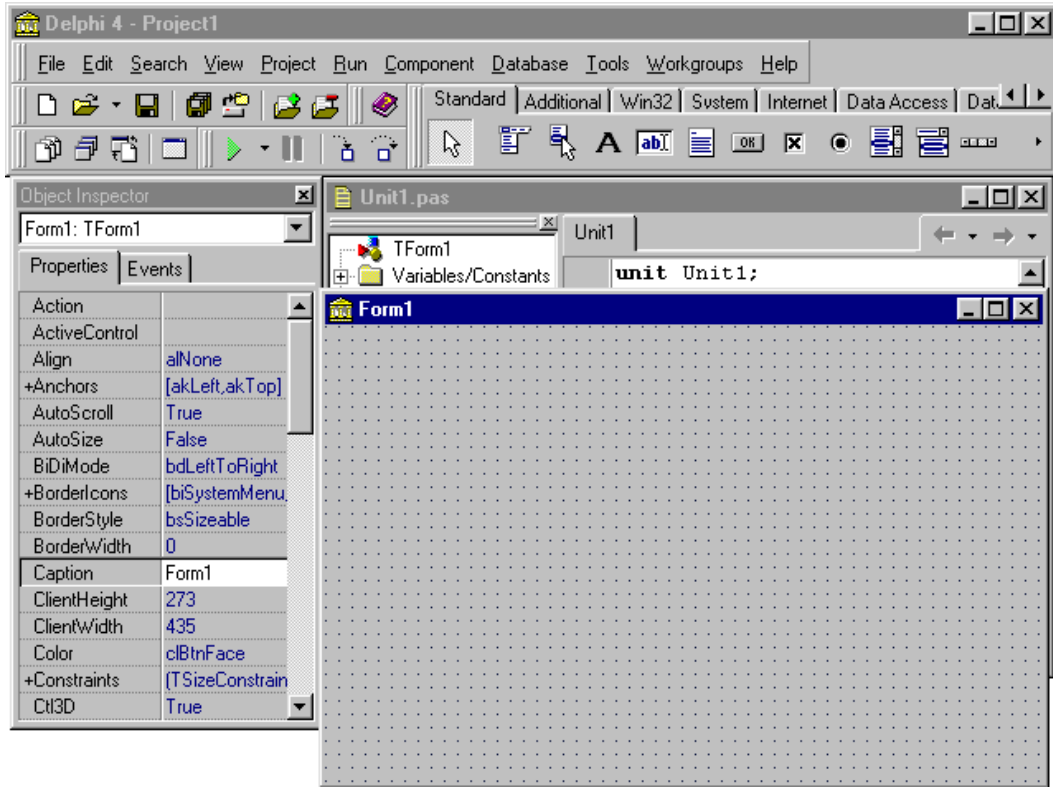


BÖLÜM 1

DELPHI'YE GİRİŞ

Delphi, Borland'ın derleyici teknolojisindeki en iyi yanlarını alan, bunları Borland'ın veri tabanı teknolojisindeki en iyi yanlarıyla birleştiren ve yeni görsel programlama araçlarını kullanan yeni, özgün bir üründür. Bu özellikleriyle karma bir ürün olan Delphi, programcılara hem standart uygulamaları hem de istemci/sunucu uygulamalarını hızlı hazırlama olanağını eşi görülmemiş bir biçimde sağlamaktadır. Başka bir deyişle Delphi, Borland'ın derleyici ve veri tabanı teknolojilerini güçlü bir araçta bir araya getirmek amacıyla görsel araçlardan yararlanan bir üründür.

Başlat menüsünden Delphi'yi temsil eden komutu verirseniz, Delphi çalıştırılmak üzere hard diskten belleğe okunmaya başlanır. Delphi'nin belleğe okunması işlemi tamamlandınca Şekil 1.1'deki gibi bir ekran görüntüsü ile karşılaşacaksınız.



Şekil 1.1. Varsayılan Delphi Ekranı

Delphi programlama ortamının kendine özgü yanlarını tanıtmaya başlamadan önce, Delphi'ye biçim veren birkaç genel niteliğe değinmek isterim. Bu ayırdedici niteliklerden bir çoğunun Delphi'deki çalışmanızın genel yapısının tanımladığını

göreceksiniz. Delphi ortamının görünümü diğer Windows uygulamalarına göre farklıdır. Örneğin Borland Pascal for Windows 7.0, Borland C++ 4.0 ve Word gibi programlar MDI uygulamalarıdır ve bu nedenle Delphi'ye göre farklı bir görünüme sahiptirler. MDI uygulamaları Multiple Document Interface (Çoklu Belge Arabirimi) tanımına uyarlar. Buna göre daha küçük olan bir dizi pencere, büyük bir pencerenin içine yerleştirilir.

Delphi ise başka bir tanıma, Single Document Interface'e (SDI) uyar. Delphi için SDI'nin seçilmiş olmasının nedeni, bunun Windows 95'te kullanılan uygulama modeline bağlı kalmasıdır.

SDI, uygulamaların daha çok nesneye dayalı ve doküman merkezli olmalarını sağlar. Önceden özellikleri belirlenmiş olan bir dizi pencereyi daha büyük tek bir pencere içinde sınırlamaktansa, SDI tüm biçim ve boyutlardaki pencerelerin bir uygulamaya eklenmesine izin verir. Böylelikle uygulamalar, kolaylıkla kullanıcılar yada üçüncü parti programcılar tarafından genişletilebilir.

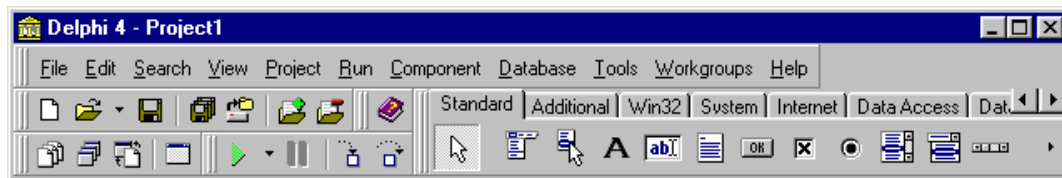
Delphi gibi SDI uygulamalarını kullanıyorsanız, işinize başlamadan önce diğer uygulamaları minimize etmek kullanımı basitleştirir. Sonra başka bir uygulamaya geçmek isterseniz Delphi'nin ana menüsü üzerindeki simge durumuna küçültme düğmesine tıklayıp görsel programlama ortamındaki herhangi bir pencereyi küçültüp diğer uygulamalar için yer açabilirsiniz. Delphi, aynı modeli sizin kendi uygulamalarınıza da sunar.

1.1. Delphi Ekranı

Delphi ile çalışırken klasik dillerde olduğu gibi sadece programı yazdığınız editörle baş başa kalmayacaksınız. Delphi'nin visual yapısından dolayı masa üstünde birkaç pencere ile programınızı oluşturup takip etmeniz gerekir. Bu pencereleri ve ne olduklarını sırasıyla inceleyelim.

1.1.1. Delphi'nin Ana Formu

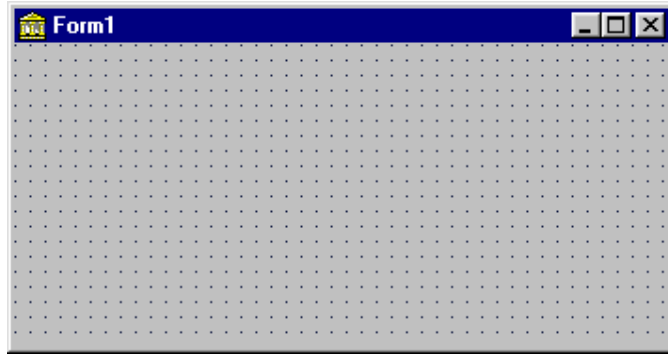
İlk olarak Delphi çalıştırıldığında Delphi'nin kendine ait olan Şekil 1.2'deki



formu görülür. Görüldüğü gibi Delphi formu diğer bazı programlardan farklı olarak sadece menülerden ve başlık çubuğundan oluşur.

1.1.2. Programcı Formu (Form Designer)

Delphi masaüstünde bulacağımız diğer önemli bir form da aşağıdaki gibi bir form penceresidir.



Şekil 1.3. Form Penceresi

Bu form gerçekte Delphi'ye ait değil, yapacağımız programa ait bir formdur. Bu form programa ait kullanıcı arabiriminin oluşturulduğu yerdir. Ve burada oluşturduğumuz formun görüntüsü program çalıştığında da (bazı kontroller hariç) aynen görülecektir. Delphi'i visual yapan özelliklerden biri bu özelliğidir. Diğer dillerde programın ekran görüntüsü program çalışması esnasında oluşturulurken Delphi'de bu iş tasarım esnasında da yapılabilir. Böylece programınızda ekranı düzenleyecek sıkıcı kodlar yazmaktansa bu işi kolayca ve görerek yapabilmekteyiz. Programda birden fazla form da bulunabilir.

Programa yeni bir form eklemek için **Form/New** menüsünden **Form** seçeneği kullanılabilir. Her form kendi ismiyle ve **DFM** uzantısı ile kaydedilir.

1.1.3. Component Palette

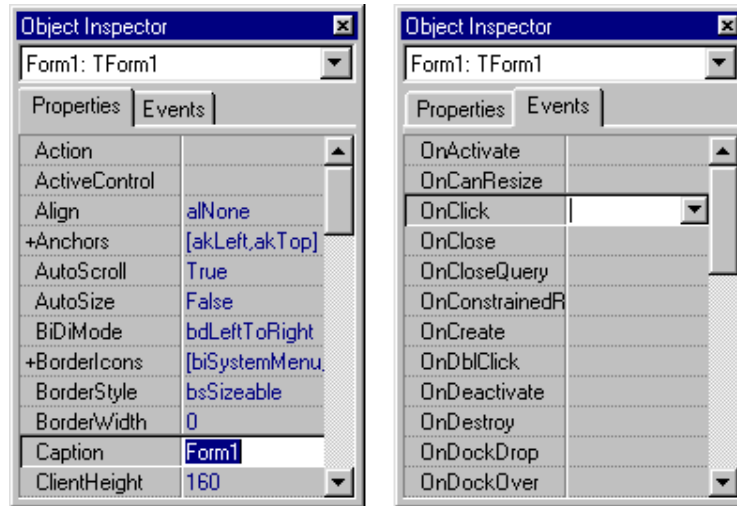
Component Palette, programcı formuna (Form Designer) yerleştirmek istediğiniz kontrol araçlarını seçmenizi sağlar. Component Palette'i kullanmak için

hazır araçlardan birini seçtikten sonra form üzerinde herhangi bir yere tıkladığımızda seçtiğimiz araç form (Form Designer) üzerinde fareyle işleyebileceğimiz bir nesne olarak belirecektir.

Component Palette sayfalamaya mecazını kullanır. Paletin tabanı boyunca **Standart, Additional, Dialogs** gibi sekmeler bulunmaktadır. Eğer bu sekmelerden birini tıklarsanız, Component Palette'in bir sonraki sayfasına geçebilirsiniz. Bu tip sayfalamaya mecazı Delphi'de geniş kullanım alanı bulmaktadır. Ayrıca, isterseniz kendi uygulamalarınızda da kolayca kullanabilirsiniz.

1.1.4. Object Inspector (Özellikler ve Olaylar Penceresi)

Object Inspector penceresi, form üzerine component paletten yerleştirilen her kontrolün özellik ve olaylarını belirler. Object Inspector'daki bilgi form üzerinde seçili olan bileşene bağlı olarak değişir. Burada önemli olan nokta, her bileşenin aslında bir nesne olması ve bu nesnenin ayırdedici niteliklerinin Object Inspector kullanılarak değiştirilebilmesidir. Object Inspector penceresi, form üzerinden bir kontrol seçildikten sonra **F11** tuşu veya **View/Object Inspector** menü seçeneği ile görüntülenir.



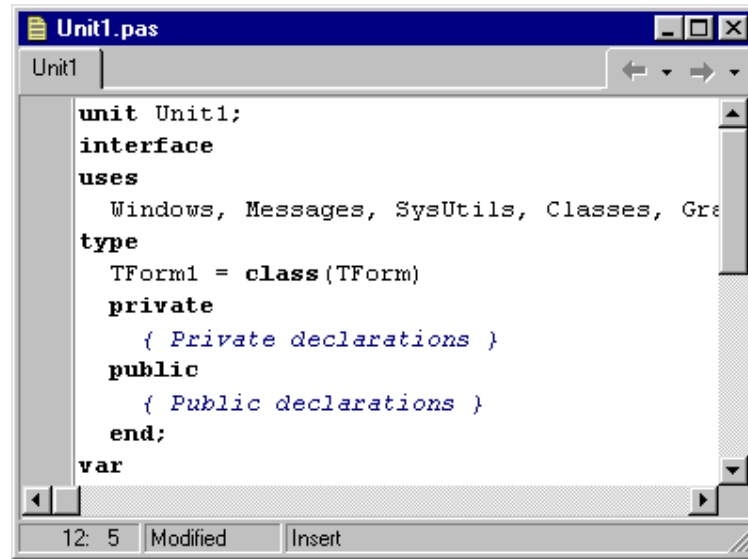
Şekil 1.5. Object Inspector Penceresi

Yukarıdaki pencerelerden ikisi de Object Inspector penceresidir. Birinci pencerede form üzerine alınan elemanın tasarım zamanı özellikleri değiştirilirken, ikinci pencerede ise form üzerindeki elemana ait olaylar (**events**) görüntülenir.

Visual dillerin diğer dillerden bir farkı da budur. Program çalıştırılmadan önce bu özellikler hiçbir koda gerek kalmadan değiştirilebilmektedir. Pencerenin ikinci hali ise o kontrole ait olayları belirlemek için kullanılır. Kontrole ait olayların (events) listesi pencerede görülmektedir. Bu pencerenin karşısındaki kutu çift tıklanarak varsayılan olayın yazılmasını veya kendiniz bir alt program ismi yazarak, o olay gerçekleştiğinde seçtiğiniz olayın çalışmasını sağlayabilirsiniz. Olayların ne olduğunu daha sonraki bölümlerde göreceğiz.

1.1.5. Kod Penceresi

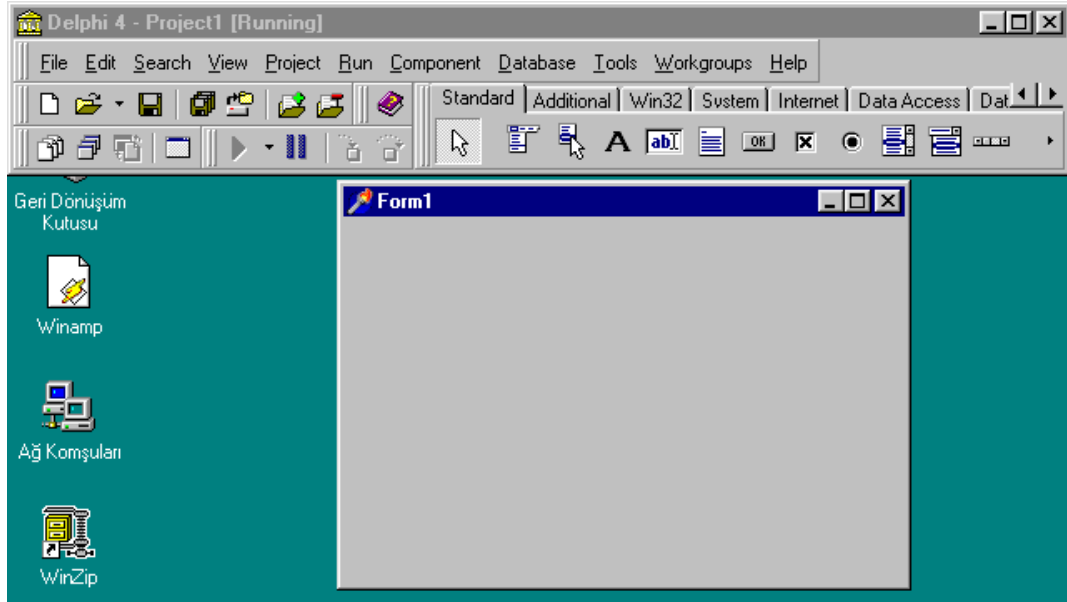
Şekil 1.1’de verilen, Delphi’nin ilk defa çalıştırıldığında ekran görüntüsünden de tespit etmiş olabileceğiniz gibi Delphi başlatıldığında geçici adı **Project1** olan bir proje otomatik olarak hazırlanmakta ve yine bu proje için **Form1** adında bir form otomatik olarak hazırlanmaktadır. Bunun dışında otomatik olarak hazırlanan proje için otomatik olarak **unit1.pas** adında, Pascal program kodu içeren bir unit hazırlanmakta ve projeye dahil edilmektedir. Projenize dahil edilen editör benzeri bu forma program kodları yazabilirsiniz. Başlangıçta Pascal program kodu içeren bu pencere Form1’in altında kaldığı için masa üstünde görünmemektedir. Aşağıdaki pencere (Şekil 1.6.) Form1’e ait kod penceresidir.



