi x1, x2 komponentlərindən, s2 yazı tipli dəyişəni işarə tipli y1, y2, y3 komponentlərinidən, s3 yazı tipli dəyişəni sətir tipli z1, z2, z3 komponentlərindən ibarətdir. s1, s2, s3 yazı tipli dəyişənlərini bir yazı tipli dəyişən kimi aşağıdakı qaydada elan etmək olar. Məsələn,

```
TYPE
        DEY=(bir,iki, uc);
var
        D:dey;
        Per:record
                d:dey of
                        bir:(x1, x2:real);
                                                iki:(y1,y2,y3:char);

                                                uc:(z1,z2,z3:string);
                end;
```

Burada **Per** yazı tipli dəyişəninin variant hissəsi CASE işçi sözü ilə başlayır, işlənmə prinsipinə görə variant operatorunu xatırladır. Variantda olan D dəyişəninə bəzən yazının diskrimnantı da deyilir. Onun qiymətindən asılı olaraq arzu olunan komponentlər variant hissəsindən seçilir. Yazı tipli dəyişənin təsvirində ancaq bir variant operatorundan istifadə oluna bilər.

with..do operatoru.

Bu operator aşağıdakı kimi yazılır.

```
with <yazı tip dəyişənin adı>  do
        <mürəkkəb operator>  ;
```

Burada göstərilmiş mürəkkəb operatorda yazı tipli dəyişənin komponentinə adı ilə müraciət etmək mümkündür. Uyğun yazı tipli dəyişənin adı with xidmətçi sözündən sonra göstərilir. Məsələn, yuxarıda göstərilmiş mues yazı tipli dəyişənin komponentlərinə qiymət mənimsədilməsi aşağıdakı with…do operatorunun köməyi ilə həyata keçirilə bilər.

```
with mues do
        begin
m_adi:= 'BMZ';
i_sayi:=300;
ecol:=false;
gaz:=5000;
end;
```

Indi isə havaya ən çox zəhərli qaz buraxan müəssisənin adını, orada çalışan işçilərin sayını və orada ekoloji faktorun olmasının ekrana çıxarılmasını yazı tipli dəyişəndən istifadə etməklə aşağıdakı kimi tərtib etmək olar.

```
uses crt;
var
        mues:record
        m_adi:string[20];
        i_sayi:integer;
Ecol: boolean;
Gaz:integer;
end;
but_mues:array[1..30] of mues;
j, index, max:integer;
```

```
begin
        for j:=1 to 30 do
        with but_mues [j] do
read (m_adi, i_sayı, ecol,gaz);
max:=but_mues[1].gaz;
for j:=2 to 30 do
if max < but_mues[j].gaz then
begin
max:=but_mues[j].gaz;
index:=j;
end;
with but_mues[index]  do
write (m_adi, i_sayi, ecol);
end.
```

7.9.9.4. Çoxluq tipi

Çoxluq tipi elementləri bir qrup hərf və ya ədədlərdən ibarət olan tip olub, riyaziyyatda olduğu kimi bu tipə birləşmə, kəsişmə, daxildir, daxil deyil, tamamlama və çıxma əməllərini tətbiq etmək olar,

```
type
        Ededcox=set of byte:
        Herfcox=set of  'a'..'z';
Var
D,E,C,A,B:Ededcox;
  F:Herfcox;
Begin
        A:=[1,4,12,50,14];
        B:=[100,200,12,4];
```

Yuxarıda elementləri byte tipində olan Ededcox və 'a'-dan 'z'-ə qədər olan Herfcox çoxluq tipləri təsvir olunmuşdur. D,E,C,A,B dəyişənləri Ededcox, F dəyişəni Herfcox tiplərinə aiddirlər. Sonra isə A və B çoxluqları üzərində aşağıdakı əməliyyatlar aparılmışdır:

C:=A+B; [1,4,12,50,14]+[100,200,12,4]=[1,4,12,50,4,100,200]
D:=A*B ;[1,4,12,50,14]*[100,200,12,4]=[4,12]
E:=A-B =[1,4,12,50,14]-[100,200,12,4]=[1,14,50,100,200]

Bunlardan başqa iki çoxluğun elementləri eynidirsə A=B olduqda true qiyməti, çoxluqların elementləri muxtəlif olduqda A<>B halında true qiyməti qaytarılır. A çoxluğu A:=[1,2,3,4,5], B çoxluğu B:=[2,3]; bu halda A>=B şərti ödənir. Bu o deməkdir ki, A çoxluğu B çoxluğunu daxilinə alır.

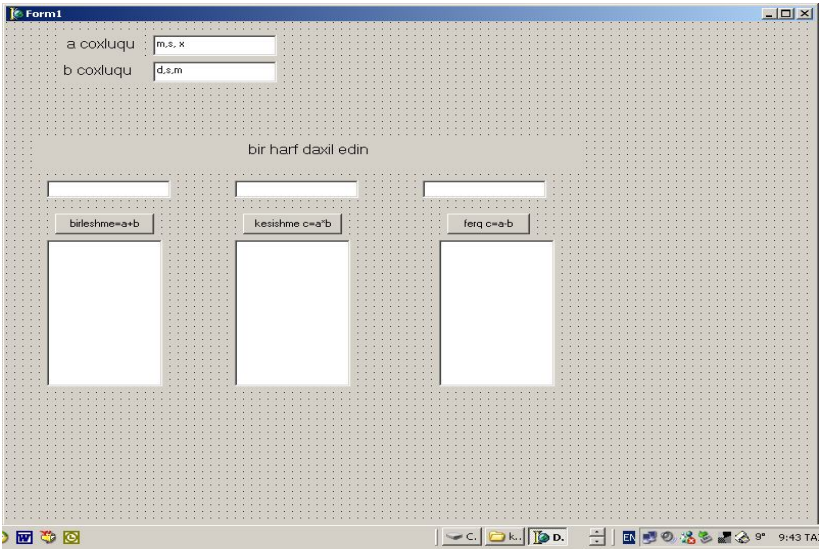
Hər hansı bir elementin çoxluğa daxil olması in operatoru ilə göstərilir.

Misal : A və B çoxluqları aşağıdakı kimi təyin olunmuşdur.

A:=['m','s','x'];
B:=['d','s','m'];

Bu çoxluqlar üzərində birləşmə ,kəsişmə və çıxma əməllərinin yerinə yetirilməsi məsələsini həll etməyə çalışaq.

Bunun üçün forma üzərində
Button1,Edit1,Button2,Edit2,Edit3,Button3,Edit4,Label1,
Label2,Edit5,Label3,Memo1,Memo2,Memo3 komponentlərini yerləşdirək.



Proqram kodu aşağıdakı kimi olar.

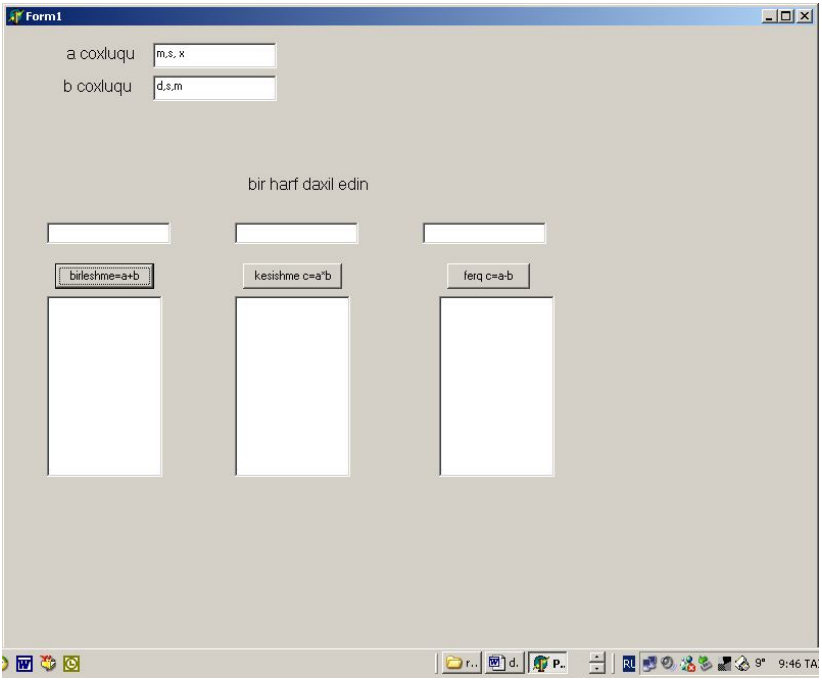
```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Mask;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Edit1: TEdit;
    Button2: TButton;
    Edit2: TEdit;
    Edit3: TEdit;
    Button3: TButton;
    Edit4: TEdit;
```