Şekil 1.6. Kod Penceresi

Bu penceredeki bütün program satırları otomatik olarak hazırlanmaktadır. Bu program satırlarını yakından inceleyecek olursak, bu satırların bazılarının en az özelliğe sahip olan Pascal programında olan program satırları olduğunu görürüz. Örneğin bütün Pascal programlarının başına program dahilinde kullanılacak unitler **Uses** deęimi ile programa dahil edilmektedir. Bütün Pascal programlarının en son satırında **End** deyiminin bulunması bir zorunluluktur.

Delphi işinizi kolaylaştırmak için her projede standart olarak yer alan tanımlamaları ve program satırlarını otomatik olarak hazırlamaktadır. Bu pencerede yazılan kodlar PAS uzantılı dosyalara kaydedilecektir. Delphi'nin başlatılması sırasında otomatik olarak hazırlanan proje üzerinde herhangi bir işlem yapmadan projeyi çalıştıracak olursanız, otomatik olarak hazırlanan form1 penceresi aşağıdaki (Şekil 1.7) gibi ekrana gelir. Üzerinde çalıştığınız Delphi projesini çalıştırmak için **Run** menüsündeki **Run** komutunu verebilir veya doğrudan **F9** tuşuna basabilirsiniz.



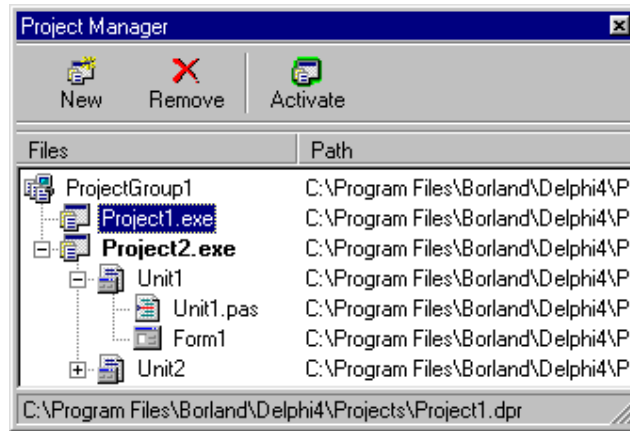
Şekil 1.7. Çalıştırılmış (RUN) bir proje

Proje çalıştırıldığında Object Inspector penceresi otomatik olarak masa üstünden kaldırılır. Delphi projesinin çalışmasını sona erdirmek için Form1 penceresinin denetim düğmesine çift tıklama yapabilir veya **Run** menüsünden **Program Reset** komutunu verebilirsiniz.

1.2. Delphi Projelerinin Bileşenleri

Yukarıdaki sayfalarda anlatıldığı gibi Delphi programlarının veya projelerinin en temel bileşenleri formlardır. Ancak formlar Delphi programlarının tek bileşeni değildir. Delphi'nin başlatılması sırasında karşılaşılan ekran görüntülerinde üzerinde çalışılan proje ile ilgili olarak programcıya bilgi veren herhangi bir pencere ekranda yoktu. Eğer üzerinde çalıştığımız Delphi projesi hakkında bilgi edinmek istiyorsanız varsayım olarak ekranda gösterilmeyen **Project Manager** penceresini ekrana getirmeniz gerekir. Project Manager penceresini ekrana getirmek için **View** menüsündeki **Project Manager** komutundan yararlanılmaktadır.

View menüsünden Project Manager komutunu verirseniz o sırada üzerinde çalıştığımız proje hakkında bilgi içeren Şekil 1.8'deki gibi bir pencere açılır. Bu pencerede üzerinde çalıştığım Project1 ve Project2 projeleri ile bu projelere ait form ve unitler ağaç yapısı şeklinde gösterilmektedir. Yeni bir proje eklemek için pencerenin üst kısmındaki **New** düğmesine basmanız yeterli. Eğer mevcut projelerden birini silmek istiyorsanız **Remove** düğmesine basmalısınız. **Activate** düğmesiyle ise mevcut projelerinizden istediğinizi seçerek üzerinde çalışma yapabilirsiniz.



Şekil 1.8. Project Manager Penceresi

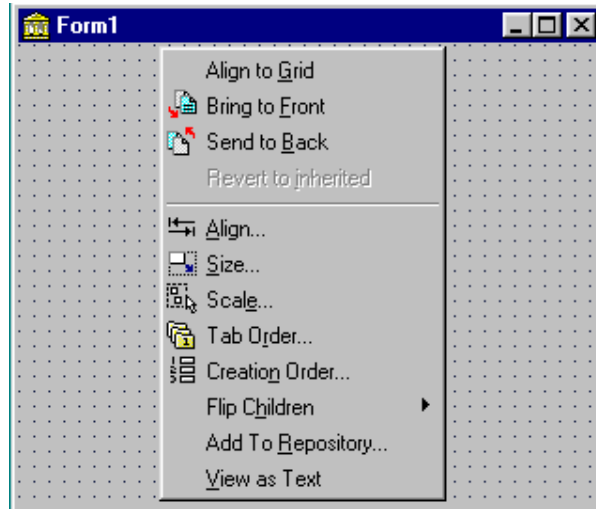
Delphi projelerinde formlardan başka birde PAS uzantılı Pascal program kodlarını içeren dosyalar bulunmaktadır. PAS uzantılı program kodu içeren dosyalara Delphi dahilinde **Unit** adı verilmektedir. Delphi projelerinde birden fazla PAS uzantılı Unit dosyası veya DFM uzantılı form dosyaları bulunabilir. Delphi ile hazırlanan projeler DPR uzantısı ile kaydedilmektedir. Örneğin 'Project1.dpr' gibi.

1.3. Sağ Fare Tuşunun Kullanımı

Windows altında çalışan programların çoğu sağ fare tuşunu destekler. Böylece sağ fare tuşu ile açılan popup menüler aracılığı ile yapılacak işlemlere oldukça hızlı bir erişim sağlanmış olur. Burada Delphi 4'te sağ fare tuşunun kullanıldığı yere göre aktif olan menü seçenekleri açıklanacaktır.

1.3.1. Form Üzerinde Sağ Fare Tuşu

Aşağıda, Şekil 1.9'da görüldüğü gibi form üzerinde iken farenin sağ tuşuna basıldığında bir popup menü açılmaktadır. Buradaki menü seçenekleri **Edit** menüsünün seçeneklerinin aynısıdır. Menüdeki komutları kısaca açıklayalım:

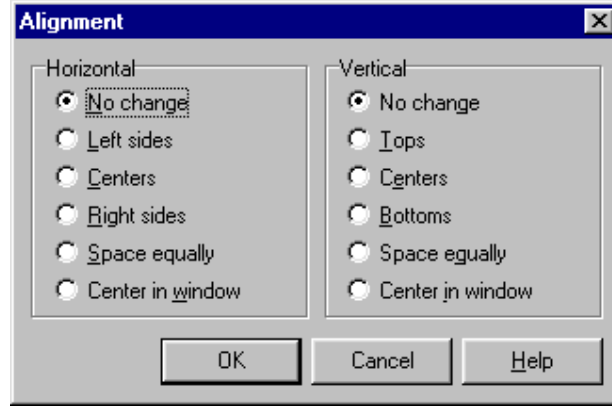


Şekil 1.9. Form üzerinde sağ fare tuşu

Align to Grid: Seçilen kontrolün koordinatları form üzerindeki gridlere denk gelmiyorsa en yakın grid noktasına denk gelecek şekilde taşınır.

Bring to Front, Send to Back: Üst üste gelen kontrollerden birini öne alırken diğerini arkaya atar.

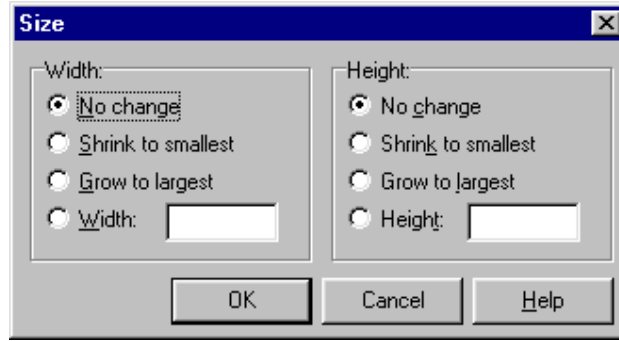
Align: Seçilen kontrolleri aynı hizaya getirmeye yarar. Align menü seçeneği şekil 1.10'deki pencereyi görüntüler.



Şekil 1.10. Alignment Penceresi

Alignment penceresi yardımıyla form üzerindeki birden fazla elemanı bazı ayarlamalara tabi tutmak mümkün olmaktadır. Buradaki radio düğmeleri kullanılarak seçili olan elemanlar sağdan, soldan, üstten, merkezden, alttan aynı hizaya getirilebilir. Hatta seçili olan elemanlar formun tam ortasına alınabilirler.

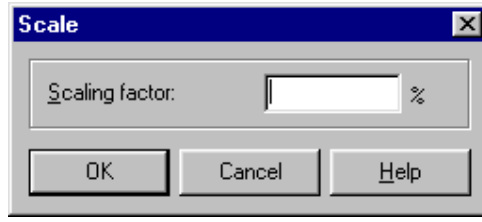
Size: Seçilen kontrollerin hepsinin aynı boya veya aynı yüksekliğe sahip olmasını sağlar. Açılan pencerede yapılan seçime göre hepsi en küçüğünün veya en büyüğünün boyuna yada girilen boya getirilmesi sağlanır.



Şekil 1.11. Size Penceresi

Size menü seçeneği ile Şekil 1.11'deki pencere görüntülenir. Burada **No Change** seçeneği ile seçili elemanlar üzerinde herhangi bir işlem yapılmazken, **Shrink to Smalest** seçeneğiyle seçili olan elemanlar en küçük boyutlu elemanın boyutuna, **Grow to Largest** ile de en büyük elemanın boyutuna getirilir. **Width, Height** seçenekleriyle de seçili olan elemanların genişlik ve uzunlukları değiştirilir.

Scale: Form üzerindeki kontrolleri, seçilen oranda büyültüp küçültmeye yarar. Bir seferde %25 ile %400 oranında ölçeklendirilebilir.



Şekil 1.12. Scale Penceresi

Tab Order: Form üzerindeki kontrollerin tab sırasını değiştirmeye yarar. Tab sırası kullanıcının kontroller arasında **Tab** tuşu ile geçiş yaparken sırası ile hangi kontrollere geçeceğini belirler.

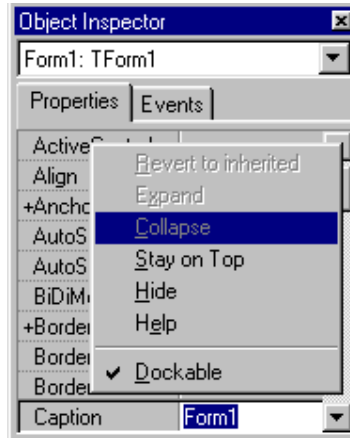
Creation Order: Uygulamanın oluşturduğu Visual olmayan elemanların sırası bu seçenek aracılığı ile düzenlenir.

Flip Children: Form üzerindeki kontrollerin sırasını sağdakini sola, soldakini sağa alarak değiştirir. Kontrolleri aynada oluşacak görüntülerine göre sıralar diyebiliriz.

View As Text: Oluşturulan DFM dosyasını metin olarak görüntüler.

1.3.2. Object Inspector Penceresinde Sağ Fare Tuşu

Object Inspector penceresi üzerinde sağ fare tuşuna tıkladığımızda aşağıdaki Şekil 1.13'te görülen bir popup menü açılır. Bu menünün seçenekleri:



Şekil 1.13. Object Inspector Penceresi

Expand: Alt seçenekleri + ile temsil edilen özellikler için aktif hale gelir. Bu seçeneğin tıklanmasıyla alt seçenekler de görülür.

Collapse: Alt seçenekleri açılmış özellikler için aktif hale gelir. Bu seçeneğin kullanılmasıyla alt seçenekli özellikler + halinde temsil edilir.

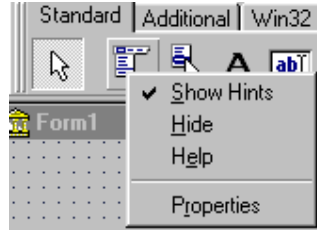
Stay On Top: Object Inspector penceresini daima en üstte tutar.

Hide: Object Inspector penceresini gizler.

Help: Pencereyle ilgili yardım dosyasını açar.

1.3.3. Component Palette Üzerinde Sağ Fare Tuşu

Component Palette üzerinde sağ fare tuşuna basılırsa Şekil 1.14'teki gibi bir popup menü açılır. Bu menünün seçenekleri:



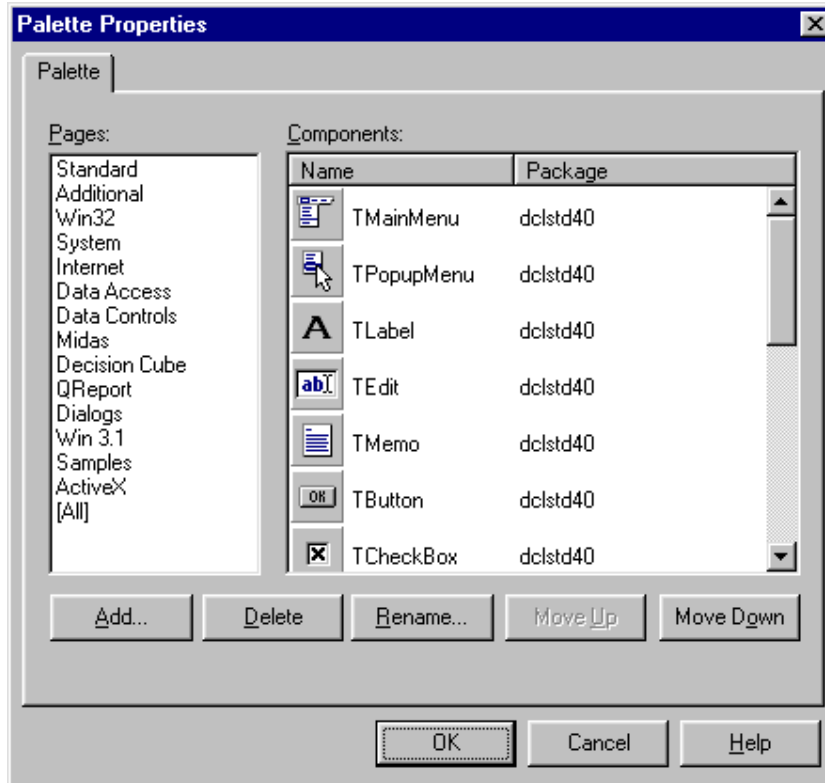
Şekil 1.14. Component Palet üzerinde sağ fare tuşu

Show Hints: Component Palette'de bulunan elemanların isimlerini görüntüler.