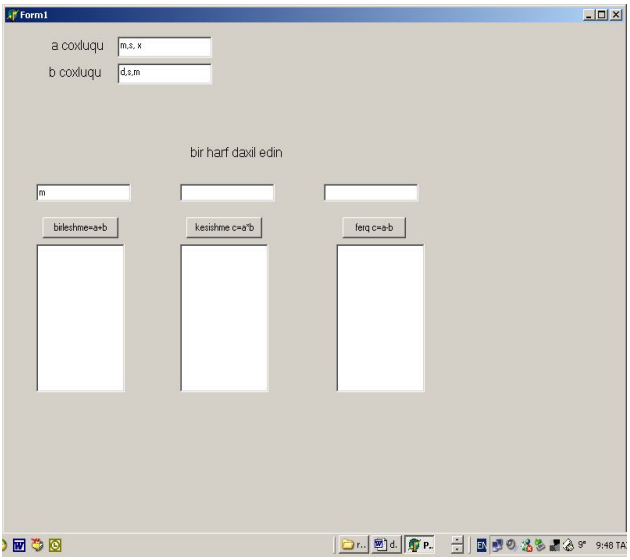
```
    Label1: TLabel;
    Label2: TLabel;
    Edit5: TEdit;
    Label3: TLabel;
    Memo1: TMemo;
    Memo2: TMemo;
    Memo3: TMemo;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Edit4Change(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
type
coxluq=set of char;
var
 a,b,c:coxluq;

begin
a:=['m','s','x'];
b:=['d','s','m'];


c:=a+b;


if edit1.Text[1] in c then
begin
showmessage('dogru');
memo1.Lines.Add(edit1.Text[1]);
edit1.Clear;
end


else
begin
showmessage('yanlis');
edit1.Clear;
end;
end;
```

```
procedure TForm1.Button2Click(Sender: TObject);

type
coxluq=set of char;
var
 a,b,c:coxluq;
 begin

a:= ['m','s','x'];
b:=['d','s','m'];


c:=a*b;

if edit2.Text[1] in c then
begin
showmessage('dogru');
memo2.Lines.Add(edit2.Text[1]);
edit2.Clear;
end

else
begin
showmessage('yanlis');
edit2.Clear;
end



 end;

procedure TForm1.Button3Click(Sender: TObject);

type
coxluq=set of char;
var
 a,b,c:coxluq;
 y:integer;
g:variant;
begin
a:= ['m','s','x'];
b:=['d','s','m'];


c:=a-b;

if edit3.Text[1] in c then
begin
showmessage('dogru');
memo3.Lines.Add(edit3.Text[1]);
edit3.Clear;
```

```
end

else
begin
showmessage('yanlis');
edit3.Clear;
end;
end;

procedure TForm1.Edit4Change(Sender: TObject);
type
coxluq=set of char;
var
 a,b,c:coxluq;
 y:integer;
g:variant;
begin
  for y:=1 to length(edit4.Text) do
 begin
 g:=g+edit4.Text[y];
  edit1.Text:=g;

 end;
 end;

end.
```

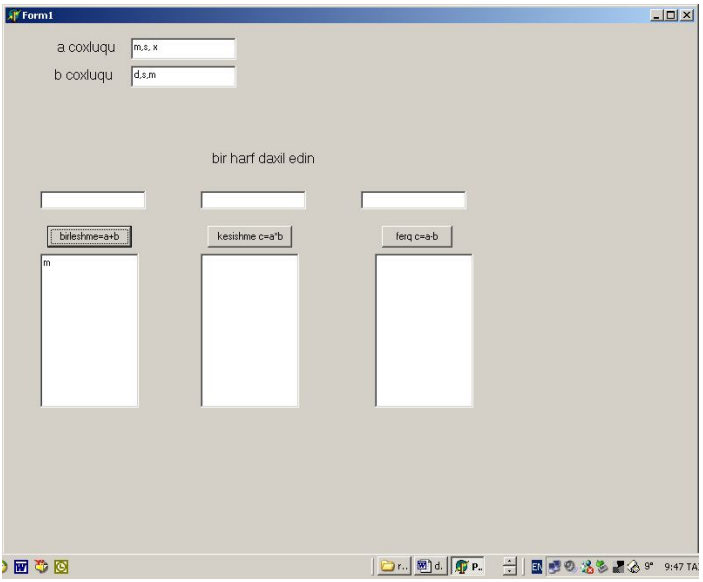
Proqram yerinə yetirildikdən sonra aşağıdakı ekran pəncərəsi görüntüyə gələcəkdir.

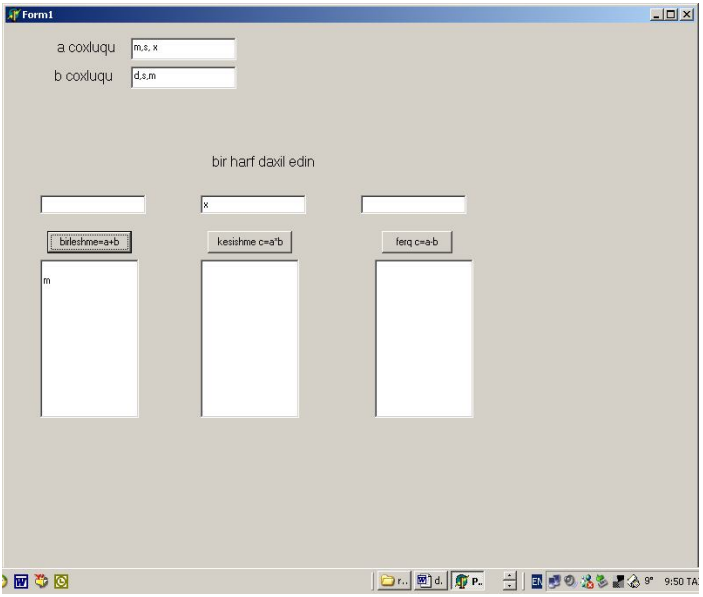

Form1
a coxluqu
m,s, x
b coxluqu
d,s,m
bir harf daxil edin
birleshme=a+b
kesishme c=a*b
ferq c=a-b


Form1
a coxluqu
m,s, x
b coxluqu
d,s,m
bir harf daxil edin
m
birleshme=a+b
kesishme c=a*b
ferq c=a-b

Edit1 mətn sətri komponentində ‘m’ işarəsini daxil edib, birleshme düyməsi üzərində mausun sol düyməsini vursaq aşağıdakı mesaj pəncərəsi ekrana gələcəkdir.

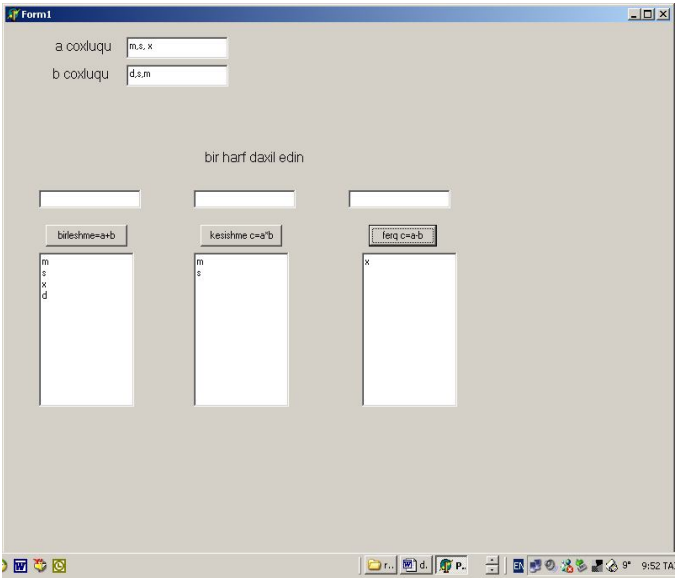
Bu mesaj pəncərəsi üzərində OK düyməsinin üzərində mausun sol düyməsini vursaq ,aşağıdakı pəncərə görüntüyə gələcəkdir.





Edit2 mətn sətri komponentindən ‘x’ işarəsini daxil edib kesishme düyməsi üzərində mausun sol düyməsini vursaq,aşağıdakı mesaj pəncərəsi alınacaqdır.Yəni ‘x’ işarəsi çoxluqların kəsişməsinə daxil deyildir.

Bütün elementlərin əməllərin nəticəsinə daxil olub-olmamasını yoxladıqdan sonra yekun olaraq aşağıdakı nəticəni alarıq.



7.9.10. Göstəricilər və dinamiki yaddaş

Dinamiki yaddaş-proqram işləyərkən kömpüterin ona ayırdığı əməli yaddaşdır. Verilənlərin dinamiki yerləşməsi dedikdə proqram işləyərkən ona ayrılan dinamiki yaddaş nəzərdə tutulur. Bundan fərqli olaraq statiki yerləşdirmə Object Pascal-ın kompilyatoru ilə kompilyasiya mərhələsində yerinə yetirilir. Dinamiki yerləşdirmədə yerləşdirilən verilənlərin tipi və sayı barədə heçnə məlum olmur.

Kompüterin əməli yaddaşı-hər birinin xüsusi nömrəsi olan (informasiyanı saxlamaq üçün) xanaların məcmuundan-baytlardan ibarətdir. Bu nömrələr ünvanlar adlanır və ünvana görə yaddaşın istənilən baytına müraciət etmək olar. Object Pascalda proramçılara dinamiki yaddaşı çevik idarə etmək üçün vasitə-göstəricilər daxil edilmişdir. Göstərici-yaddaş baytının ünvanını özündə saxlayan dəyişəndir. Göstəricilərin köməyi ilə istənilən tip dəyişəni dinamiki yaddaşda yerləşdirmək olar. Bu tiplərin bəziləri (Byte, Char, Shortint, Boolean) yaddaşda bir bayt, qalanları isə bir neçə qonşu baytlarda yerləşirlər. Ona görə də göstəricini verilənlərin birinci baytına ünvanlaşdırırlar.

Bir qayda olaraq göstərici hər hansı tip verilənlə əlaqələndirilir. Belə göstəriciləri tipli adlandırılacağıq. Tipli göstəricinin elan olunması üçün ^ (kare) işarəsindən istifadə olunur.

Məsələn,

```
var
    P1:^Integer;
    P2:^Real;
```

```
type
        PersonPointer= ^PersonRecord;
        PersonRecord=record
                Name:String;
                Job:String;
                Next:PersonPointer
end;
```

Proqramdan görünür ki, PersonPointer tipini elan edərkən biz əvəlcədən elan olunmayan PersonRecord tipinə istinad edirik.