Hide: Component Palette'i saklar.

Help: Component Palette hakkında bilgi verir.

Properties: Componentlerle ilgili bazı ayarlamaların yapıldığı aşağıdaki pencereyi görüntüler.



Şekil 1.15. Palette Properties Penceresi

Bu penceredeki seçenekler kullanılarak istenilen elemanın yeri, adı ve bulunduğu sayfası değiştirilebilir.

BÖLÜM 2

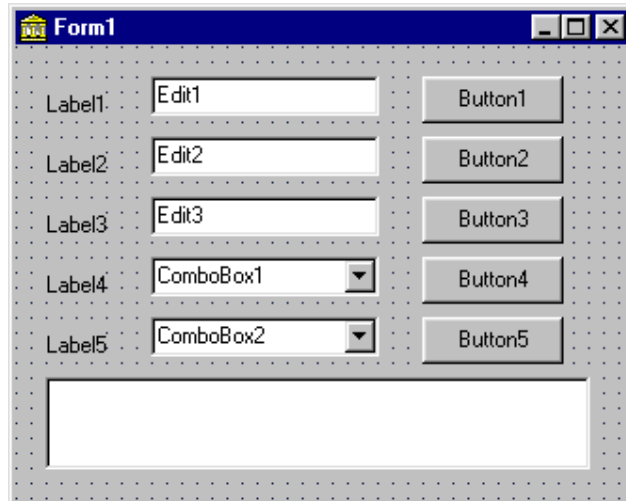
DEPHİ İLE VISUAL PROGRAMLAMAYA GİRİŞ

Delphi programlama dilinin en önemli özelliklerinden biri programın ekran tasarımını kodlama ile değil bir resim çiziyormuş gibi rahatça yapılabilmesidir. Şimdi ekran tasarımının yani form tasarımının nasıl yapıldığını görelim.

2.1. Form Tasarımı

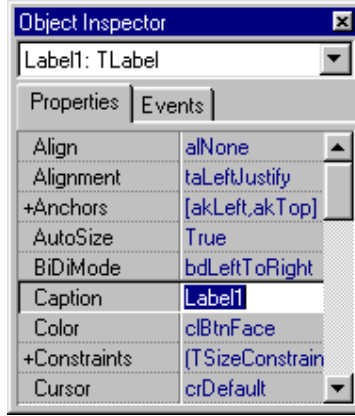
Programlarda kullanılabilecek standart işlemler birer kontrol olarak tasarlanmış ve programcının kullanımına sunulmuştur. Örneğin personel ile ilgili bilgilerin girileceği bir programda personelin adı, doğum yeri, doğum tarihi gibi bilgiler için birer boş kutu (buna Text kutusu yada Edit kutusu diyoruz), kullanıcının ne yapması gerektiğini belirten yazılar (Label kutusu), Mesleğini seçmesi için hazır olarak daha önceden girilmiş mesleklerden birini seçme imkanı sağlayacak aşağı doğru açılan bir liste (ComboBox), programlardan çıkmak için veya kaydetmek için gerekli komut düğmeleri (buna komut düğmesi veya Buton diyoruz), mevcut personellerin listesini gösterecek bir liste kutusu programda bulunabilecek seçeneklerdendir. İşte bu tip işleri yapmak için hazır kontrollerimiz bulunmaktadır. Bunları uygun şekillerde ekrana yerleştirerek kolayca programımızın ekranını tasarlayabiliriz.

Form üzerindeki kontrolleri oluşturmak için Toolbox kutusunda bulunan kontrollerden seçerek form üzerine çizmeniz gerekir. Şimdi Şekil 2.1'deki formu oluşturmaya çalışalım.



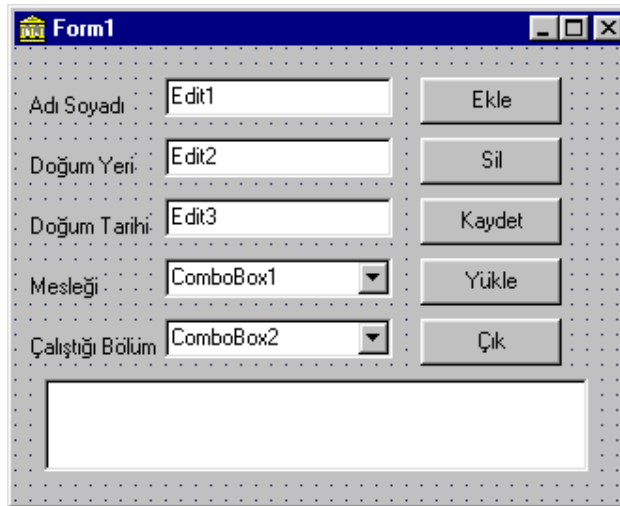
2.2. Object Inspector Penceresi ile Özellikleri Ayarlama

Formumuzu Şekil 2.1'deki gibi oluşturduktan sonra Label1, Label2, Button1 vb. yerine bunların açıklamalarını yazmamız gerekir. Bu işlemleri program kodları ile değil, tasarım zamanında yapabiliriz. Şimdi Label1 nesnesini seçelim ve Şekil 2.2'deki Object Inspector penceresine bakalım. Object Inspector penceresinin aktif (seçili) durumdaki nesnenin özelliklerini listelediğini biliyorduk.



Şekil 2.2. Object Inspector Penceresi

Yukardaki Object Inspector penceresinde, sol tarafta gördüğünüz **Align**, **Alignment** gibi kelimelere **properties** yani özellikler diyoruz. Sağ tarafındaki **alNone**, **taLeftJustify** gibi yazılarda bu özelliğin şu anki değerini göstermektedir. Her nesnenin farklı ve ortak özellikleri vardır. Şimdi bu özelliklerin etkisini görmek için **Caption** yazan özelliğin karşısındaki **Label1** kelimesi yerine Adı Soyadı kelimesini yazın, aynı

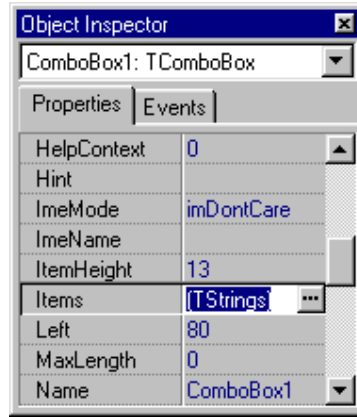
The image shows a window titled 'Form1' with a standard Windows-style title bar. The form contains several input fields and buttons. On the left, there are five labels with corresponding input fields: 'Adı Soyadı' with 'Edit1', 'Doğum Yeri' with 'Edit2', 'Doğum Tarihi' with 'Edit3', 'Mesleği' with 'ComboBox1', and 'Çalıştığı Bölüm' with 'ComboBox2'. To the right of these fields are five buttons: 'Ekle', 'Sil', 'Kaydet', 'Yükle', and 'Çık'. At the bottom of the form is a large empty rectangular box.

Şekil 2.3. Tasarlanmış Form

şeyin formunuzun üzerindeki Label1 üzerine de yazıldığını göreceksiniz. İşte Caption özelliği ile kutuların önündeki Label'lere gerekli açıklamalar bu şekilde yazdırılacaktır.

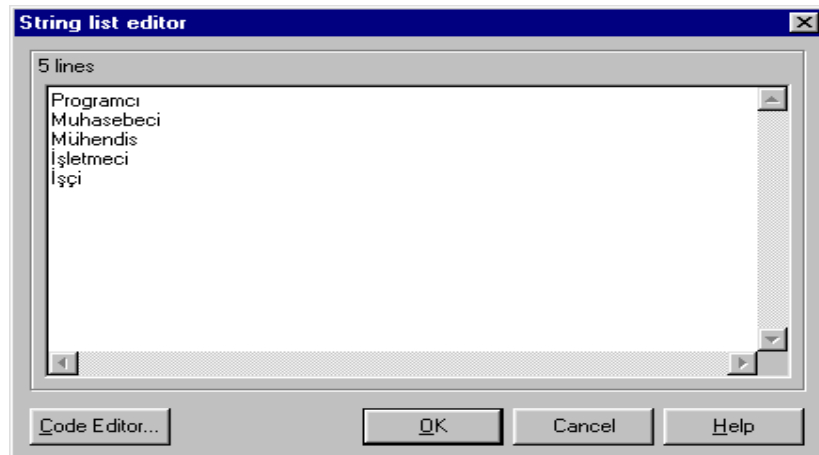
Bu özelliği, form üzerindeki diğer Label, Button ve Forma da uyguladıktan sonra Şekil 2.3'de verilen ekran görüntüsünü aldım.

ComboBox kutusunda bir kaç tane meslek gösterilmesini sağlamamız gerekiyor. Bu işi **Caption** özelliği ile değil, ComboBox'un **Items** özelliği ile yapacağız. Bu kutuda şu mesleklerin bulunmasını isteyelim: Programcı, Muhasebeci, Mühendis, İşletmeci, işçi.



Şekil 2.4. Object Inspector Penceresi

Bunu yapmak için **ComboBox1** kontrolünü seçerek Object Inspector Pencersinden **Items** özelliğinin yanındaki üç noktalı düğmeye bastığımızda açılan Şekil 2.5'deki pencereye personele ait meslekleri girip **OK** düğmesine basalım.



Şekil 2.5. ComboBox Liste kutusuna değer girişi

Aynı şeyi ComboBox2 içinde tekrarlayalım. Satış, Pazarlama, Teknik Servis ve Halkla İlişkiler bilgilerinin gösterilmesini sağlayalım.

2.3. Programı Çalıştırma

Artık programımızın ekran tasarımı hazır. Henüz hiçbir kod yazmadan programımızı çalıştıralım. Programı çalıştırmak için Run menüsünden **Run** seçeneğini kullanabilirsiniz veya doğrudan **F9** tuşuna basabilirsiniz. Aşağıdaki (Şekil 2.6) ekran görüntüsünü, programı çalıştırdıktan sonra aldım.

Şekil 2.6. Programın çalışması sırasında alınan ekran görüntüsü

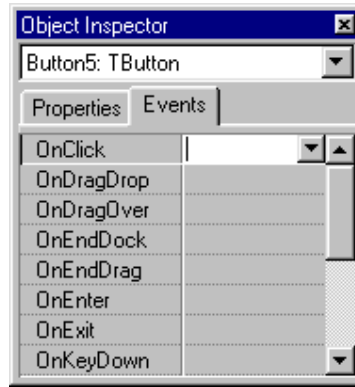
Artık tasarım ortamında değil çalışma ortamındasınız. Şimdi programdaki kutulara yazı yazmayı, birinden kopyalayıp diğerine yapıştırmayı ve formu boyutlandırmayı deneyin. Hiçbir kod olmadığı halde bu işlemleri yapabildiğinizi göreceksiniz. Ancak komut düğmeleri henüz çalışmıyor. Bunlar için kod yazmamız gerekiyor.

2.4. Kod Yazma

Önce en kolay olanını yazalım. Çıkış için gerekli olan kodu yazalım. Kodu herhangi bir yere değil o nesne üzerinde yapılabilecek değişik işlemlere göre ilgili yere

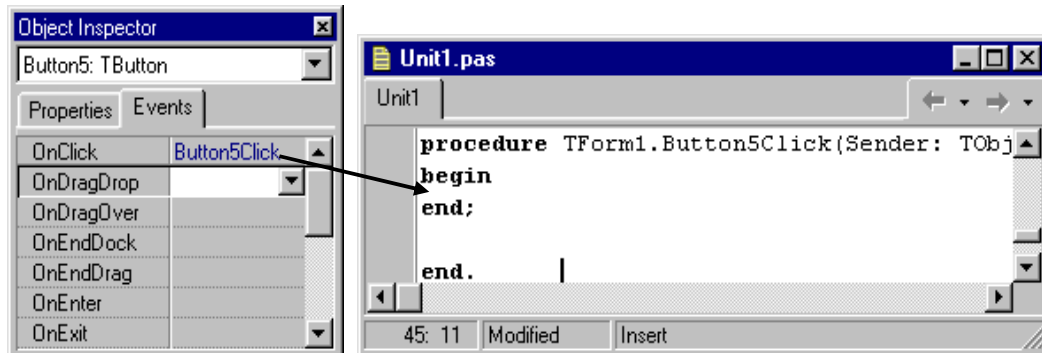
yazmamız gerekir. Örneğin mouse düğme üzerinden geçerken bir mesaj vermek istiyorsak, bunu o komut düğmesinin **OnMouseMove** olayına yazmamız gerekir. Biz, düğme tıklanınca programın sona ermesini istediğimiz için kodumuzu **OnClick** olayına yazacağız. Bu olay alt programına ulaşmak için iki yöntem var. Birincisi, varsayılan olaylara ulaşmak için o düğmeyi çift tıklamanız gerekir. Komut düğmesini çift tıklarsanız düğmenin **Click** olayına gidersiniz. Ancak diğer olaylara gitmek için yine **Object Inspector** penceresini kullanacağız.

Üzerine ÇIKIŞ yazdığımız düğmeyi seçerek **F11** tuşu ile Object Inspector penceresini açalım. Pencerenin üzerinde iki tab vardır. Bunlardan biri daha önce özellikleri değiştirmek için kullandığımız **Properties** tabı diğeri de kod yazmak için kullanacağımız **Events** tabıdır. Events tabını seçtiğimizde penceremiz aşağıdaki gibi olacaktır (Şekil 2.7).



Şekil 2.7.

Pencerde sol tarafta komut düğmesine ait olayları, sağ tarafta ise bu olayların olması halinde çalışacak program parçasının ismini göreceğiz. Henüz hiçbir olaya kod yazmadığımız için bütün olayların karşısı boş görünmektedir. **OnClick** olayının karşısındaki kutuya gidip fare ile iki defa tıkladığımızda bu olayın karşısında **Buton5Click** kelimesi yazacak ve aşağıdaki kod penceresi açılacaktır.

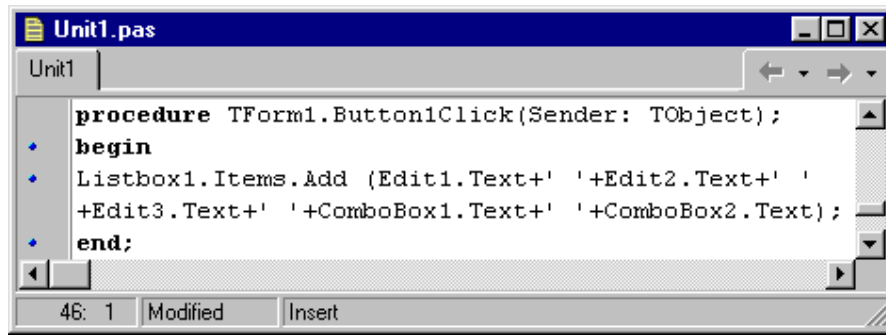


Şekil 2.8. Kod Penceresi

Şu anda Button5 düğmesinin Click olayına gerekli kodu yazabiliriz. **Begin-End;** bloğu arasına, programdan çıkmak için gerekli olan **Close** komutu yazılır. Artık programımız çalıştırıldığında, çıkış düğmesine basılması halinde programdan çıkılır.