Object Pascal-da hər bir istifadə olunan identifikatorun əvvəlcədən elan olunması prinsipi ardıcıl olaraq həyata keçirilir.Göstəricilər müstəsna hal kimi, elan olunmamış verilənlər tipinə istinad edə bilər.

Object Pascal-da göstəricisini heç bir verilənlər tipi ilə əlaqələdirmədən də elan etmək olar. Məsələn,

```
var
  P:Pointer;
```

Bu tip göstəricilər tipsiz adlanırlar. Tipsiz göstəricilər konkret olaraq heç bir tipə bağlı olmadıqlarından onların köməyi ilə strukturu və tipi proqram işləyərkən dəyişən verilənləri dinamiki olaraq yerləşdirmək əlverişlidir.

Qeyd etdiyimiz kimi, göstəricinin qiyməti yaddaşın ünvanını göstərən dəyişəndir. Ona görə də belə mülahizə yürütmək olar ki, bir göstəricinin qiymətini digərinə mənimsətmək olar. Lakin həqiqət də bu heç də belə deyil. Object Pascalda eyni tip verilənlərə bağlı olan göstəriciləri bir-birinə mənimsətmək olar.

Misala baxaq:

```
var
        PI1, PI2: ^integer;
        Pr:^real;
        P:Pointer;
```

Bu cür elan etmədə aşağıdakı birinci mənimsətmə mümkün, ikinci isə mümkün deyil.

```
PI1:=PI2;
PI1:=PR;
```

Səbəb odur ki, PI1 və PR göstəriciləri müxtəlif tip verilənləri ünvanlaşdırırlar. Bu məhdudiyyət tipsiz göstəricilərə şamil edilmir, ona görə də aşağıdakı operatorları yazmaqla lazımi nəticəni almaq olar:

```
P:=PR;
PI1:=P;
```

7.9.11. Dinamiki yaddaşın ayrılması və azad olunması

Object Pascalda bütün dinamiki yaddaşa bütöv baytlar massivi kimi baxılır və dəstə adlanır.

Istənilən dinamiki yerləşdirilən dəyişən üçün yaddaş New prosedurunun köməyi ilə ayrılır. Prosedurun müraciət parametri tipli göstərici olur. Müraciət nəticəsində göstərici verilənləri hansı mövqedən yerləşdirmək lazım olduğu qiyməti alır. Məsələn,

```
var
  PI, PJ:^Integer;
  PR: ^Real;
begin
   New(PI)
   New(PR);
     …..
```

```
end;
```

Göstərici müəyyən qiymət aldıqda daha doğrusu yaddaşda konkret fiziki baytı göstərdikdə, həmin ünvana uyğun tipli istənilən qiyməti yerləşdirmək olar. Bunun üçün mənimsətmə operatorunda göstəricidən sonra ^ işarəsi qoyulur. Məsələn,

```
PJ^:=2; // PJ yaddaş sahəsinə 2 ədədi yerləşdirilib.
PR^ := 2*P; //PR yaddaş sahəsinə 6,28 ədədi yerləşdirilib.
```

Beləliklə, göstəricinin ünvanlaşdığı qiymətlər, yəni verilənlər, göstəricidən sonra qoyulan ^ simvolu ilə işarə olunurlar. Əgər göstəricidən sonra ^ simvolu yoxdursa, onda verilənlər yerləşən ünvan başa düşülür. Buradan bu nəticəyə gəlmək olar:

Istənilən göstəricinin qiyməti ünvan, əgər bu ünvanda yerləşən verilənlə söhbət gedirsə, onda göstəricidən sonra ^ işarəsi qoyulmalıdır.

Dinamiki yerləşdirilmiş verilənlə proqramın istənilən yerində mümkün sabit və dəyişənlərlə birlikdə istifadə etmək olar. Məsələn,

```
PR^:=sqr(PR^)+PI^-17;
```

Aşağıda göstərilən operator yol verilən deyil, belə ki, PR göstəricisinə həqiqi qiyməti mənimsətmək olmaz.

```
PR:=sqr(PR^)+PI-17;
```

Eyni qayda ilə aşağıdakı operator da yol verilən deyil.

```
PR^:=sqr(PR);
```

Səbəb odur ki, PR göstəricisinin qiyməti ünvan olub, onu kvadrata yüksəltmək mümkün deyil.

PR^:=PJ; mənimsətmə operatoru da səhvdir. Belə ki, PR göstərici ilə təyin olunan ünvanda yerləşən həqiqi verilənə göstəricinin qiüymətini mənimsətmək olmaz.

Dinamiki yaddaşı yalnız dəstədən götürmək deyil, əksinə dəstəyə yaddaşı qaytarmaq da olar. Bunun üçün Dispose prosedurundan istifadə olunur. Məsələn, aşağıdakı operatorlar əvvəlcədən PJ və PR göstəricilərinə bərkidilmiş yaddaşı dəstəyə qaytarır.

```
Dispose (PJ);
Dispose (PR);
```

Qeyd etmək lazımdır ki, Dispose (pPtr) proseduru pPtr göstəricisinin qiymətini dəyişmir, yalnız yaddaşa dəstəyə qaytarır. Prosedurun təkrar sərbəst göstəriciyə tətbiqi icra mərhələsində səhvə gətirəcək. Azad olunmuş göstəricini proqramçı NIL ehtiyat sözü ilə qeyd edə bilər. Bu və ya digər göstəricinin qeyd olub-olmadığını aşağıdakı şəkildə yoxlamaq olar:

```
var
    PR: ^Real=NIL;
    Procedure MyProc;
    begin
    …..
    if PR=NIL then
       NEW(PR);
    …..
    DISPOSE (PR);
    PR:=NIL;
    …..
    end;
```

Göstəricilər üzərində heç bir müqayisə əməli aparmağa icazə verilmir.

Qeyd etdiyimiz kimi NEW prosedurunun parametri tipli göstərici olmalıdır. Tipsiz göstəricilərlə işləmək üçün GetMem və FreeMem prosedurlarından istifadə olunur:

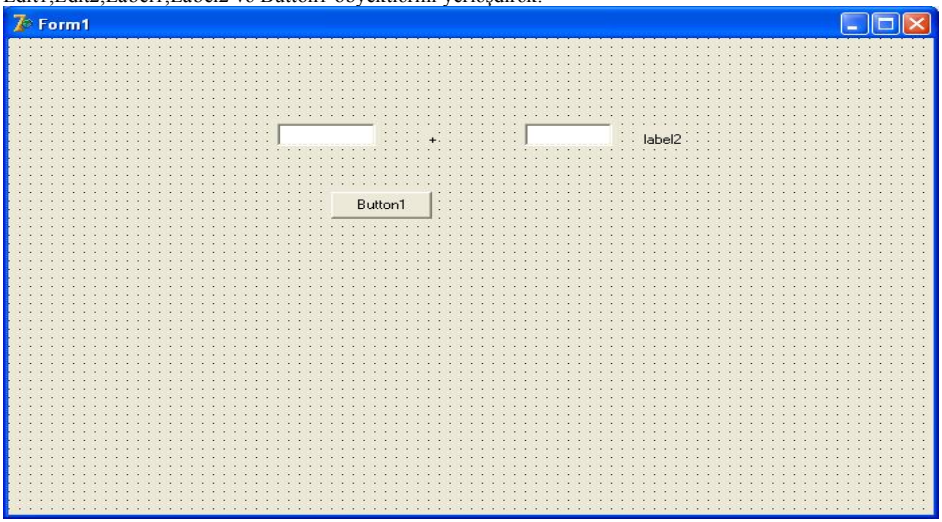
```
GetMem (P, Size)//yaddaşın ayrılması
FreeMem(P,Size)//yaddaşın azad olunması
```

Burada P-tipsiz göstərici, Size-dəstənin tələb və ya azad olunan baytlarla yaddaş ölçüsüdür.

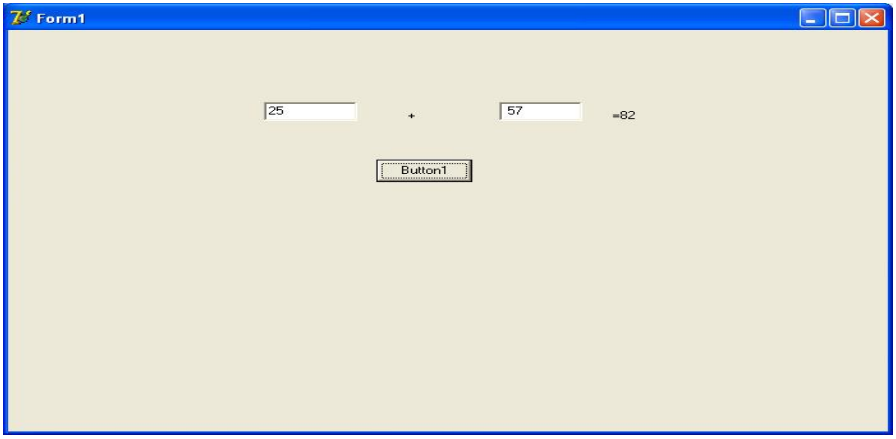
7.9.12. Variant tipli dəyişən

Indiyə qədər öyrəndiyimiz bütün dəyişənlər konkret olaraq tam,real,məntiqi,işarə və sətir tiplərdən birinə aid edilirdi.Hər hansı bir dəyişən tam tipli elan olunmuşdursa,həmin dəyişəni kompüterə kəsr ədəd kimi daxil etdikdə kəsilmə meydana gəlir və Delphi daxil edilən ədədin yararsız olduğu barədə məlumat verir. Variant tipli dəyişəndə isə yuxarıdakılardan fərqli olaraq tam,həqiqi, string,char və boolean kimi bir-birindən fərqli dəyişənlərdən istifadə etmək olar.

File▶ New ▶Application əmri ilə yeni bir proyekt yaradaq.Forma üzərində Edit1,Edit2,Label1,Label2 və Button1 obyektlərini yerləşdirək.



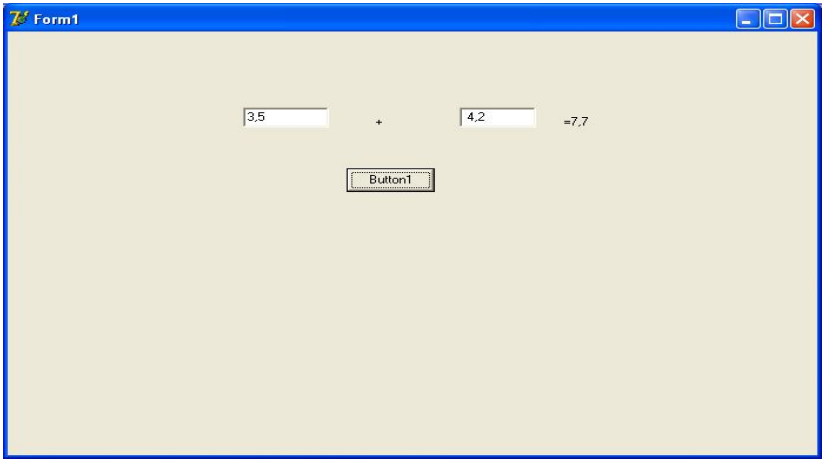
Button1 düyməsinin sıxılmasının emaledicisinə iki ədədin toplanması proqramını yazaq. Əvvəlcə bu proqramda Eded1,Eded2 və Netice dəyişənlərini tam tipli elan edək.



Proqram işlədikdə Eded1 və Eded2 kəsr ədəd daxil edilərsə,kəsilmə alınar.

Indi isə Eded1,Eded2 və Netice dəyişənlərini Variant tipli elan edək və proqramda StrToInt funksiyasının əvəzinə StrToFloat yazaq.Proqramı təkrar işə salıb kəsr ədədlər daxil etsək, proqram normal işləyəcəkdir.

```
var
  Form1: TForm1;
  eded1,eded2,netice:variant;
implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
begin
eded1:=StrToFloat(Edit1.Text);
eded2:=StrToFloat(Edit2.Text);
netice:=eded1+eded2;
Label2.Caption:='='+FloatToStr(netice);

end;

end.
```



7.10. Obyekt-Pascal-ın əsasıdır

Indiyə qədər biz Pascalda əvvəldən bildiyimiz verilənlər tipi ilə işləyirdik. Onların köməyi ilə olduqca böyük həcmli proqramlar yazmaq olar ki, bu da professional proqramçılardan çox böyük əməktutumlu iş tələb edir. Yalnız ədəd, sətir, massivlər və yazılardan istifadə edərək, ilkin mətni bir neçə min sətirdən ibarət proqram yaratmaq olar. Bu da insanın imkanlarının özünə məxsus əndazəsidir. Belə ki, ilkin alqoritmin verilənlərinin strukturu bir-biri ilə əlaqəsi olmayan çox kiçik elementar hissələrə bölünür. Onlar müxtəlif modullar üzrə paylanmış, mənimsətmə operatorlarının köməyi ilə əlaqələndirilirlər. Proqramın ilkin mətninin həcmi artıqda proqramın artıq normal işləyən hissəsinə toxunmadan dəyişənləri emal etmək mümkün olmur.

80-ci illərdə proqramçıların işinin məhsuldarlığını kəskin artıran obyekt yanaşmaya əsaslanan ilk kommersiya sistemləri işlənməyə başlandı.

Yanaşma obyekt anlayışına əsaslanır ki, burada qruplaşdırılmış verilənlərin xassələri ilə yanaşı onların emal metodları birləşir.

Obyekt faktiki olaraq bizi əhatə edən aləmin real və mücərrəd təsvirini verir. Məsələn, avtomobil özünün markası, mühərrikin tipi, təkərlərin sayı və sükanı, fayl isə ölçüsü və adı kimi xassələri ilə xarakterizə olmur. Avtomobilin «metodları» onun xassələrinə uyğun lazımi istiqamətdə hərəkət qabiliyyətini xarakterizə edir: bakdakı benzinin həcmi, sükanın dönmə bucağı. Fayldan istifadə edərək verilənləri saymaq, onları dəyişdirmək və yenidən yazmaq olar.

Obyekt anlayışına əsaslanaraq, proqramın layihələndirilməsini daha tez və sadə şəkildə yerinə yetirmək olar. Müxtəlif tipli proqramların yaradılması üçün, ixtisasçılar tərəfindən çoxlu sayda obyektlərin məcmuusu ayrılmışdır. Bu obyektlərdən onların proqramlaşdırılmasına vaxt sərf etmədən dəfələrlə istifadə etmək olar. Delphi mühitində bu cür yanaşma reallaşdırılmışdır.

7.10.1 Sinif anlayışı

Pascalda obyekt və sinif anlayışları arasında dəqiq məhdudiyyətlər vardır. Sinif-verilənlərin tipini, obyekt isə kompyuterin yaddaşında konkret mövcud olan sinifin ekzemplyarını göstərib, uyğun tipə aid olan dəyişəndir.

Pascalın əvvəlki versiyalarında obyekt tipin təsviri üçün object açar sözündən istifadə olunurdu və eyni zamanda bu tipin ekzemplyarı da obyekt adlanırdı. Indi yenə Delphi-

nin əvvəlki versiyaları ilə uyuşanlığı təmin etmək üçün object sözündən istifadə etmək olar, lakin object açar sözünün əvəzinə class açar sözündən istifadə etmək daha düzgün olardı.

Sinif sahəyə (record verilənlər tipi kimi), xassələrə (sinifin daxili strukturunun deklarasiyasının ciddiliyini artırmağa imkan verən, verilənlərin oxunma və yazılma mexanizmlərini təyin edən əlavə təsvir edicilərə malikdir) və metodlara (sinifin sahə və xassələrini emal edən altproqmlara) malikdir.

Class tipinə aid dəyişən təsvir olunursa, onda yaddaşda onun sahə və xassələrinə uyğun yaddaş sahəsi ayrılır, lakin sinifin metodlarını translyasiya edən maşın kodu yeganə ekzemplyarda olur. Obyekt yaranarkən bir dəfə xüsusi metod konstruktor cağrılır. Konstruktorun köməyi ilə obyektin sahəsinin başlanğıc inisializasiyasına aid müxtəlif işlər görülür. Obyekt ləğv olunarkən (məsələn, o prosedura daxilində lokal dəyişən kimi elan olunmuşdursa, proseduranın yerinə yetirilməsi başa çatdıqdan sonra yaddaşdan silinir) digər metod-destruktor çağrılır ki, zəruri hallarda yaddaşın azad olunmasına aid müxtəlif işləri görür. Konstruktor və destruktoru aşkar şəkildə proqram kodu vasitəsi ilə çağırmaq mümkün deyil. Bu avtomatik olaraq baş verir.

Sinif kimi elan olunan dəyişən, faktiki olaraq ekzemplyarına göstərici rolunu oynayır. Bu siniflərlə sinfin işin effektivliyini artırmaq üçün nəzərdə tutulmuşdur. Lakin belə dəyişənlərdən istifadə edərkən göstəricilərlə işləmək əməliyyatların (^,(@) tətbiq etmək lazım deyildir. Onlara dəyişənlərə müraciət kimi, sinifin hissəsinə isə yazının sahəsinə müraciətdə olduğu kimi nöqtənin köməyi ilə müraciət etmək kifayətdir.

7.10.2 Obyektyönlü proqramlaşdırmanın üç prinsipi

Əvvəldən yaradılmış varislik sinifdən təkrarən effektiv istifadə etmək üçün verilənlər və metodların vahid strukturda birləşdirilməsi kifayət deyil. Məsələn, avtomobil minik və yük avtomobilləri ola bilər ki uyğun sinifləri ümumi sahə və metodlara malik olmaqla yanaşı fərqlənə də bilərlər. Lakin köhnə tipdə bir neçə xassəni dəyişmək və ya əlavə etmək olunursa, yenidən tam yeni verilənlər tipi təyin etmək əlverişli deyildir. Bu dir də ona görə əlverişli deyildir ki, əgər hər iki sinifdə olan metodda dəyişiklik etmək tələb olunursa, bu işi iki dəfə altproqramın iki eyni surəti üzərində yerinə yetirmək lazımdır.

Obyekt yönlü proqramlaşdırmada bu cür lazımsız işlərdən qaçmaq üçün xassə və metodların varislik prinsipi daxil edilmişdir.

Proqramçıya bir baza sinfi (məsələn, “avtomobil”) təsvir edib, “minik avtomobil”-i və “yük avtomobil”-i siniflərini bu baza sinfinin əsasında yaratmaq lazımdır. Yeni yaradılmış siniflər baza (və ya valideyn) sinfinin bütün sahə, xassə və metodlarına varislik edəcəklər və onları yenidən təsvir etməyə heç bir ehtiyac yoxdur.

Varislik zənciri qeyri-məhdud uzunluqda ola bilər. Belə ki, “yük avtomobili” sinfinin əlavə xassə və metodlara malik varis sinifləri (və ya övlad sinifləri) “MAZ” və “KAMAZ” ola bilər. (Bunlar obyekt deyil sinifdir. Avtomobilin markası deyil, konkret MAZ yük avtomobili obyekt olacaqdır) “Düymə” sinfinin “qrafiki düymə” və “dairəvi düymə” və s. kimi varis sinifləri mövcuddur. Bununla yanaşı hər bir varis üçün müxtəlif metodları yenidən təyin etməyə icazə verilir.