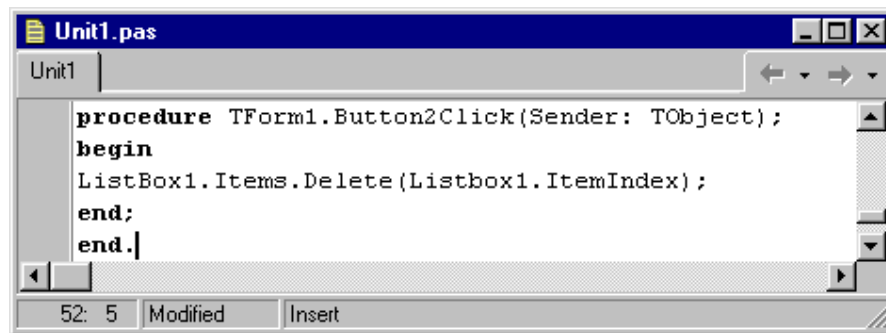
Şimdi de Ekle düğmesi seçildiğinde personele ait bilgileri listeye eklemek için gerekli kodu yazalım. Edit1 içine yazılan adı öğrenmek için Edit1 nesnesinin **Text** özelliğini kullanacağız. Label, Button ve Form gibi kullanıcının giriş yapamadığı nesnelerin, üzerlerindeki yazıyı Caption özelliği belirlerken, **Edit** kutusu gibi kullanıcının değiştirebildiği nesnelerin içerisindeki yazıyı ise **Text** özelliği belirler.

Edit kutusuna girilen ismi listeye eklemek için Liste kutusunun **Items.Add** özelliğini kullanacağız. Üzerine Ekle yazdığımız Button1 düğmesini çift tıklayarak, **Click** olayına ulaşarak gelen kod ekranının **Begin-End;** bloğu arasına aşağıdaki kodu tek satır olarak yazarsınız (Şekil 2.9). Artık **Ekle** düğmesi de çalışıyor.



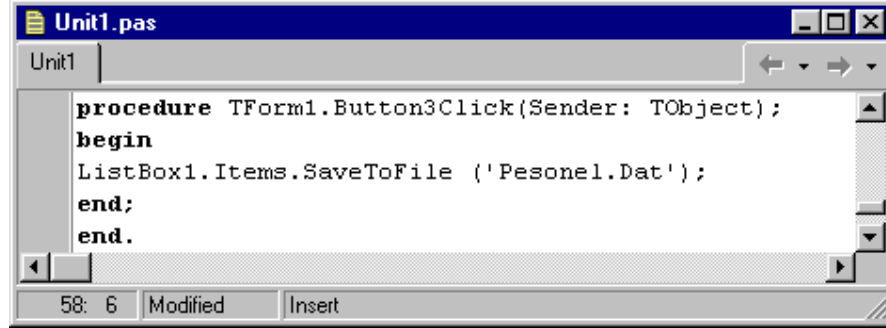
Şekil 2.9. Ekle düğmesine ait kod sayfası

Şimdi **Sil** düğmesi ile listeden seçilen personeli listeden çıkarmak isteyelim. Listedene bir elemanı silmek için Liste kutusunun **Items.Delete** özelliğini, hangi elemanın seçili olduğunu öğrenmek için de **ItemIndex** özelliğini kullanacağız. **Button2** düğmesinin **Click** olayına aşağıdaki kodu yazalım (Şekil 2.10).



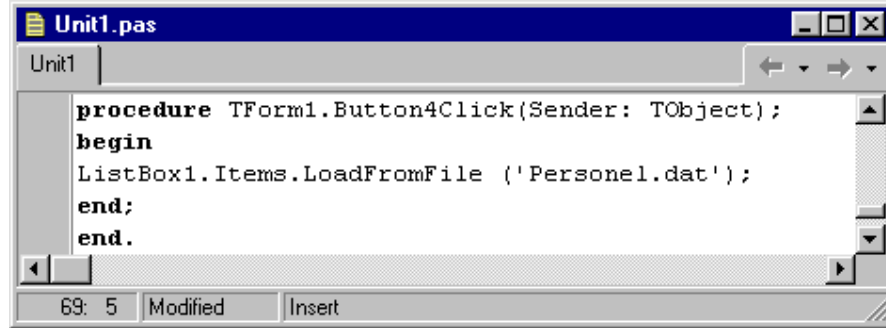
Şekil 2.10. Sil düğmesine ait kod sayfası

Bu işlemden sonra **Listbox** elemanına eklediğimiz personele ait bilgileri bir dosyada saklamak isteyelim. Dosyanın adı 'Personel.Dat' olsun. Bunun için **Kaydet** düğmesinin **Click** olayına ait kod aşağıdaki gibi olmalıdır (Şekil 2.11).



Şekil 2.11. Kaydet düğmesine ait kod sayfası

Son olarak da programa giriş yapıldığında '**pesonel.dat**' dosyasına kaydedilen bilgilerin ListBox kutusuna tekrar yüklenmesini sağlayalım. Bunun için Button4 yani **Yükle** düğmesinin **Click** olayına aşağıdaki kodu yazmamız gerekecektir (Şekil 2.12).



Şekil 2.12. Yükley düğmesine ait kod sayfası

Programımızın tasarımı için yapılması gerekenler burada bitmiştir. Programı çalıştırarak birkaç kayıt girdikten sonra Şekil 2.13'deki ekran görüntüsü alınmıştır.

Böylece bir programın ekranının (formunun) nasıl tasarlanacağını, form üzerindeki kontrollerin özelliklerinin (properties) nasıl değiştirileceğini ve gerekli kodu hangi olaylara (Events) nasıl yazacağımızı öğrenmiş olduk. Yapacağımız bütün programlarda bu adımları kullanacağız.

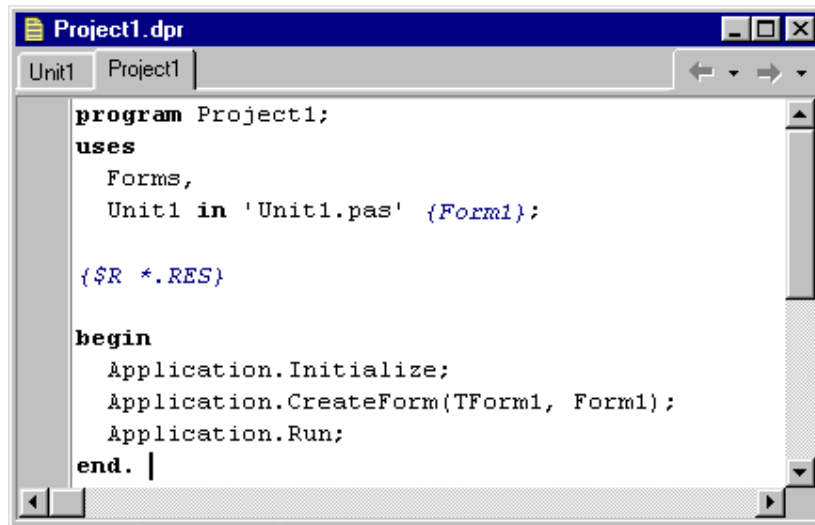


Şekil 2.13. Programın çalışması sırasındaki ekran görüntüsü.

2.5. Delphi Projelerinin Özellikleri

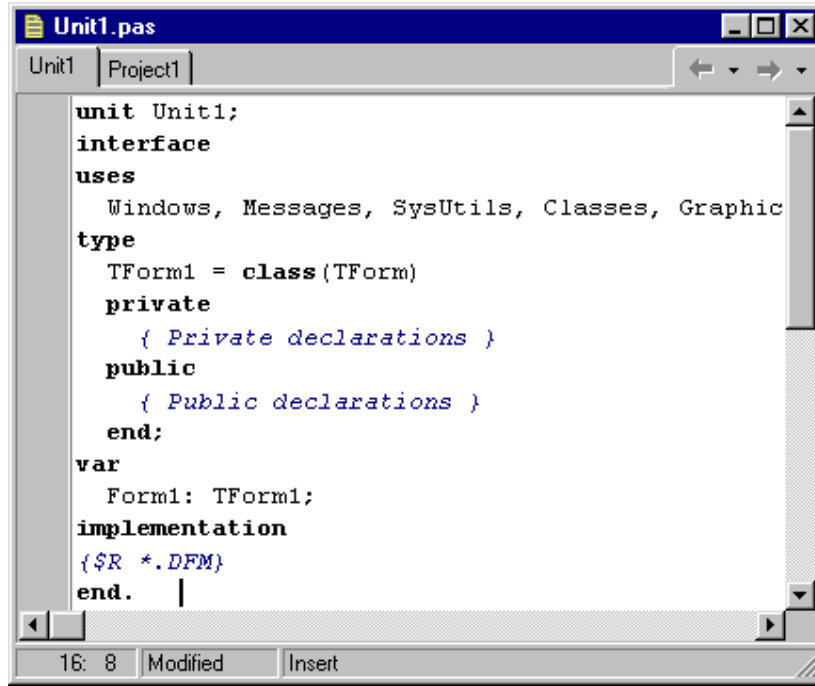
Bu bölümde Delphi projeleri hakkında öncelikle bilmeniz gereken konuları kısaca anlatacağım. Bu amaçla her Delphi projesi için otomatik olarak hazırlanan program kodlarını vereceğim. Proje dosyası içinde hazırlanan program kodlarını içeren dosyanın harddiske DPR uzantısı ile kaydedildiğini biliyoruz.

Delphi başlatıldığı zaman otomatik olarak hazırlanan ve varsayılan adı 'Project1' olan proje için Delphi tarafından otomatik olarak hazırlanan program kodlarını görmek ve gerekirse değişiklik yapmak için View menüsünden **Source** komutunu verince ekrana program kodu içeren Şekil 2.14'deki gibi bir pencere geldi.



Şekil 2.14. Project1.Dpr Penceresi

Unit penceresine geçilmekte, Unit penceresinde iken bu komut verildiğinde ise Form penceresine geçilmektedir. Aşağıda verilen ekran görüntüsünü üzerinde çalıştığım projedeki 'Form1' adlı forma ait pencere aktif iken View menüsünden **Toggle Form/Unit** komutunu verdikten sonra aldım.



Şekil 2.16

Şimdi size Unit1.Pas penceresindeki program satırlarını ve bu satırlarda kullanılan deyimleri tek tek anlatacağım.

Projelere **program** deyimi ile nasıl ad veriliyorsa Unit'lere aynı şekilde **Unit** deyimi ile ad verilmektedir. Yeni hazırlanan proje otomatik olarak dahil edilen Unit'e **Unit1** adı verildiği için Unit1.Pas adlı Unit penceresinin ilk satırında 'Unit Unit1' satırı bulunmaktadır. Daha sonra bu Unit'i harddiske başka bir ad ile kaydedecek olursanız Unit'e **unit** deyimi ile verilen ad otomatik olarak değişir.

Unit'lerde Unit adının belirtildiği ilk satırdan sonra **Interface** adlı kısımda, **Uses** bildiri deyimi ile Unit dahilinde kullanılacak hazır Delphi Unitleri belirlenmektedir. Yukarıda ekran görüntüsü verilen Delphi projesine otomatik olarak dahil edilen Unit'e ait program satırlarından tespit edileceği gibi her Delphi Unit'ine Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms ve Dialogs adlı hazır Delphi Unit'leri **USES** bildiri deyimi ile otomatik olarak dahil edilmektedir.

Type bildiri deyimi ile başlatılan blokta yeni tipler tanımlanmaktadır. Delphi projesine otomatik olarak dahil edilen Form1 adlı form, gerçekte bir nesnedir. Projedeki 'Form1' adlı nesne **Class** deyimi ile **TForm** nesne tipinden yararlanılarak tanımlanmaktadır. Delphi'de formları tanımlamada kullanılan nesne tipine **TForm** adı verilmektedir.

```
TForm1 = class(TForm)
```

Bu tanımlamadan sonra proje için TForm1 adlı form nesnesi tipi hazırlanmış olur. Üzerinde çalışılan proje dahilinde ikinci bir forma gerek duyulması halinde File menüsündeki **New Form** komutundan yararlanılmaktadır. Projedeki ilk form Delphi'nin başlatılması sırasında otomatik olarak hazırlanmaktadır. Üzerinde çalışılan projeye File menüsündeki New Form komutu ile ikinci bir formun dahil edilmesi halinde Unit1.Pas ile aynı program satırları içeren Unit2.Pas adlı program dosyası hazırlanıp projeye dahil edilir.

Type bildiri deyimi ile başlatılan tanımlama bloğunda Unit dahilinde kullanılacak nesne ve Procedure'lerin tanımlanması dışında varsa **Public** ve **Private** özellikli diğer tip tanımlamaları yapılabilmektedir. Type deyimi ile başlatılan tanımlama bloğunun sonu **End** deyimi ile işaret edilmektedir.

Var bildiri deyimi ile Unit dahilinde kullanılacak olan değişkenler tanımlanmaktadır. Henüz hazırlanan ve yalnızca 'Unit1' adında bir Unit içeren projede daha önce **Class** deyimi ile **TForm1** adında nesne tipi tanımlanmıştı. Değişken tanımlaması yapılan **Var** bloğunda Class deyimi ile hazırlanan nesne tipinden yararlanılarak değişken tanımlama işlemi yapılmaktadır.

```
Form1:TForm1 ;
```

Bu değişken tanımlama satırından sonra projeye dahil edilmiş olan Form1 adındaki form, **Form1** adlı nesne özellikli değişken ile temsil edilir. Var değimi ile başlatılan tanımlama bloğunda **Implementation** bildiri deyimi ile Unit'in asıl program satırlarına geçilmektedir. Unit'lerde istenilen sayıda yordam veya fonksiyon bulunabilmektedir. Başlangıçta Unit'te işlem yapmaya yönelik herhangi bir program satırı yoktur. Pascal programlama dilinin kurallarına göre Unitler ve Programlar **End** deyimi ile sona erdiği için Unit'in en son satırında sonunda nokta (.) işareti olan 'End' değimi var.

BÖLÜM 3

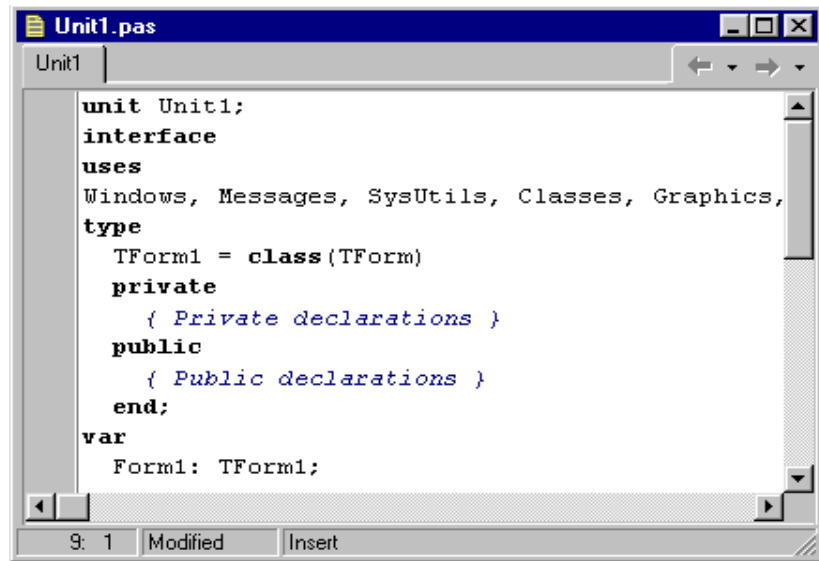
DEĞİŞKENLER

3.1. Değişken Tanımlamak

Quick Basic, Visual Basic ve xBASE program geliştirme araçlarının aksine, Delphi, proje dahilinde kullanılacak bütün değişkenlerin önceden tanımlanmasını zorunlu kılmaktadır. Henüz tanımlanmayan bir değişkeni Delphi projesi içinde kullanma şansınız yoktur.