Məsələn, “hərəkət etmək” metodu “MAZ” və “KAMAZ” sinifləri üçün mövcüd olub bir qədər fərqlənəcəklər: yanacaq müxtəlif miqdarda işlənilir, sürəti müxtəlif cür götürürlər v əs.

7.10.2.1. Polimorfizm

“KAMAZ” sinfinə məxsus dəyişənə müraciət, və varis olaraq götürülmüş “hərəkət etmək” metodunu çağırdıqda proqram hansı metodu çağırmaq lazım gəlməsi məsələsini həll etməlidir.

“Avtomobil” sinfinin, “yük avtomobili” sinfinin və ya “KAMAZ” avtomobili sinfinin metodunu. Polimorfizm prinsipinə görə həll bu metodu çağıran dəyişənin tipinə görə

qəbul olunur. Belə ki, əgər dəyişən "KAMAZ" tipinə məxsus dəyişən kimi elan olunmuşdursa, onda "KAMAZ" tipinə məxsus "hərəkət etmək" metodu çağrılmalıdır.

7.10.2.2. Inkapsulyasiya

Sinif sahələr, metodlar və xassələrin bir arada cəmlənməsini təmin edir. Sinifin bu xassəsi inkapsulyasiya adlanır. Inkapsulyasiya sinfi proqramın qalan hissələrindən izolə etməyə və konkret məsələnin həlli üçün onu "yararlı" etməyə imkan verir. Nəticədə sinif həmişə özündə müəyyən funsionallığı saxlayır. Məsələn, TForm sinfi Vindovs pəncərənin yaranması üçün zəruri olanları özündə cəmləşdirir.

Inkapsulyasiya işə hazır proqram məhsullarının mübadiləsi üçün çox güclü vasitədir.

7.10.3. Sahə

Sinifdə cəmlənmiş verilənlər sahə adlanır. Sahə istənilən tip, o cümlədən sinif tip də ola bilər. Məsələn:

```
type
TMyClass=class
aIntField:Integer;
aStrField:String;
aobj Field:TObject;

end;
```

Hər bir obyekt unikal sahələrin məcmuunu alır, lakin verilmiş sinfin bütün obyektləri üçün xassə və metodlar ümumidir. Inkapsulyasiyanın əsas prinsipi sahələrə xassə və metodların köməyi ilə müraciət etməkdir. Object Pascal da sahələrə birbaşa da müraciət etmək olar.

```
var
aObject: TMyClass;
begin
aObject aIntField:=0;
aObject aStrField:='Sebuhi';
end;
```

Varis sinif özünün sələflərinin bütün sahələrinə malik olur və onu yeni sahə ilə tamamlaya bilər. Eyni zamanda varis sinif öz sələflərinin sahəsindən imtina edə və ya ləğv edə bilməz. Beləliklə sinif nə qədər aşağı ierarxik səviyyədə yerləşirsə, onun obyektləri daha çox verilənlərdən istifadə edə bilər.

7.10.4. Sinfin təsviri

Yeni tip class açar sözünün köməyi ilə təsvir olunur

```
type sinfin_adı
class(valideyn_sinfin_adı)
sinfin_elementlərinin_siyahısı;
end;
```

Yeni sinif valideyn sinfin bütün xarakteristikalarına varislik edir. Hər hansı elementi valideyn sinfin xarakteristikalarına varislikdən azad etmək olmaz.

Əgər valideyn sinfi yazmasaq onda yeni sinif saymaya görə TObject baza sinfinin bütün xarakteristikalarına varislik edəcəkdir.

```
type sinfin_adı
class
sinfin_elementlərinin_siyahısı;
end;
```

7.11. Dinamiki qoşulan kitabxanalar (DLL)

Proqramlar arası informasiya mübadiləsini təmin etmək üçün ən sadə üsul .DLL (Dynamic Linking Library) kitabxanasından istifadə etməkdir. Əslində DLL kitabxanası proqram olmayıb, proqram kodu (məsələn, funksiya) və resursların (məsələn, forma) saxlancıdır. Onlar proqrama dinamiki olaraq, yəni proqram isə salındıqdan sonra qoşulurlar.

.DLL kitabxanasından istifadə etmək çox əlverişlidir. Məsələn, fərz edək ki, istifadəçi tərəfindən daxil edilən qiymətlər əsasında hər hansı qiymətin proqnozlaşdırılması proqramı vardır. Istifadəçi interfeysi yaradaq və proqnozlaşdırılmanın hesabat alqoritmini (kifayət qədər mürəkəb) ayrıca funksiyada verək və proqrama qoşulan DLL kitabxanasında yerləşdirək.

Indi, əgər tədqiqatçı proqnozlaşdırma alqoritmini təkmilləşdirirsə, onda istifadəçiyə yalnız DLL kitabxanasında dəyişiklik etmək lazım gələcək, interfeysə cavabdeh proqram kodunda heç bir dəyişikliyə ehtiyac olmayacaq.

7.11.1. DLL kitabxanasının yaradılması

Fərz edək ki, bizə iki ədədi toplama funksiyasını özündə saxlayan kitabxana yaratmaq lazımdır.

```
function Sum(x,y:integer):integer;
```

File▶ New▶ Other ▶DLL Wizard əmrlərini yerinə yetirək. Bu zaman boş kitabxana modulu yaranacaq. Modulun mətni Unit sözü ilə deyil library sözü ilə başlayır. Buraya Sum funksiyasının təsvirini, kitabxana modulunun axırına isə exports açar sözünü əlavə edək. Bu açar sözündən sonra digər əlavələr tərəfindən istifadə olunan verilmiş kitabxananın-export olunan funksiyalarının siyahısını vermək lazımdır. Bütünlükdə modul aşağıdakı kimi yazılır:

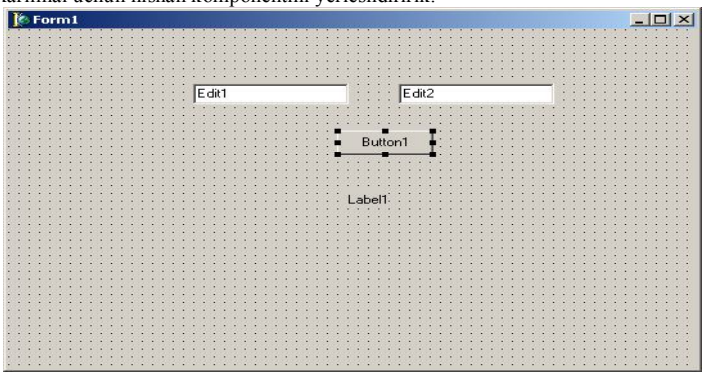
```
library topla;

{ Important note about DLL memory management: ShareMem must be the
  first unit in your library's USES clause AND your project's (select
  Project-View Source) USES clause if your DLL exports any procedures or
  functions that pass strings as parameters or function results. This
```

```
  applies to all strings passed to and from your DLL--even those that
  are nested in records and classes. ShareMem is the interface unit to
  the BORLNDMM.DLL shared memory manager, which must be deployed along
  with your DLL. To avoid using BORLNDMM.DLL, pass string information
  using PChar or ShortString parameters. }

uses
  SysUtils,
  Classes;

{$R *.res}
function sum(x,y:integer):integer;
begin
sum:=x+y;
end;
exports
     sum;

begin
end.
```

Bundan sonra Project menyusundan Compile topla(və ya Ctrl+F9) əmrini veririk.

7.11.3. DLL kitabxanasina muraciet

Yeni proyekt yaradaq. Forma ucherine iki metn sahesi, duyme ve neticenin chixarilmai uchun nishan komponentini yerleshdirirk.



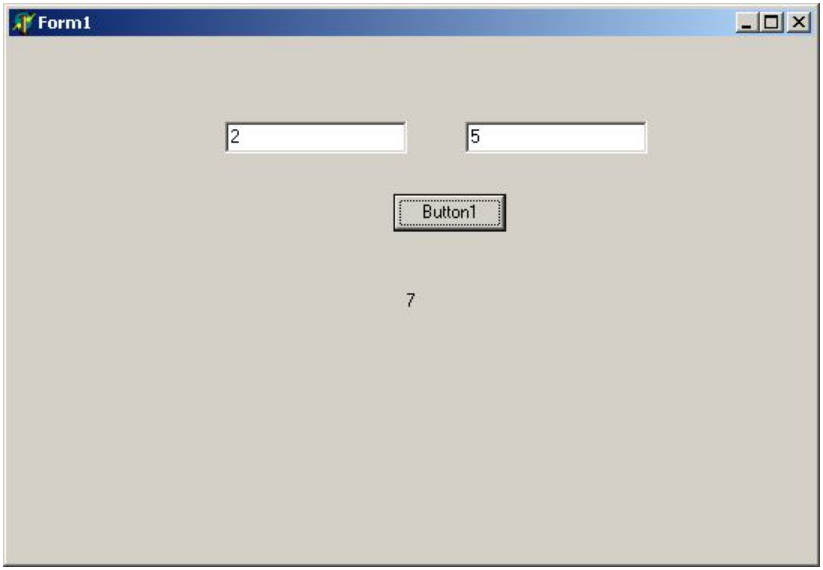
Interface bölmesnde eksport olunanunksiyanın təsvirini və external sözündən sonra faylın adını (.DLL kitabxanasını) göstərən sətiri verək.

function SUM (x,y:integer):integer;external ‘topla.dll’;

Düymənin emaledicisi aşağıdakı kimi olacaq:

```
procedure TForm1.Button1Click(Sender:TObject);
begin
Label1.Caption:=IntToStr(Sum(StrToInt(Edit1.Text),
StrToInt(Edit2.Text)));
end;
```

Indi artıq proqramı isə salmaq olar.