Delphi gibi görsel program geliştirme araçları dahilinde kullanılan çok sayıda hazır nesne bulunmaktadır. Bu nesneler gerçekte dizi değişkenler gibi işlev gördüğü için programcının değişken tanımlama ihtiyacını azaltmaktadır. Örneğin dışarıdan girişi yapılan bilgileri saklamak için ayrıca değişken tanımlamaya gerek yoktur. Çünkü dışarıdan girilen veya dosyadan okunup ekrana yazılan bilgiler için forma eklenmiş olan metin kutuları nesnelere ait **Text** özelliğinden yararlanılmaktadır.

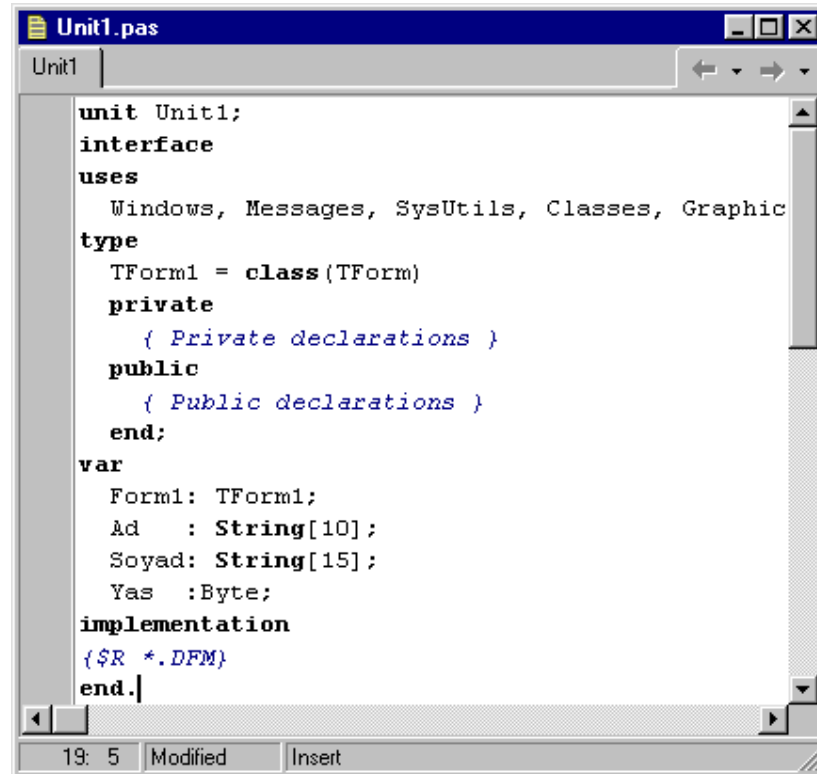
Delphi başlatıldığında veya File menüsündeki New Project komutu ile yeni bir projeye başlandığında, otomatik olarak hazırlanan UNIT'te değişken tanımlama işlemi yapılacak yerler Delphi tarafından işaret edilmektedir. Delphi tarafından otomatik olarak hazırlanan UNIT'in ekran görüntüsünü aşağıda verdim (şekil 3.1). UNIT'in başında Projede gerek duyulan hazır Delphi Unitleri, USES deyimi ile söz konusu UNIT'e dahil edilmektedir.



Şekil 3.1.

Uses deyimi ile UNIT'in gerek duyduğu bütün hazır Delphi Unitleri yüklendikten sonra, sıra **Type** deyimi ile tip tanımlamalarını yapmaya gelmektedir. Delphi programları dahilinde kullanılabilecek bütün bilgi tipleri için Delphi hazır tiplere sahiptir. Ancak programcı Delphi'nin standart tiplerinden yararlanarak yeni tipler tanımlayabilir. Şekil 3.1'de verilen ekran görüntüsünde Class deyimi ile TForm1 adında bir Form nesnesi tipi tanımlanmaktadır. Ardından Proje dahilinde kullanılacak **Private** veya **Public** yordamların tanımlanacağı yerler işaret edilmektedir.

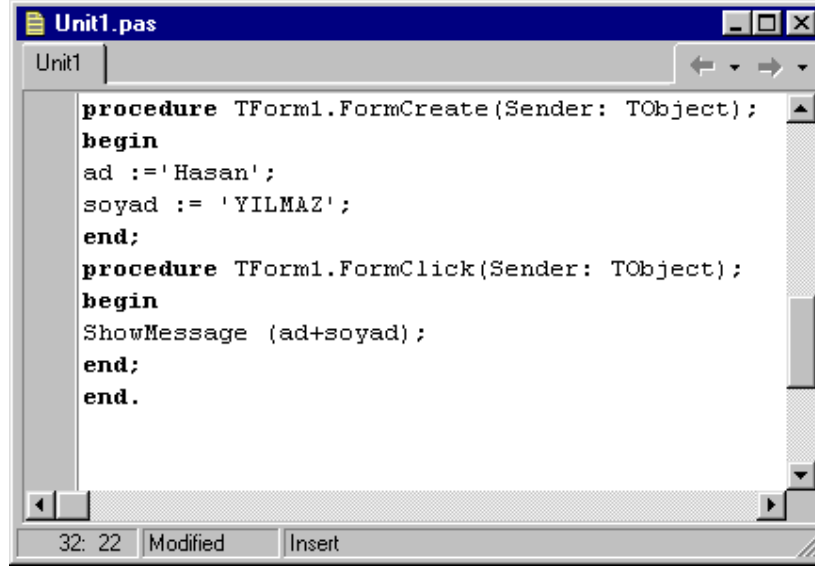
En son olarak **Var** bildiri deyimi ile işaret edilen blokta Unit dahilinde kullanılacak değişkenlerin tanımlanması işlemi yapılmaktadır. Yeni hazırlanan Delphi projelerine otomatik olarak bir Form nesnesi dahil edildiği için, daha önce Class deyimi ile hazırlanan Tipten yararlanılarak 'Form1' adında bir değişken tanımlanmaktadır. Aşağıda verilen ekran görüntüsünü üzerinde çalıştığım Projenin 'Unit1' adındaki Unit'i için bir kaç değişkeni tanımladıktan sonra aldım.



```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics;
type
  TForm1 = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
  Ad : String[10];
  Soyad: String[15];
  Yas :Byte;
implementation
{$R *.DFM}
end.
```

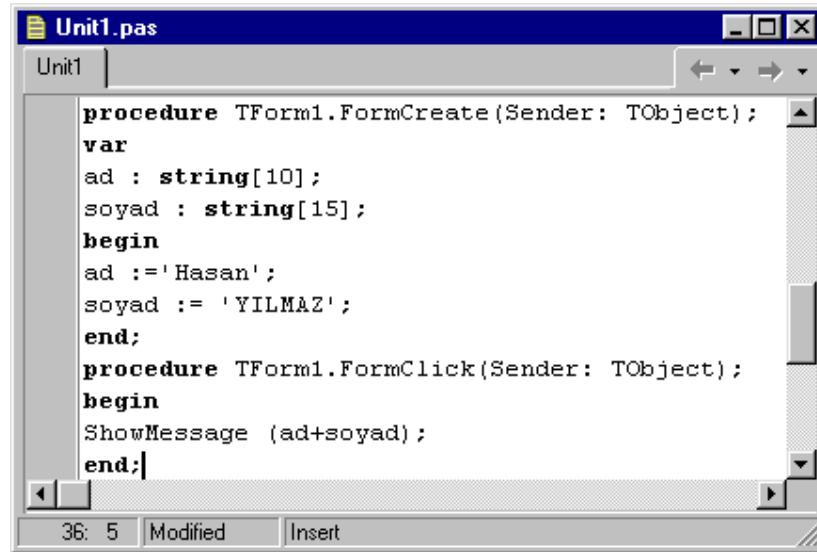
Şekil 3.2.

Bu şekilde tanımlanan bütün değişkenler Unit içinde istenen yordamda kullanılabilir. Unit'in **Var** bildiri deyimi ile başlatılan değişken tanımlama bloğunda tanımlanan değişkenlerin, Unit'e dahil edilmiş olan bütün yordamlarda kullanılabileceğini göstermek için bu Unite iki ayrı yordam ekledim. Aşağıda ekran görüntüleri verilen bu yordamlardan ilkinde değişkene doğrudan bilgi aktardım. İkinci yordamda ise bu değişkenlerin içeriklerini ShowMessage deyimi ile ekrana getirdim.



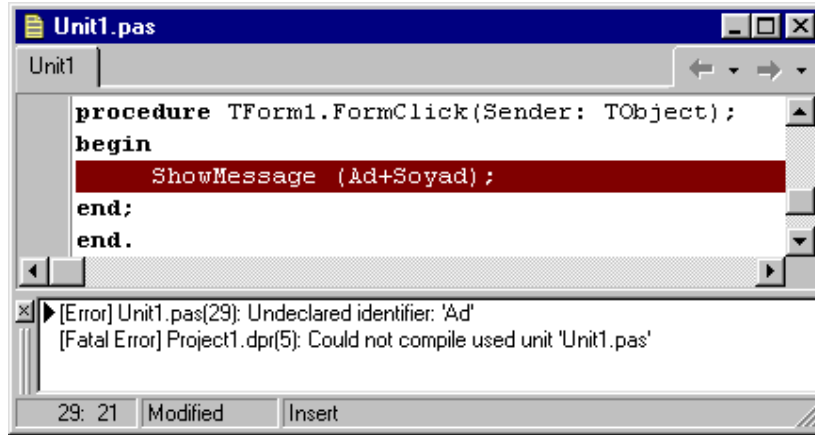
Şekil 3.3.

Eğer yalnızca üzerinde işlem yaptığınız yordamda kullanmak üzere değişken tanımlamak istiyorsanız, bu değişkenleri yordamın içinde tanımlayabilirsiniz. Yordamın içinde tanımlanan değişkenleri Unit'teki diğer yordamlarda kullanmazsınız.



Şekil 3.4.

Şekil 3.3'deki Unit'in her yerinde kullanmak üzere tanımladığım 'Ad' ve 'Soyad' değişkenlerini ilk tanımladığım yerden FormCreate yordamının içine taşıdım (Şekil 3.4). Bu tanımlamaya göre Ad ve Soyad değişkenleri yalnızca içinde tanımlandıkları FormCreate yordamında kullanılabilir. Bu nedenle Ad ve Soyad değişkenleri FormClick yordamı içinde tanınmadıkları için hata meydana gelir. Şekil 3.5'de verdiğim ekran görüntüsünü bu şekilde değiştirdiğim ve yalnızca bir form içeren bu örnek projeyi çalıştırdıktan hemen sonra aldım.



Şekil 3.5.

Herhangi bir yordamın içinde tanımlanan değişkenler, o yordamdan çıkılır çıkılmaz bellekten silinirler. Programcılar değişken tanımladıkları satırda aynı zamanda değişkene değer atama gereğini duyabilirler. Aşağıdaki program satırında değişken tanımlandıktan sonra değişkene değer aktarılmaktadır.

*Ad: **String** [10] = 'Hasan';*

Değişken tanımlama işlemi yapılırken aynı zamanda değer aktarılmasını ancak global özelliğe sahip yani Unit'in değişken tanımlama yapılan **Var** bloğunda tanımlanan değişkenler desteklemektedir. Aynı değişken tanımlama satırını bir yordamın içinde yazmanız halinde hata meydana gelir.

Bir değişken tanımlanırken aşağıdaki kuralların göz önünde bulundurulması gerekir:

- Değişken adları 63 karakter uzunluğunu geçmemelidir.
- Değişken adları ya bir alfabetik harflerle (İngiliz alfabesindeki) yada “_” karakteriyle başlamalıdır. İlk karakterden sonraki karakterler İngiliz alfabesindeki harfler, 0...9 arası rakamlar veya “_” olabilir.

- Değişken adları sembolleri içermemelidir (\$, *, % vb.).
- Delphi komutları değişken adı olarak kullanılmamalıdır. (Chr, IntToStr gibi)
- Bir değişken tanımlama işlemi tanımlama bloğunda (**Var** ile başlayan blok) yapılmalıdır.

3.2. Veri Tipleri

Önceki kuşak Pascal dillerinde kullanılan veri tipleri aynen Delphi’de de kullanılır. İstenilen bir değişkeni bu veri tipleriyle tanımlamak mümkündür. Bu veri tipleri şunlardır :

3.2.1. Tamsayı Tipleri

Ondalık nokta içermeyen tam sayısal bilgileri bellekte tutmak için tamsayı tiplerinden yararlanarak değişken tanımlama işlemi yapılır. Tamsayı tipleri aşağıda verilmiştir.

ShortInt: 1 baytlık işaretli tamsayı tipidir. –128 ile 127 arasında değer alabilir.

SmallInt: 2 baytlık işaretli tamsayı tipidir. –32.768 ile 32.767 arasında değer alabilir.

LongInt: 4 baytlık işaretli tamsayı tipidir. –2.147.483.648 ile 2.147.483.647 arasında değer alabilir.

Integer: Delphi’nin 16 bitlik versiyonunda Integer tipi 16 bitlik, Delphi’nin 32 bitlik versiyonlarında (2.0, 3.0, 4.0, +) ise bu tip 32 bitlik bir değer içerebilir. Buna Integer tipi de 4 baytlık işaretli tamsayı tipidir. –2.147.483.648 ile 2.147.483.647 arasında değer alabilir.

Byte: 1 baytlık işaretsiz tamsayı tipidir. 0 ile 255 arasında değer alabilir. (ShortInt tipinin işaretsiz halidir.)

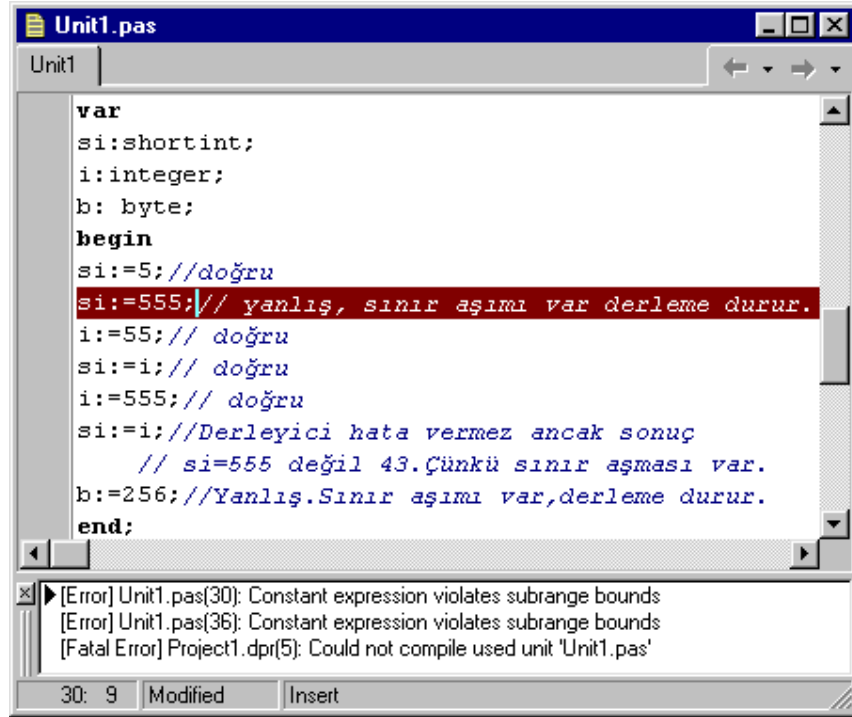
Word: 2 baytlık işaretsiz tamsayı tipidir. 0 ile 65.535 arasında değer alabilir.

Cardinal: 4 batlık işaretsiz tamsayı tipidir. 0 ile 2.147.483.647 arasında değer alabilir.

Bu tiplere alabileceği değer sınırları 0dışında değer ataması direkt olarak yapılamaz. Atama yapılmaya çalışıldığı durumlarda aşağıdaki hata mesajı çıkar.

Constant expression violates subrange bounds

Bu tipler arasında deęer ataması yapılabilir. Ancak sınırların aşması durumunda sonuç hatalı çıkacaktır. Şekil 3.6'da gördüğünüz gibi bir deęişkene sınır dışında bir deęer atadığınızda Delphi hata verecektir. Ancak bu deęer direk olarak deęil de bir



Şekil 3.6.

işlem sonucunda verilirse Delphi hata uyarısı vermeyecek ancak sonuç da yanlış çıkacaktır.

Deęişkenlerinizi tanımlarken bu sınırlara dikkat etmelisiniz. Aksi takdirde her şey doęru olduęu halde sonuçlar yanlış olabilir.

3.2.2. Reel Sayı Tipleri

Ondalık nokta içeren sayısal bilgileri bellekte saklamak için reel sayı tiplerinden yararlanarak deęişken tanımlama işlemini yapılır. Reel sayı tipleri ve sınırları aşağıda verilmiştir.

Single: 4 baytlık ondalık sayı tipidir. 1.5×10^{-45} ile 3.4×10^{38} arasında deęer alabilir. 7 – 8 haneli rakamlar.

Real: 6 baytlık ondalık sayı tipidir. 2.9×10^{-39} ile 1.7×10^{38} arasında değer alabilir. 11 – 12 haneli rakamlar.

Double: 8 baytlık ondalık sayı tipidir. 5.0×10^{-324} ile 1.7×10^{308} arasında değer alabilir. 15 – 16 haneli rakamlar.

Extended: 10 baytlık ondalık sayı tipidir. 3.4×10^{-4932} ile 1.1×10^{4932} arasında değer alabilir. 19 – 20 haneli rakamlar.

Comp: 8 baytlık ondalık sayı tipidir. $-2^{63} + 1$ ile $2^{63} - 1$ arasında değer alabilir. 19 – 20 haneli rakamlar. Comp tipi sadece 2'nin katları olan sayıları tutabilen bir tiptir.

Currency: 8 baytlık -922337203685477.5808 ile $+922337203685477.5807$ aralığında işaretli bir sayı tipidir. Bu tip para içeren değişkenler için düşünülmüştür. Çünkü parasal işlemlerin virgülden sonraki kısmı o kadar önemli değildir. Asıl önemli olan virgülden önceki bütün basamakların korunmasıdır. Bu tipte virgülden önceki basamak sayısı çok, virgülden sonraki basamak sayısı azdır.

Bu tiplere de kendi aralarında atama yapılabilir. Yine direkt sınır dışı atamalarda derleyici işlemi durdurur. Dolaylı atamalarda atama yapılır ancak sonuç doğru çıkmaz.

3.2.3. Boolean Tipi

Boolean tipi ile tanımlanan değişkenler bellekte yalnızca 1 byte yer kaplar ve bu değişkenler yalnızca **True** ve **False** değerlerini alabilirler. Eğer üzerinde çalıştığınız projede doğru – yanlış veya evet – hayır gibi yalnızca iki değer alabilecek değişkenlere gerek duyuyorsanız, Delphi'nin Boolean tipinden yararlanarak değişken tanımlayabilirsiniz.

var

EvetHayır : Boolean;

DogruYanlis : Boolean;

3.2.4. Karakter Tipleri

Char: 1 baytlık veri tipidir. Bu tip değişkenler sadece bir karakter barındırabilirler. Örneğin 'A', '2', '-' gibi. Eğer Proje dahilinde tek karakterlik bilgileri geçici olarak bellekte saklama gereğini duyuyorsanız **Var** bildiri deyimini ile başlatılan

bloкта Delphi'nin hazır Char tipinden yararlanarak değişken tanımlayabilirsiniz. Char tipinden yararlanarak tanımlanan değişkenlerde ASCII karakter kümesindeki 256 adet karakterden biri saklanabilir.

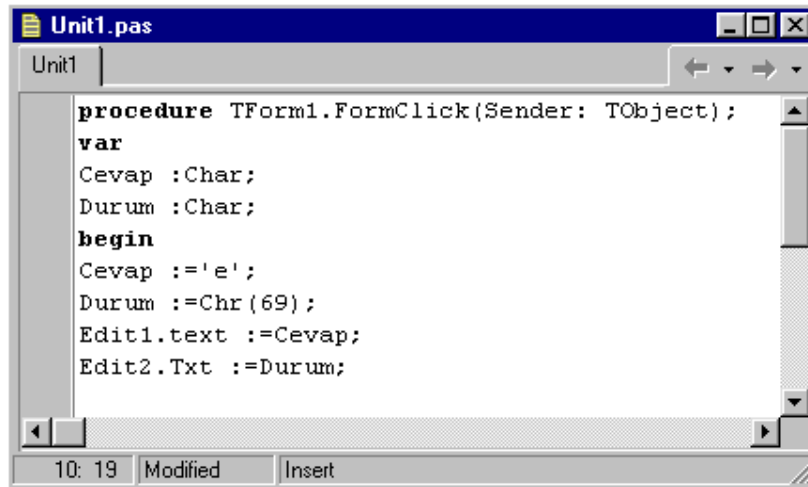
Var

EvetHayir : Char;

Bu şekilde tanımlanan değişkene daha sonra istenen karakter aktarılabilir. Char tipindeki değişkenlere aktarılan ASCII karakter kümesindeki bazı bilgiler, ekranda veya yazıcıda görüntülenebilen bir karakteri temsil etmeyip, bir etkiyi temsil etmektedir.

Bu tip bilgileri doğrudan Char tipindeki değişkenlere aktarma imkanı olmadığı için **Chr()** fonksiyonundan yararlanır. Char tipindeki değişkende saklanan bilginin ASCII kodunu öğrenmek istiyorsanız, **Ord()** fonksiyonundan yararlanabilirsiniz. Char

ve String tipine sahip bir değişkene sabit bir bilgi aktarılırken, sabit bilgi tek tırnak (' ') içine alınmaktadır.



Şekil 3.7.

AnsiChar: Char tipi ile aynıdır.

WideChar: Bu tip 2 baytlık bir karakter tipidir ve uzak doğu dillerinde kullanılır.

3.2.5. String Tipleri

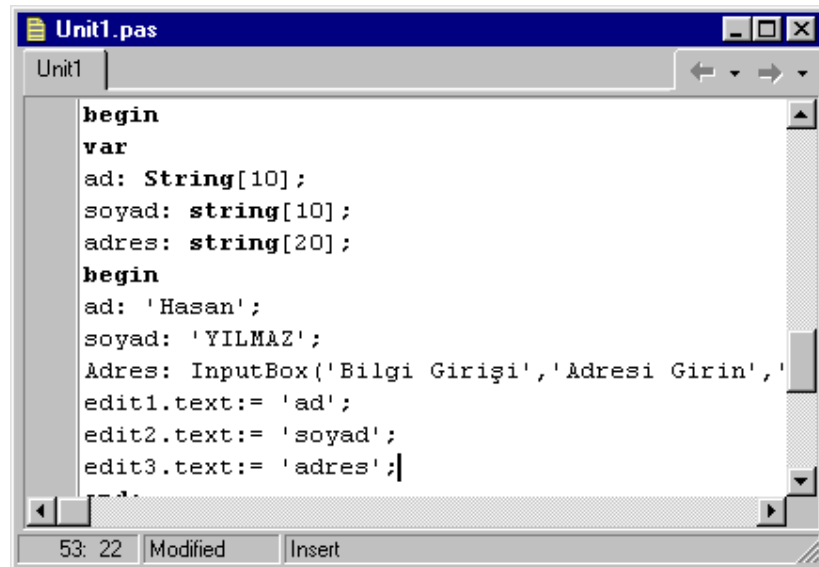
String: Birden fazla karakteri bellekte veya dosyalarda saklamak için String tipindeki değişkenlerden yararlanılır. String bildiri deyimi ile değişken tanımlama işlemi yapılırken, köşeli parantezler içinde değişkende en fazla kaç karakterin saklanmak istendiği belirtilir.

Var

Ad : String [10];

Soyad: String [15];

Bu iki değişken tanımlama satırı ile en fazla 10 karakter uzunluğunda bilgi alabilecek ‘Ad’ değişkeni ve en fazla 15 karakter bilgi alabilecek ‘Soyad’ değişkeni tanımlanmaktadır. Değişkenin belirtilen uzunluğundan daha fazla bilgi aktarılabilecek olunursa, değişkene aktarılan bilginin bir kısmı kesilebilir. Uzunluğu 1 olarak seçilen String tipindeki bir değişkenin, Char tipindeki değişkenden bir farkı yoktur.



Şekil 3.8.

Şekil 3.8’deki “Ad” ve “Soyad” adındaki ilk iki değişkene doğrudan bilgi aktarına işlemi yapılırken, “Adres” değişkenine **InputBox()** fonksiyonu aracılığı ile bilgi aktarılmaktadır. String tipindeki değişkenlere yalnızca karakterel bilgiler aktarılabilir. Sayısal veya tarihsel tipteki bilgileri String tipindeki değişkenlere aktarma

gereğini duyarsanız, **IntToStr()** gibi tip dönüştürme fonksiyonlarından yararlanmanız gerekir. Daha sonra bu fonksiyonlar hakkında bilgi verilecektir.

ShortString: 255 karaktere kadar karakter ataması yapılabilen veri tipidir. bellekte karakter sayısı 1 bayt yer kaplar. Çünkü bu tipteki değişkenlerin en yüksek seviyeli baytı değişkenin uzunluğu için kullanılır. Bir bayt içinde 0 – 255 arası sayı tutulabileceği için bu tipteki verilere 255 karakterden fazla atama yapılamaz. Yapılan atamanın 255 karakteri geçmesi sondakilerin kesilmesine neden olur.

AnsiString: Bu tip stringler dinamiktir ve belli bir sınırı yoktur. Yani bu değişkene ne kadar karakter atanırsa bellekte o kadar yer kaplar. String tipi ile aynı özelliğe sahiptir.

PChar: 64 KByte'a kadar atama yapılabilen, sonu #0 karakteri ile biten string veri tipidir.

3.3. Diziler

Aynı tipte ve birbiriyle ilgili bilgilerin oluşturduğu bütüne **dizi** denir. Program içersinde aynı anda aynı tür bilgiden çok sayıda ihtiyaç olması ve bu bilgiler üzerinde toplu işlem yapılmasının gerekmesi durumunda dizilerden yararlanılır. Dizilerin gerekliliğini iki örnek ile açıklayalım.

Örnek 1: 100 adet isim ve telefon bilgisini saklamak için ;

İsim : 100 adet

Telefon : 100 adet

Toplam 200 adet değişken kullanılmalıdır.

Örnek 2: bir sınıfta okuyan 40 öğrencinin isim ve her öğrencinin 5 farklı dersten aldıkları 3 farklı not bilgisini aynı anda makine hafızasında tutmak için:

İsim : 40 adet

Notlar : 40 öğrenci x 5 ders x 3 not = 600 adet

Toplam 640 adet değişken kullanılmalıdır.

Örnek 1 ve 2'de görüldüğü gibi çok sayıda değişken kullanılması gerekecektir. Bu durumda bilgiler üzerinde işlem yapılması imkansızdır. Bu durumu kolaylaştırmak için dizilerden yararlanılır.

Aynı örnekleri diziler yardımıyla tanımlarsak Örnek 1 için toplam 200 elemanlı 2 dizi (değişken), Örnek 2 için ise toplam 600 elemanlı 2 dizi (değişken) kullanılmalıdır. Görüldüğü gibi değişken sayısının yerini dizideki eleman sayısı almıştır. Değişken sayısı oldukça azalmıştır.

Dizi içerisinde her elemanın bir indis numarası bulunur. İndis numarası her elemanın dizi içerisindeki yerini yani kaçınca eleman olduğunu gösterir. İndis numaraları [] içerisinde yazılır.

Örnek 1 için;

isim [1] isim dizisinin 1. elemanını
isim [x] isim dizisinin x. elemanını
telefon [35] telefon dizisinin 35. elemanını temsil eder.

Dizi tanımlı **Var** kısmında aşağıdaki gibi yapılır.

Var

Dizi_adi : array [altsınır .. üstsınır] of tip

Örneğin 100 elemanlı ve **integer** tipli bir dizi şöyle tanımlanır.

Var

Sayılar : array [1 .. 100] of integer;

Burada 1'den 100'e kadar (1 ve 100 dahil) bir dizi tanımlandı. Dizinin her bir elemanının tipi integerdır. Sayılar dizisi aynı tipte 100 tane değer içerir. Dizinin 10. elemanına 50 değerini aşağıdaki gibi aktarıyoruz.

Sayılar [10] := 50;

Dizinin tüm elemanlarına 100 değerini aktarmak için ise aşağıdaki gibi bir döngü yazmalıyız.

For i := 1 to 100 do

Sayılar [i] := 100;

Diziler tek boyutlu tek boyutlu olduğu gibi çok boyutlu da olabilirler. N boyutlu bir dizi şu şekilde tanımlanır:

Var

Dizi_adi : array [altS1 .. üstS1, altS2 .. üstS2,, altSn .. üstSn] of tip ;

Buna göre iki boyutlu bir dizi tanımlaması şöyle olur:

Var

Sayilar : array [1 .. 100, 1 .. 100] of Double;

Buradaki sayılar dizisinin elemanlarına ulaşmak için sayılar [m,n] yapısı kullanılır.

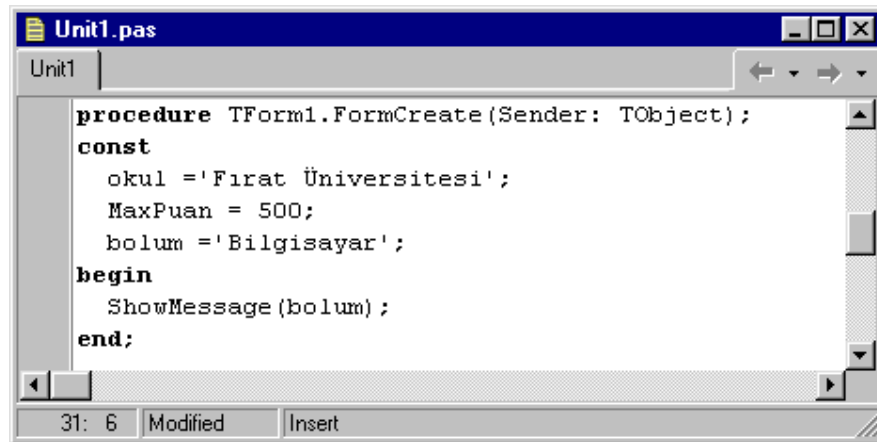
3.4. Sabit Tanımlamak

Sabitler değişmeyen değerler içerirler. Bu değerler programın başından sonuna kadar geçerliliklerini korurlar. Bir sabiti tanımlama işlemi **Const** bloğunda yapılmalıdır. Bu tanımlama işlemi **Const** deyimi ile başlar. Tanım şekli şöyledir:

Const

SabitAdı = Değeri ;

Burada sabit ismi ile değeri arasında := atama operatörü yerine = operatörü kullanılır. Ve bu atanan değer blok içersinde değiştirilemez.



Şekil 3.9.

Yukarıdaki şekilde okul, maxpuan ve bolum tanımlamaları birer sabiti ifade ederler. Ve program boyunca değerleri değiştirilemez.

Const kısmında bu tip sabitler tanımlanabilirken aynı zamanda ilk değer atanması gereken ve static olan değişkenlerin tanımında da kullanılır.

3.5. İlk Değer Atama ve Static Değişken Tanımı

Değişkenlere ilk değerin atanması ve bu değişkenin değerini prosedürün çalışması bittikten sonra da koruması için yine **Const** kısmı altında değişken tanımlı yapılır.