7.11.4. Resursların kitabxanaya əlavə olunması

DLL kitabxanasında yalnız proqram kodu deyil, həm də resursları, xüsusi halda Delphi6-nın formasını saxlamaq olar.

Project1.dll proyektini aktivləşdirək və File▶New Form əmrlərini yerinə yetirək. Cari proyektdə yeni forma alınacaqdır. Ona bir düymə əlavə edək və onun emaledicisi aşağıdakı kimi olsun:

```
Procedure TForm2.Button2Click(Sender:TObject);
begin
showMessage('Sebuhi')
end;
```

Formanı digər əlavələrdən çağırmaq üçün, onu dinamiki olaraq yaratmaq lazımdır, belə ki, Delphi6-nın adi mexanizmləri DLL kitabxanalarında işləmir. Bunun üçün yeni prosedura yazaq.

```
Procedure ShowMyForm (AOwner:TComponent);
Var MyForm:TForm2;
begin
MyForm:=TForm2.Create (AOwner);
MyForm.ShowModal;
MyForm.Free;
end;
```

Bu prosedura formanın konstruktoru üçün zəruri olan TComponent tipli parametrə malik olmalıdır. Yaradılan proseduru exportlar siyahısına əlavə edək.

```
exports sum, ShowMyForm;
```

Sonra Project2.exe proyektində yeni import olunan proseduru göstərək.

procedure showMyForm(AOwner:TComponent);

```
external 'project1.dll';
```

Formaya yeni bir düymə əlavə edək və emaledicisini aşağıdakı kimi təyin edək:

```
Procedure Tform1.Button2Click(Sender:TObject);
begin
  ShowMyForm(Self);
end;
```

Proqramı işlətdikdən sonra, bu düymə üzərində mausun düyməsini vursaq, DLL kitabxanasından götürülmüş işləyən forma ekrana çıxacaq

8. ActiveForm komponenti

Active Form-ların əsas funksiyası Delphi işləmələrini Web səhifələrinə yerləşdirməkdir. Yazacığımız proqramın Internet və ya Intranet şəbəkəsində Web istifadə oluna bilməsi üçün proyekti Active Form olaraq yaratmaq lazımdır. Microsoftun ActiveX texnoloqiyasına əsaslanan Active Form-u html səhifəsinə yerləşdirdikdən sonra, istifadəçiləri onu işlətməsinə nail olmaq olar.

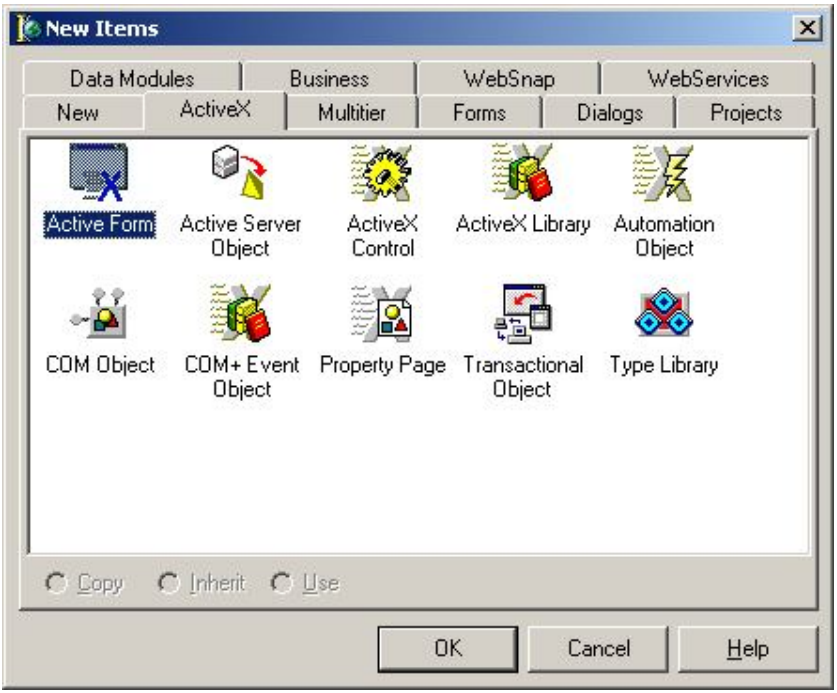
Active texnoloqiyası ilə yaranan fayllar OCX faylları adlanır və bu faylları Delphi və Cit+Builder komponentlər palitrasına yerləşdirərək, bir komponent kimi istifadə etmək olar. OCX fayylarını Web səhifəsinə yerləşdirərək və ya Vizual Basic alətlər çubuğuna qoyaraq istifadə etmək olar.

8.1 ActiveForm yaradılsması

Delphi-nin ActiveForm Wizardından istifadə edərək əyani olaraq, ActiveForm yarada bilərik.

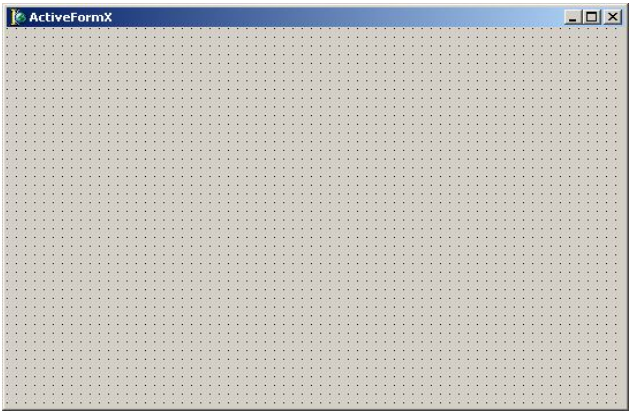
Yeni bir AcitveForm yaratmaq üçün File ▸ Close all əmrləri bütün faylları bağlayırıq.

File ▸ New ▸ Other əmrləri ilə açılan pəncərədən ActiveX səhifəsini seçin.



Active səhifəsindəki AcitveForm variantını seçdikdən sonra OK düyməsini basın. ActiveForm sehrbazı forma ilə bağlı bəzi əla xassələri vermək üçün aşağıdakı pəncərəni açacaqdır.

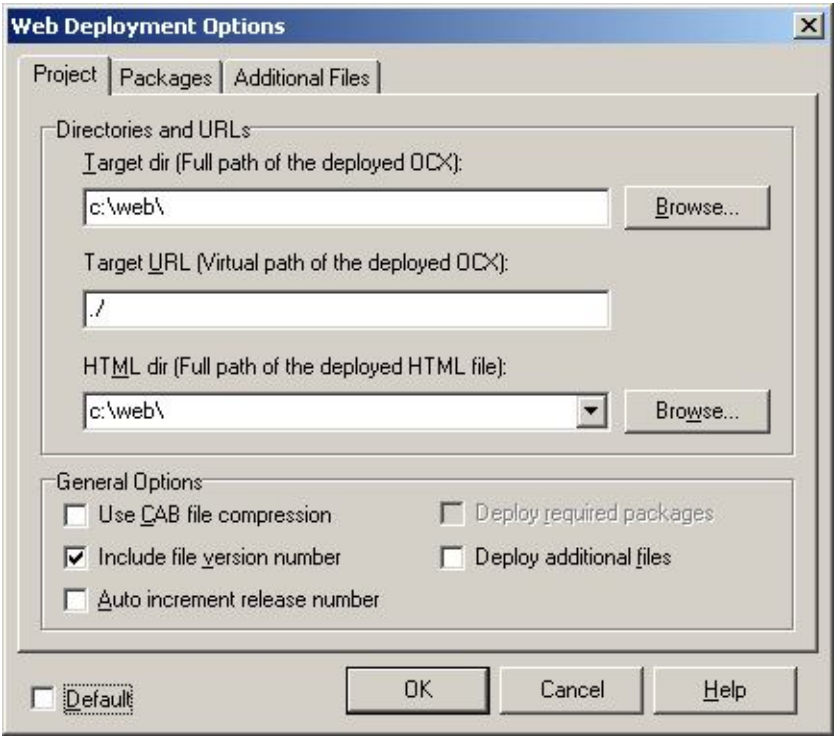
Əgər yaratdığımız OCX faylını lisenziyalı satmaq istəyiriksə, pəncərədən Make Control Licenzed variantını işarələyin. Bu qayda ilə OCX faylına aid. Lic genişlənməli bir fayl yaranır. Bu lisenziya faylı olmadan OCX faylını işlətmək mümkün olmayacaq.
Pəncərədəki Include About the variantını seçsək OCX faylına About box pəncərəsi qoşulacaqdır.
Include version information variantını seçsək ocx faylından Visual Basic mühitində istifadə etmək mümkündür. OK düyməsinə basaraq Active form yaradırıq.



Active form üzərində istədiyiniz komponentləri yerləşdirib və hadisələrinə lazımi proqram kodlarını yaza bilərik.