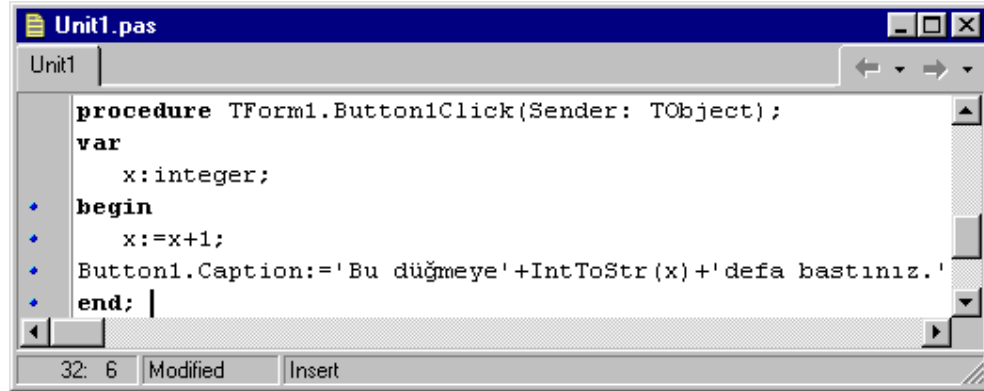
Const

Deg_Adi : Tipi = İlkDeğeri ;

Dikkat edilirse sabit tanımından farklı olarak bu değişkenin adı ile ilk değeri dışında değişkenin tipi belirtilir ve bu tanım bir sabit tanımı değildir. Bu bir değişkendir ancak özel bir durumu vardır.

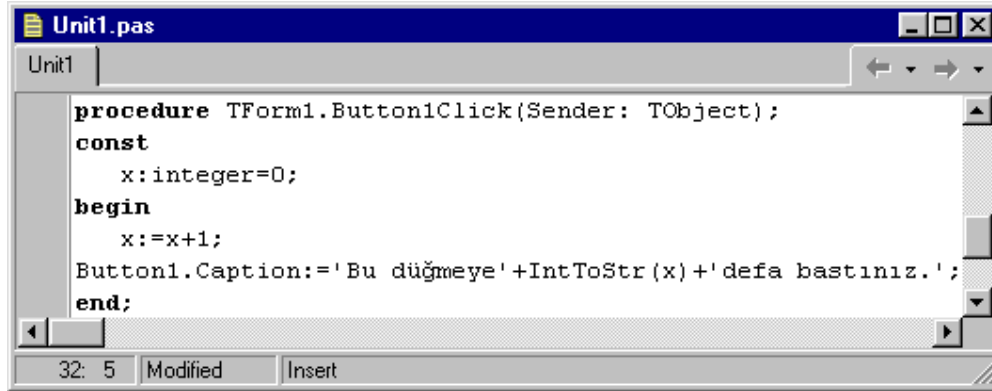
Örneğin bir düğmenin kaç defa tıklandığını yazacak bir programımız olsun. Programı Şekil 3.10'daki gibi yazalım.

Programı çalıştırırsanız düğmeye her bastığınızda aynı sonucu ve 1 olmayan bir sonucu göreceksiniz. Çünkü değişkene ilk değer ataması yapılmamıştır dolayısıyla x'in değeri rasgele bir sayıdır. Her seferinde aynı sonucu görmeyişinizin sebebi ise değişkenin Local olması nedeniyle işlem bittikten sonra bellekten atılması ve bir sonraki sefer basıldığında bir önceki sefer kaldığı değeri hatırlamamasıdır.



Şekil 3.10.

Sayıya ilk değer atamak ve bir sonraki seferde de bir önceki değerini hatırlayabilmesi için programın aşağıdaki gibi olması gerekir.



Artık kodumuz düğmeye kaç defa basıldığını doğru olarak gösterebilecektir. Burada dikkat edilirse ilk değer atama işlemi her sefer yapılmamaktadır. Böyle olsa idi her sefer aynı sonucu alacaktık. İlk değer atama işlemi sadece prosedür ilk defa çalıştırıldığında yapılacak sonraki seferlerde ise bir önceki seferde değişkenin aldığı değerden devam edecektir.

Bir diziye ilk değer atamanın genel formu ise aşağıdaki gibidir.

Const

Dizi_Adi : Array [AltSınır .. ÜstSınır] of Tipi = (Değer1, Değer2, ...);

Örneğin Mevsimler adındaki dizinin 1. elemanına 'İlkbahar', 4. elemanına da 'Yaz' değerini atayan program satırları aşağıdaki gibi olmalıdır.

Const

Mevsimler : Array [1 .. 4] of String = ('İlkbahar', 'Yaz', 'Sonbahar', 'Kış');

3.6. Pointerler

Pointerler gerçekte bir değeri değil bir değerın bulunduğu adresi gösterirler. Eğer pointerın nasıl bir bilgiyi göstereceği belli ise aşağıdaki pointer tiplerinden biri kullanılabilir.

PAnsiString: AnsiString tipinde bir değişkeni gösterecek pointer.

PByteArray: TByteArray tipinde bir değişkeni gösterecek pointer. Çoğunlukla bellekte ayrılmış bölgelerdeki her bayta ulaşabilmek için kullanılır.

PCurrency: Currency tipinde bir değişkeni gösterecek pointer.

PShortString: ShortString tipinde bir değişkeni gösterecek pointer.

PWordArray: TWordArray tipinde bir değişkeni gösterecek pointer. Çoğunlukla bellekte ayrılmış bölgelerdeki her worde (2 bayt) ulaşabilmek için kullanılır.

Bunların haricinde herhangi bir tip veya değişken için ^ karakteri kullanılarak kolayca pointer tanımlanabilir.

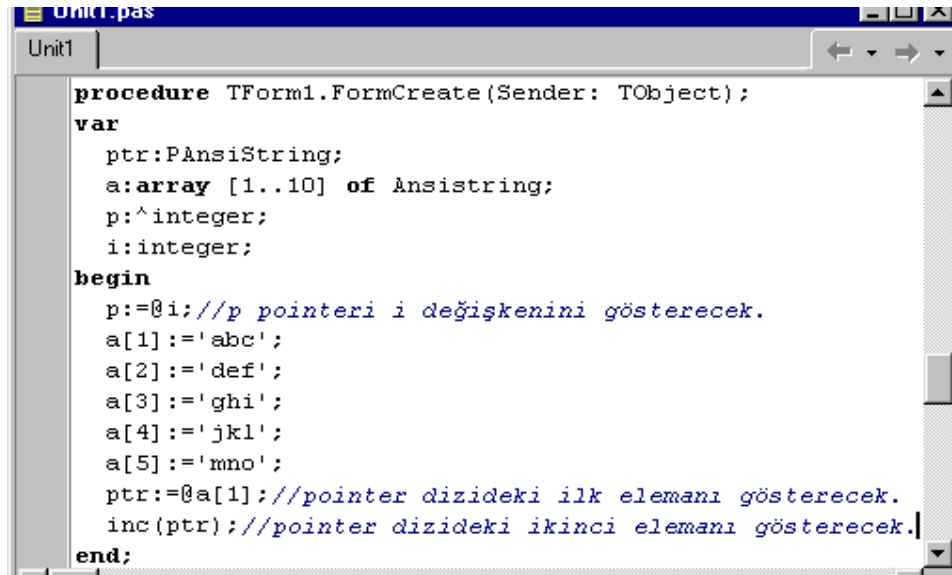
Var

Ptr :PAnsiString ; // AnsiString tipinde bir değeri gösteren pointer.

P : ^ Integer; // integer tipinde bir değeri gösterecek pointer.

3.6.1. Pointerin Barındıracağı Adres

Pointer tipler gerçekte bir değeri değil bir adresi barındırırlar. Bu yüzden hangi adresi barındırdıkları belirlenmelidir. Bu adres belirleme işlemi @ karakteri ile yapılır.



Şekil 3.12.

Yukarıdaki örnekte;

P:=@i;

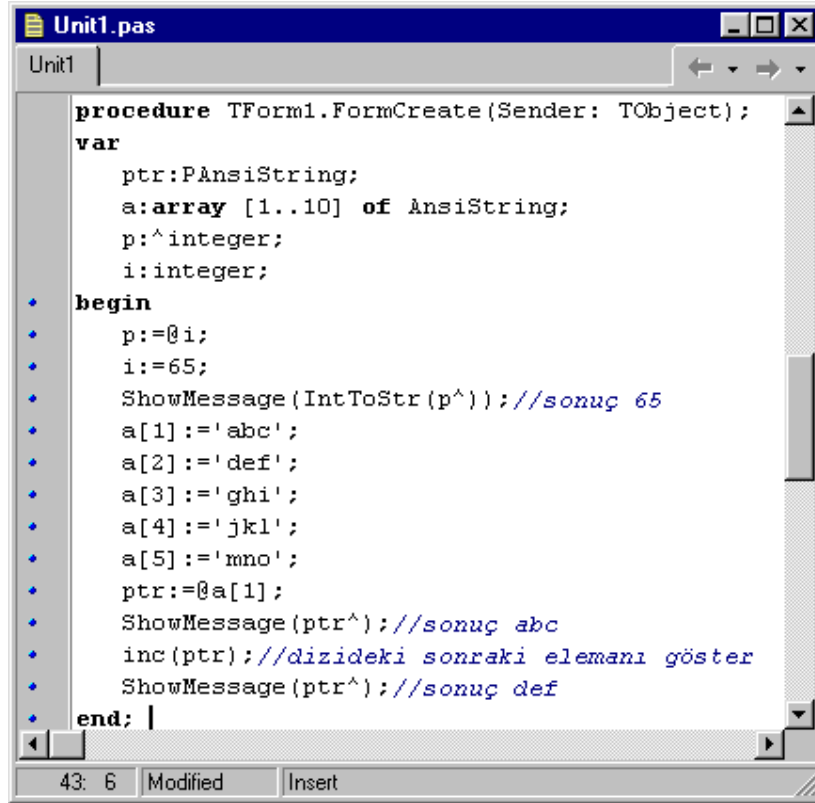
Bu satır ile p pointeri i değişkeninin adresini barındıracaktır.

Ptr:=@a[1];

Bu satır ile de ptr pointerinin dizideki ilk elemanın adresini barındıracağını belirtmiş oluyoruz.

3.6.2. Pointerin Gösterdiği Değer

Bir pointerin değeri değil adresi barındırdığını belirtmiştik. Bir pointerin barındırdığı adreste bulunan değeri ise ^ karakterini kullanarak öğrenebiliriz. Şekil 3.13’te örnek bir program kodu ve çıktıları verilmiştir.



```
Unit1.pas
Unit1

procedure TForm1.FormCreate(Sender: TObject);
var
  ptr:PAnsiString;
  a:array [1..10] of AnsiString;
  p:^integer;
  i:integer;
begin
  p:=@i;
  i:=65;
  ShowMessage(IntToStr(p^)); //sonuç 65
  a[1]:='abc';
  a[2]:='def';
  a[3]:='ghi';
  a[4]:='jkl';
  a[5]:='mno';
  ptr:=@a[1];
  ShowMessage(ptr^); //sonuç abc
  inc(ptr); //dizideki sonraki elemanı göster
  ShowMessage(ptr^); //sonuç def
end;
```

Şekil 3.13.a. Program Kodu



Şekil 3.13.b. Programın Çıktıları

3.6.3. Pointer Kullanımı

Pointer, tiplerin önüne ^ işareti konularak tanımlanırlar.

Var

X : ^ Integer;

Y : ^ String;

Bu değişkenlere değer atanırken değişkenin arkasında ^ operatörü yine kullanılır.

Var

X : ^ Integer;

Y : ^ String;

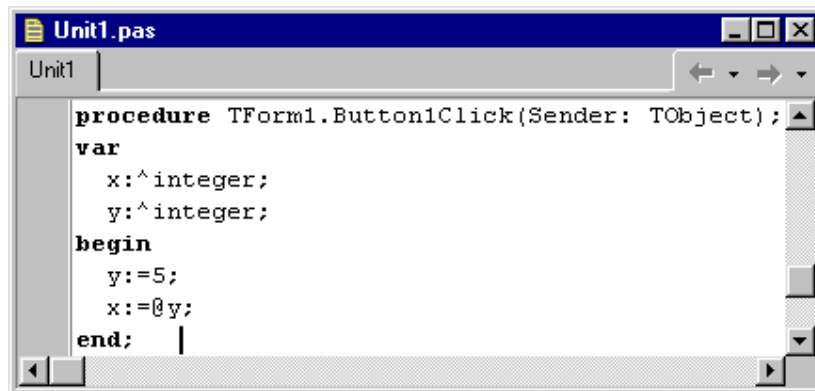
Begin

X ^ := 10;

End;

Normalde değişkenler içlerinde bir değer bulundurlar. Ancak pointerler içlerinde bir değer değil bir adres bulundurlar. Bulundurdıkları bu adresteki değeri temsil ederler.

Pointer veya bir değişkenin adresini öğrenebilmek için @ operatörü kullanılabilir.



Şekil 3.14.

Şekil 3.14'teki program kodlarında x pointerinin y değişkeninin adresini barındırması istenmektedir. Bu durumda x'in içeriğinde y'nin bellekteki adresi bulunacaktır. @x şeklinde kullanılması halinde ise bu adresteki değer yani 5 kastedilmektedir.

Şekil 3.14'teki tanımlama ile Delphi her iki değişken için bellekte yer ayıracaktır. Örnek olarak Delphi, x değişkeni için 100 numaralı adresi, y değişkeni için de 200 numaralı adresi kullanıyor olsun. Tanımlama yapılırken y değişkenine 5 değeri atandığı için 200 numaralı bellek bölgesinde 5 değeri bulunacaktır. **x := @y;** ataması ile de x pointerine y değişkeninin adresi yani 200 atanmaktadır.

Adres	Bellek içeriği	Açıklaması
.....
100	200	Bellekte x değişkeninin bulunduğu yer
.....
200	5	Bellekte y değişkeninin bulunduğu yer
.....

Bu işlemden sonra x demekle 200 sayısını yani bir adresi, @x demekle x'in içinde bulunan adresteki değeri, yani 200 nolu adresteki değeri (5) kastetmiş olacağız.

Bir pointer bir diziye de referans gösterebilir.

Var

X: ^ Integer;

Y: array [0 .. 2] of Integer;

Begin

X := @ y; // x pointeri diziye gösterecek

End;

Adres	Bellek içeriği	Açıklaması
100	200	Bellekte x değişkeninin bulunduğu yer
.....
200	y [0]	Bellekte y dizisinin bulunduğu yer
	y [1]	
	y [2]	

Burada görüldüğü gibi bir pointerin diziye göstermesini istediğimizde dizideki bir elemanı değil dizinin ismini kullanıyoruz. Bu durumda pointer dizideki ilk elemanı gösterecektir. İstenirse dizideki eleman belirtilerek de pointerin o elemanı göstermesi sağlanabilir.

Var

X: ^ Integer;

Y: array [0 .. 2] of Integer;

Begin

X: = @ y [1]; // x pointeri dizideki ikinci elemanı gösterecek

End;

Adres	Bellek içeriği	Açıklaması
100	204	Bellekte x değişkeninin bulunduğu yer
.....
200	y [0]	Bellekte y dizisinin bulunduğu yer
204	y [1]	
208	y [2]	
.....

Pointerlerde değişkenin uzunluğu tanımlandığı tipe göre değişmez. Çünkü pointer hangi tipe tanımlanırsa tanımlansın sonuçta içeriğinde o tipte bir değer bulunmaz, sadece bir adres bulunur.

Var

X: ^ Integer; // x pointeri int tipinde bir değeri gösterecek

C: ^ Char; // c pointeri char tipinde bir değeri gösterecek

L: ^ LongInt; // l pointeri longint tipinde bir değeri gösterecek

Yukarıdaki program satırlarında verilen her üç pointerde bellekte eşit uzunlukta yer kaplar. Sadece bunların tipleri gösterdikleri bölgedeki değerlerin hangi tipte olduğunu belirtmek içindir. Pointerler üzerinde yapılan işlemler de bellek işlemleridir.