8.2 Web sehifesini yerleshdirme

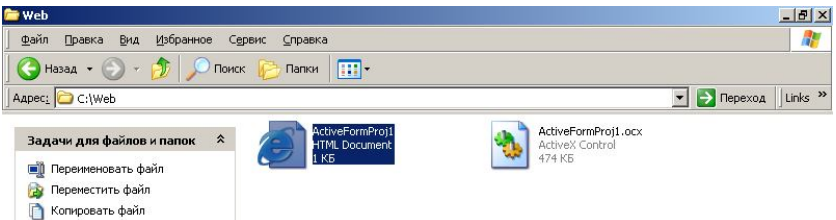
Formani yaradib proqram kodunu yazdiqdan sonra onu Web sehifesinde yerleshdirmek mumkundur. Bu ishde de Delphi bize komek edecekdir.
Project ▸ Web Deployment Options emri ile ashagidaki pencereni achiriq:



Pəncərədəki **Target dir** və **HTML dir** sətirlərinə yaradılacaq OCX faylının və onun yerləşəcəyi Web səhifəsinin yolunu yazın. Nümunənin kompyuterdəki Web kataloquna yazılması üçün c:/Web yazın. Target Url sətirinə isə ocx faylının Internedəki ünvanını yaza bilərsiniz. Əgər Internet və ya Intranet şəbəkələrində html faylı ilə ocx faylini eyni kataloqa yazmaq isteyirsinizse unvan olaraq ./ yaza bilersiniz.

Penceredeki **Use CAB file compression** variantene isharelesek, OCX fayli CAB fayli halina getirilerek Internetden yuklenmesi sixishdirilaraq daha da asanlashir.

Bu ishlerden sonra Project menyusundan Web Deploy sechsek hem OCX fayli hem de OCX-e esaslanan html fayli sechdiyimiz faylda qeyd olunacaqdir.

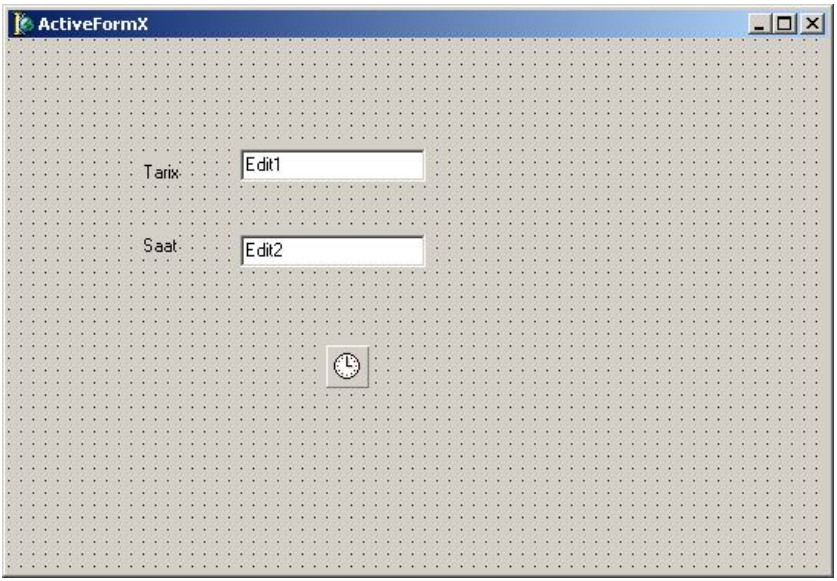


Yerləşdirilən Veb səhifənin koduna baxaraq Active Formun Veb səhifəsinə hansı kodla daxil olduğuna baxa bilərik.

«HTML»
«HI» Delphi 6

«HTML»

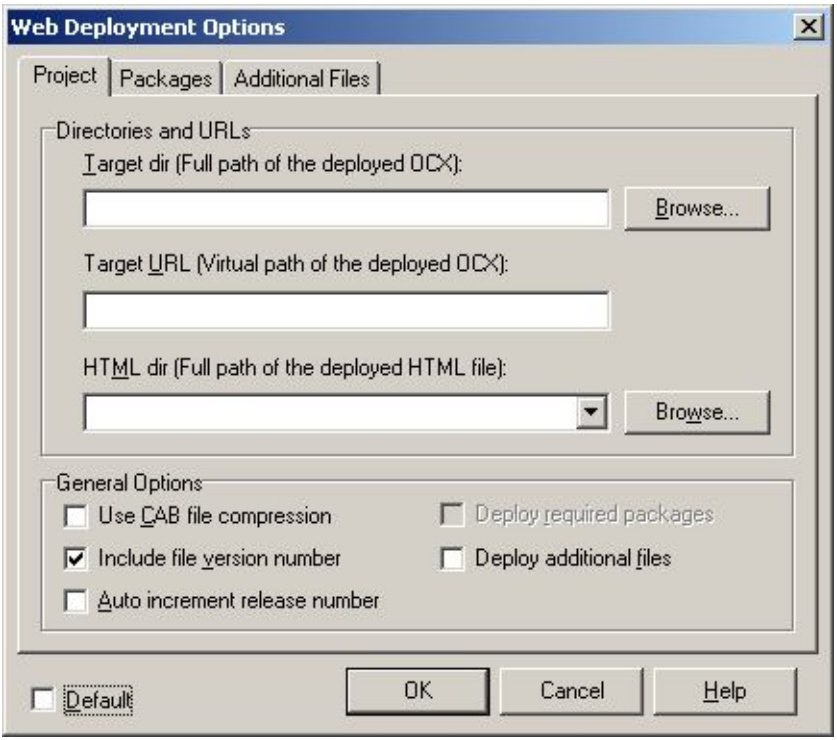
Bu kodlari <OBJECT…> </OBJECT> setirleri arasinda qalan hisseni öch WEB sehifemize kochurub istifade ede bilerik.
Numune: Timer vasitesi ile saat ve tarixi ekrana chixaran bir ActiveForm yaradaq.
File ▸ New ▸ Other ▸ ActiveX ▸ ActiveForm əmrlərinin köməyi ilə yeni bir ActiveForm yaradıb üzərində aşağıdakı işləri görək:



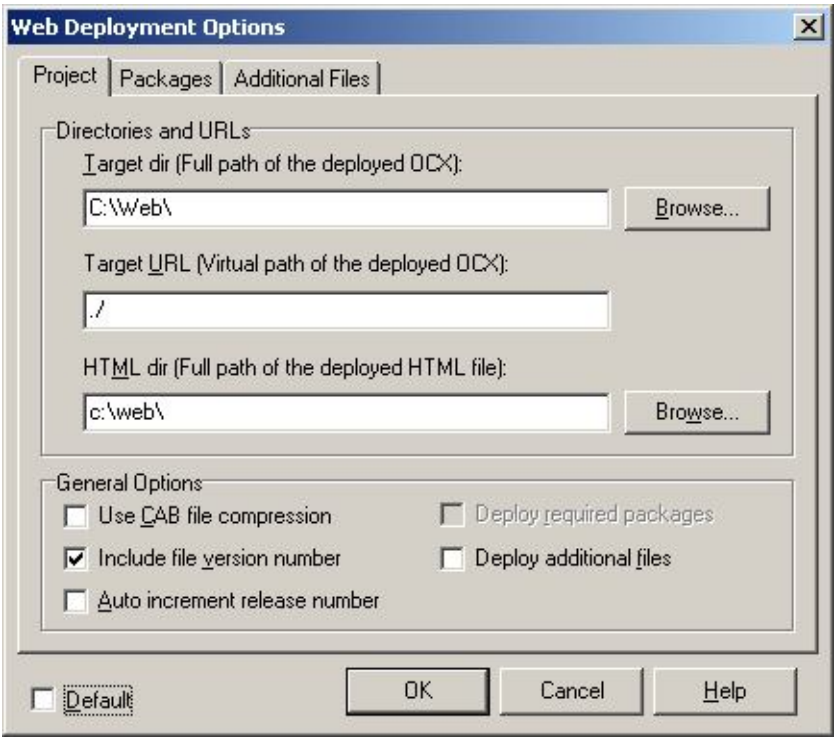
On Timer hadisəsinə aşağıdakı proqram kodunu yazaq.

```
procedure TActiveFormX.Timer1Timer(Sender: TObject);
begin
Edit1.Text:=DateToStr(Date);
Edit2.Text:=TimeToStr(Time);
end;
```

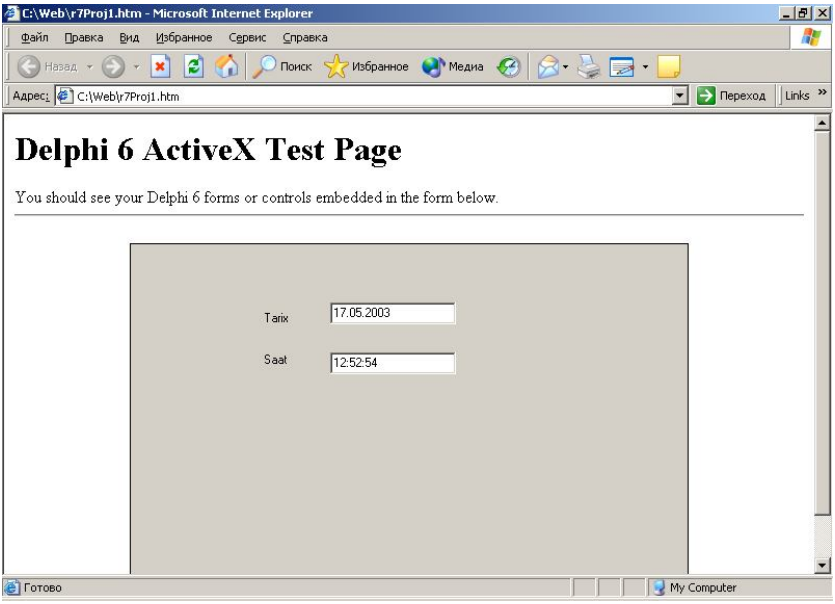
ProjectWeb Deployment Options emrleri ile ashagidaki pencereni achaq ve lazim olan kataloqlari sechek:



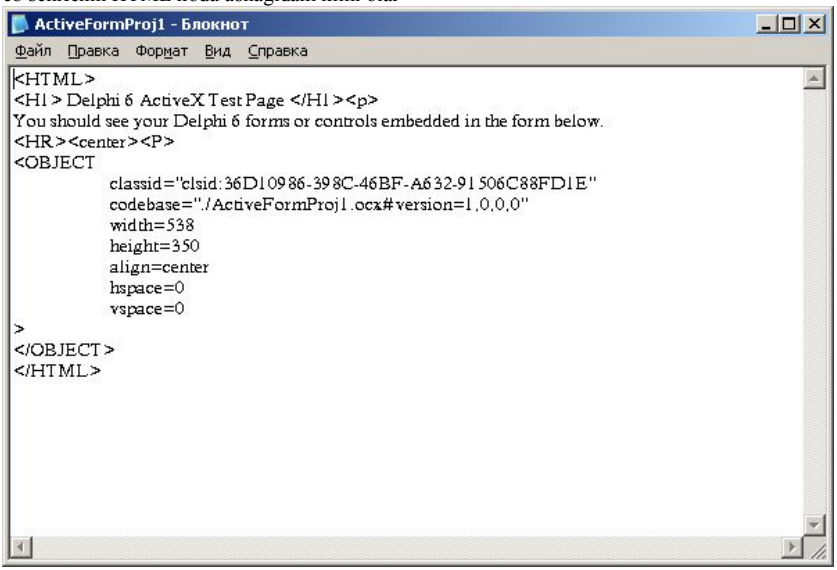
Bu penceredeki setirlerde OCX faylinin ve yaradilacaq html faylinin yolunu gosterek.



Burada Target-dir və HTML-dir eyni olarsa, kataloq olaraq Target Url xanasına «./» ifadəsi yazılır. OK düyməsini basdıqdan sonra Project Web deploy əmrləri ilə Web səhifəsinin və OCX faylının istədiyiniz kataloqda yerləşməsini yoxlayırıq.
HTML faylının yerləşdiyi kataloqa keçib, yaradılmış Web səhifəsini aktiv hala gətirib çalışdıraq.
Web səhifəsini çalışdırdıqdan sonra aşağıdakı nəticənin alındığını görəcəyik:



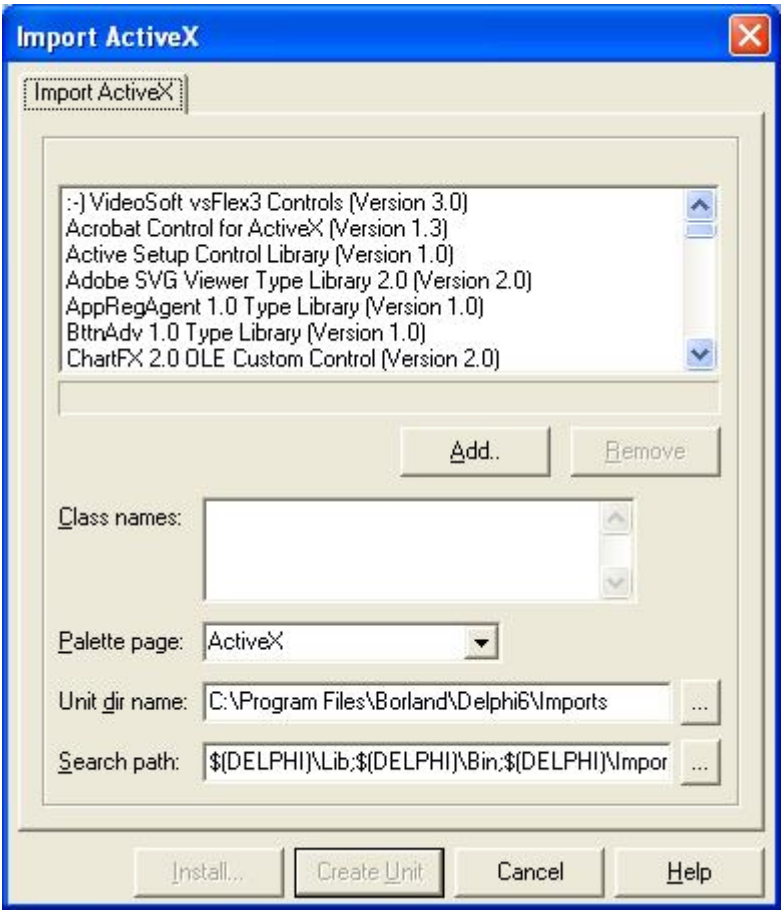
Web sehifenin HTML kodu ashagidaki kimi olar



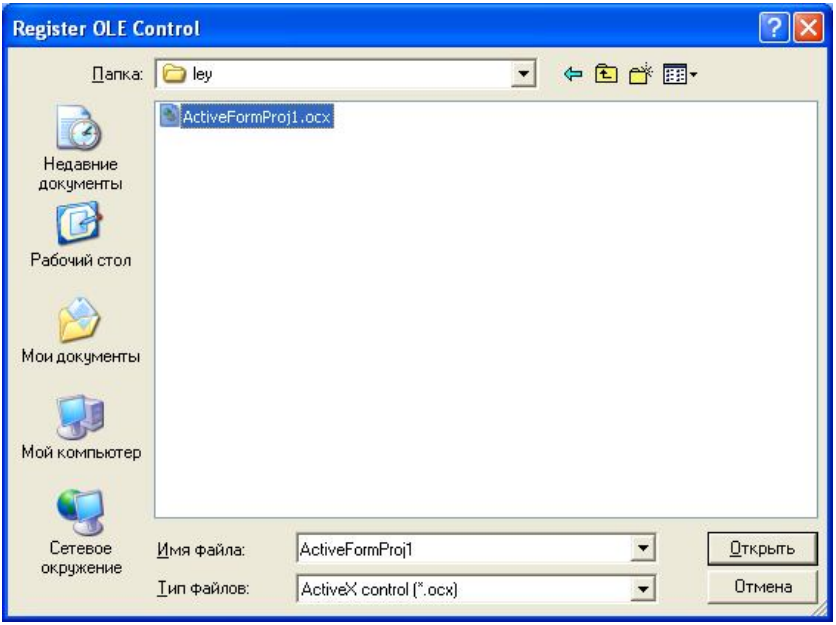
8.3.ActiveForm-un komponentler palitrasina yerleshdirilmesi

Yaradilmish ActiveX-i komponentler palitrasina yerleshdiririk ondan diger komponentler kimi istifade etmek olar.

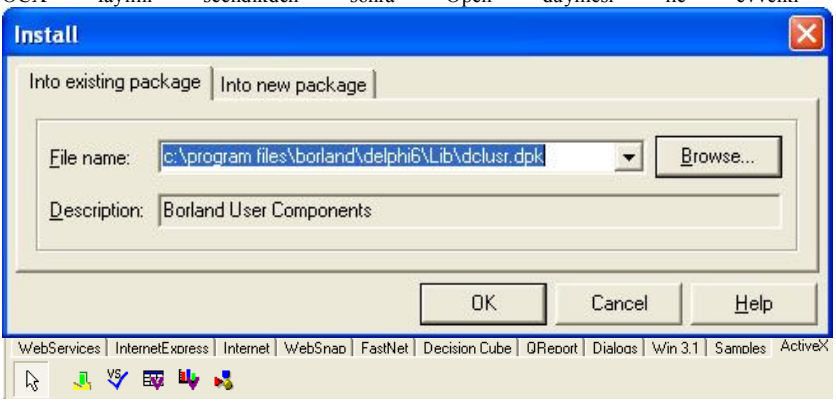
Component ▶ Import ActiveXControl emri ile ashagidaki pencereni achiriq:



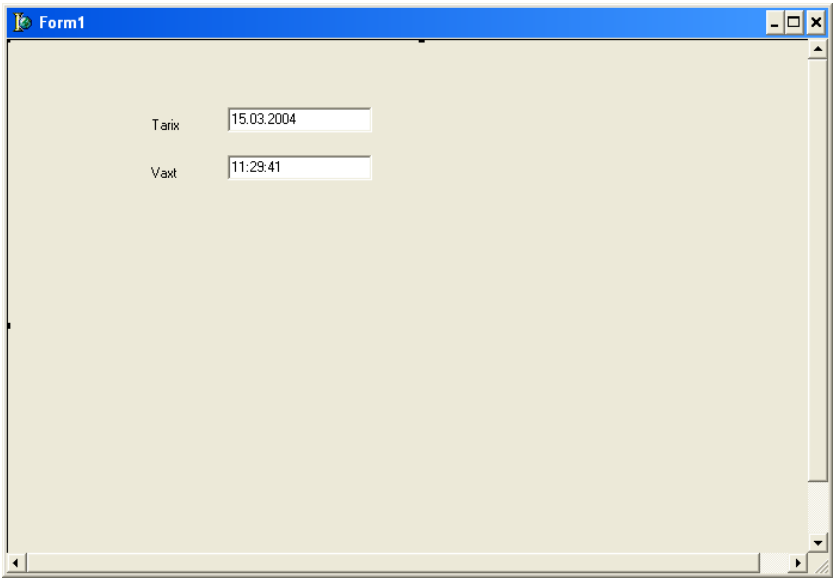
Penceredeki Add duymesini vuraraq, OCX faylinin yerleshdiyi kataloqa kechek.



OCX faylini sechdikden sonra Open duymesi ile evvelki



Bundan sonra yeni komponentimiz komponentler palitrasinda oz yerini alacaqdir.
Bu komponentden de diger komponentler kimi istifade etmek olar.


Form1
Tarix
15.03.2004
Vaxt
11:29:41