Var

X: ^ Integer;

Begin

Inc(x);

End;

Yukarıdaki işlem x'in değerini bir arttırmaz, **Integer** tipinin uzunluğu kadar artırır. Yani 4 bayt arttıracaktır.

Var

X: ^ Integer;

Y: array [0 .. 2] of Integer;

Begin

X: = @ y; // x pointeri y dizisini gösterirsin

End;

Delphi, belleğe bu değişkenleri aşağıdaki gibi yerleştirmişse **x:=@y** ataması sonucunda x'in içeriği y'nin adresi olan 200 olacaktır.

Adres	Bellek içeriği	Açıklaması
.....
100	200	Bellekte x değişkeninin bulunduğu yer
.....
200	y [0]	Bellekte y dizisinin bulunduğu yer
204	y [1]	
208	y [2]	
.....

Bu işlemden sonra **Inc(x);** yapılırsa x=201 olmaz. Bunun yerine x pointeri bellekte bir sonraki integer değerini gösterecek şekilde 4 artırılır yani 204 yapılır. Böylece 204 numaralı adreste bulunan, yani dizinin sonraki elemanını göstermiş olur.

Var

X: ^ Char;

Y: array [0 .. 2] of Char;

Begin

X: = @ y; // x pointeri y dizisini gösterirsin

End;

Delphi, belleğe bu değişkenleri aşağıdaki gibi yerleştirmişse **x:=@y** ataması sonucunda x'in içereceği y'nin adresi olan 200 olacaktır.

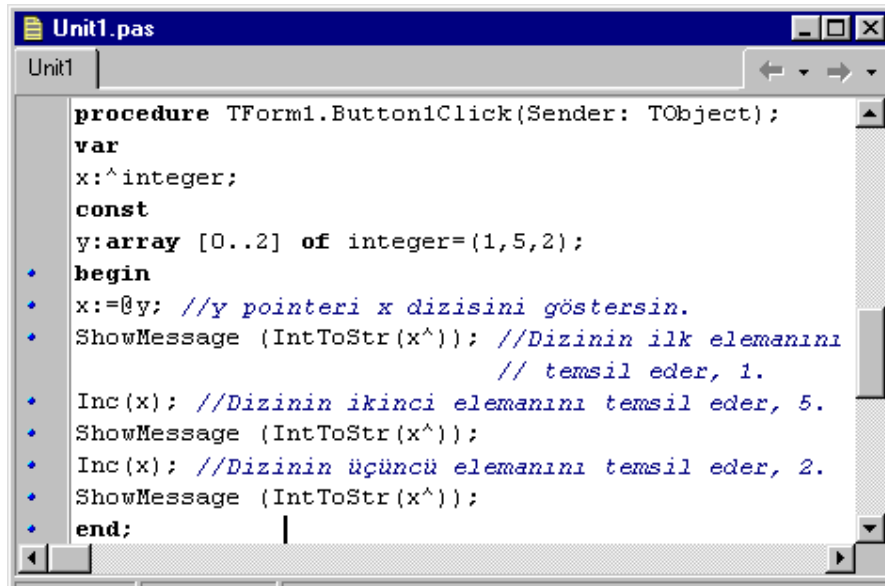
Adres	Bellek içeriği	Açıklaması
.....
100	200	Bellekte x değişkeninin bulunduğu yer
.....
200	y [0]	Bellekte y dizisinin bulunduğu yer
201	y [1]	
202	y [2]	

--	--	--

Bu işlemden sonra **Inc(x)** yapılırsa $x=201$ olur. Çünkü **char** 1 bayt olduğu için 1 artırılması sonucunda 201 elde edilecektir.

Pointer değişkenler dizilere ulaşmakta büyük kolaylıklar sağlarlar. Inc ve dec kullanılarak pointerin gösterdiği yerler değiştirilebilir. Eğer pointer bir diziyi gösteriyorsa **Inc** bir sonraki elemanı, **Dec** bir önceki elemanı gösterir. İstenirse bir sonraki/bir önceki değil de istenen eleman da gösterilebilir.

Aşağıdaki Şekil 3.15'te örnek bir programın kod penceresi ile bu programdan elde edilen çıktılar gösterilmiştir.



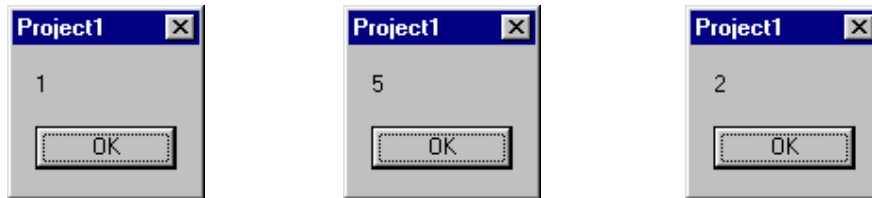
```

Unit1.pas
Unit1

procedure TForm1.Button1Click(Sender: TObject);
var
  x:^integer;
const
  y:array [0..2] of integer=(1,5,2);
begin
  x:=@y; //y pointeri x dizisini gösterecek.
  ShowMessage (IntToStr(x^)); //Dizinin ilk elemanını
                                // temsil eder, 1.
  Inc(x); //Dizinin ikinci elemanını temsil eder, 5.
  ShowMessage (IntToStr(x^));
  Inc(x); //Dizinin üçüncü elemanını temsil eder, 2.
  ShowMessage (IntToStr(x^));
end;

```

Şekil 3.15 a. Program Kod Penceresi



Şekil 3.15.b. Programın Çıktıları

Örnek:

Var

i, j: Integer;

p:^integer;

Begin

```

p:=@ i;
i:=12;
j:=i*p^;
i:=p^+1;
ShowMessage (IntToStr(i) + ';' + IntToStr(j));
End;

```

Yukarıdaki işlemlerin sonucunda i ve j değişkenlerinin son değerini bulmaya çalışalım. İşlemi bellek düzeyinde düşünerek yapalım. Derleyici değişkenlerimizi aşağıdaki gibi belleğe yerleştirmiş olsun. (Değişkenlerin belleğin neresine yerleştirildiği önemli değildir. Örneklerin iyi anlaşılması amacıyla bu şekilde anlatılmaktadır.)

Adres	Bellek içeriği	Açıklaması
.....
100	??	Bellekte p değişkeninin bulunduğu yer
.....
200	??	Bellekte i değişkeninin bulunduğu yer
.....
250	??	Bellekte j değişkeninin bulunduğu yer

p:=@i ataması sonucunda p pointeri x'in adresini gösterecektir.

Adres	Bellek içeriği	Açıklaması
.....
100	200	Bellekte p değişkeninin bulunduğu yer
.....
200	??	Bellekte i değişkeninin bulunduğu yer
.....
250	??	Bellekte j değişkeninin bulunduğu yer

i:=12 ataması ile bellekte i değişkenine 12 atanacaktır. Dolayısıyla p pointeri de 200 nolu adresi göstereceği için p^ de 12 olacaktır.

Adres	Bellek içeriği	Açıklaması
.....

100	200	Bellekte p değişkeninin bulunduğu yer
.....
200	12	Bellekte i değişkeninin bulunduğu yer
.....
250	??	Belekte j değişkeninin bulunduğu yer

j:=i*p^ ataması ile bellekte j değişkenine i ile p^ nin değerinin çarpımı atanacaktır. Bellekte i'nin değeri 12 ve p^ nin değeri de 12 olacaktır.

Adres	Bellek içeriği	Açıklaması
100	200	Bellekte p değişkeninin bulunduğu yer
.....
200	12	Bellekte i değişkeninin bulunduğu yer
250	144	Belekte j değişkeninin bulunduğu yer

i:=p^ +1 işleminde ise p^ ile 1 toplanmakta ve i değişkenine atanmaktadır. p^=12 olduğundan sonuç 13 olacaktır.

Adres	Bellek içeriği	Açıklaması
.....
100	200	Bellekte p değişkeninin bulunduğu yer
.....
200	13	Bellekte i değişkeninin bulunduğu yer
.....
250	144	Belekte j değişkeninin bulunduğu yer

Yukarıda görüldüğü gibi i:=p^+1 işlemi i değişkeninin değerini değiştirmektedir. P pointeri de i'nin adresini gösterdiği için o da 13 olacaktır.

Sonuçta değişkenlerin son durumu şöyle olacaktır:

i=13, j=144, p^ =13

Örnek:

Var

J:Integer;

P:^ Integer;

Const

I:Array [0 .. 2] of Integer = (10,20,30);

Begin

P:=@ i[1];

Inc(p);

J:=p^;

ShowMessage(IntToStr(j));

End;

Yukarıdaki işlemlerin sonucunda p ve j değişkenlerinin son değerini bulmaya çalışalım. İşlemi yine bellek düzeyinde düşünerek yapalım. Derleyici değişkenlerimizi aşağıdaki gibi belleğe yerleştirmiş olsun.

Adres	Bellek içeriği	Açıklaması
.....
100	??	Bellekte p değişkeninin bulunduğu yer
.....
200	10	Bellekte i dizisinin bulunduğu yer i[0]
204	20	i[1]
208	30	i[2]
.....
250	??	Bellekte j değişkeninin bulunduğu yer

P:=@ i[1] işlemi ile p değişkenine i dizisinin ikinci elemanı atanmaktadır. p:=@i demiş olsaydık dizinin ilk elemanını göstermiş olacaktı. @i[1] işlemi dizide bir sonraki elemanın adresi demektir.

Adres	Bellek içeriği	Açıklaması
.....
100	204	Bellekte p değişkeninin bulunduğu yer
.....
200	10	Bellekte i dizisinin bulunduğu yer i[0]
204	20	i[1]
208	30	i[2]
.....
250	??	Bellekte j değişkeninin bulunduğu yer

Inc(p) işlemi p pointerinin değerini **Integer** değişkenin uzunluğu kadar arttıracaktır. Yani 204 numaralı adresi gösterirken 208 numaralı adresi gösterecektir.

Adres	Bellek içeriği	Açıklaması
.....
100	208	Bellekte p değişkeninin bulunduğu yer
.....
200	10	Bellekte i dizisinin bulunduğu yer i[0]
204	20	i[1]
208	30	i[2]
.....
250	??	Belekte j değişkeninin bulunduğu yer

J:=p[^] işleminde p'nin değeri 208'dir. P[^] ise 208 numaralı adresteki değeri yani 30 sayısını temsil etmektedir.

Adres	Bellek içeriği	Açıklaması
.....
100	208	Bellekte p değişkeninin bulunduğu yer
.....
200	10	Bellekte i dizisinin bulunduğu yer i[0]
204	20	i[1]
208	30	i[2]
.....
250	30	Belekte j değişkeninin bulunduğu yer

Değişkenlerin son durumu: j=30, p[^]=30 şeklinde olacaktır.

3.7. Yeni Veri Tipleri Tanımlamak

Delphi kullanıcıya kendi veri tipini tanımlamasına müsaade eder. Kullanıcının tanımlayabileceği veri tipleri şunlardır:

- Sayılabilir Tipler(Enumerated types)
- Aralık Sınırlı Tipler(Subrange types)

- Küme Tipleri(Set types)
- Kayıt Tipler(Record types)
- Nesne Tipler(Object types)

3.7.1. Sayılabilir Tipler

Bu tipte değişkenin alabileceği değerlerin belli olduğu durumlarda okuma kolaylığı sağlaması için kullanılır. Bu tip Type bölümünde şu şekilde tanımlanır.

Type

TipAdi: (deger1, deger2,.....degerN);

Bu tanımdan sonra bu tipten değişkenler **Var** kısmında yapılır. Ve bu tipte tanımlanan bir değişkene listedeki değerlerden biri atanabilir.

Type

Bolumler =(Bilgisayar, Muhasebe, Buro, Turizm);

Anarenkler =(Kirmizi, Sari, Mavi);

Gunler=(Pazartesi, Sali, Carsamba, Persembe, Cuma, Cumartesi,Pazar);

Şimdi bu sayılabilir tipler için değişken tanımlayalım:

Var

Bolum: Bolumler;

Renk: Anarenkler;

Gun: Gunler;

Begin

Bolum := Muhasebe;

Gun := Sali;

Renk :=Mavi;

End;

Aslında burada yapılan iş gerçek bir tip tanıımı değildir. Sadece programın okunulabilirliğini kolaylaştırmak için kullanılabilirler.

Örneğin bölümün Bilgisayar olmasını 0, Muhasebe olmasını 1, Büro olmasını 2 ve Turizm olmasını 3 sayısı ile belirtip;

Var

Bolum: Integer;

Begin

Bolum := 1;

End;

ataması da yukarıdaki gibi aynı işi yapabilir. Ancak tip tanımları yapılarak yapılan atamanın bu atamadan daha anlaşılır olduğu bir gerçektir. Zaten bu tip de gerçekte bir sayı tipidir. Yani

Type

Bolumler = (Bilgisayar, Muhasebe, Buro, Turizm);

tanımı ile derleyici Bilgisayar için 0 değerini, Muhasebe için 1, Buro için 2 ve Turizm için 3 değerini kabul ederek atamaları buna göre yapar. Burada parantez içinde verilebilecek maksimum değer sayısı 256 dır. Yani bu tipte tanımlanan değişkenler 1 Byte yer kaplarlar.

Değer atamalarını şu şekilde yapmak mümkündür:

Bolum := Bilgisayar; ile

Bolum := Okul(0); ataması aynı anlamı taşımaktadır.

Ayrıca dizideki bir sonraki elemana ulaşmak için

Bolum := Bilgisayar;

Bolum := Bolum + 1;

ataması doğru değildir. Bunu yapmak için iki yöntem kullanılabilir. Bunlardan biri **Succ** fonksiyonu ile yapmak.

Bolum :=Bilgisayar;

Bolum :=Succ(Bolum);

ataması ile listedeki sonraki bölüme gidilir. Yani Bolum değişkeninin yeni değeri Muhasebe'dir. İkinci yöntem ise tip dönüşümü (Type Casting) yaparak atamadır. **Bolum** değişkeninin 1 integer sayısı ile toplanabilmesi için **Bolum** değişkeninin de Integere çevrilmesi ve sonucun **Bolumler** tipine çevrilmesi gerekir.

Bolum :=Bilgisayar;

Bolum :=Bolumler (Integer (Bolum) +1);

Bu işlem sonucunda da **Bolum** değişkeninin değeri **Muhasebe** olacaktır. İkinci yöntemin avantajı bir sonrakine veya n sonrakine tek bir satırla gidebilmenizdir. **Succ** fonksiyonu ile tek tek ulaşabilirsiniz.

Bolum :=Bilgisayar;

Bolum :=Bolumler (Integer (Bolum) +3);

işlemi ile dizideki turizm bölümüne ulaşabiliriz. Aynı işi **Succ** fonksiyonu ile üç defa yapmak gerekir.

Bolum :=Bilgisayar;

Bolum := Succ(Bolum);

Bolum := Succ(Bolum);

Bolum := Succ(Bolum);

Dizideki bir önceki elemana ulaşmak için de **Pred** fonksiyonu veya ikinci yöntem kullanılabilir.

3.7.2. Aralık Sınırlı Tipler

Bir aralık sınırlı tip **Integer**, **Boolean**, **Char** veya sayılabilir tiplerin alabileceği değer aralığını ifade eder. Bir alt sınırlı tip tanımlamak için tanımlanacak aralığın min ve max değerlerine ihtiyaç duyulur. Tanımlama işlemi yine **Type** kısmında yapılır.

Type

Tip_Ismi=Min .. Max;

Örnek:

Type

DarAci = 0 .. 90;

GenisAci = 90 .. 180;

Dakika = 0 .. 59;

Aylar = 1 .. 12;

Alfabe = 'A' .. 'Z';

Katsayi = 1000 .. 1695;

Var

Acil: DarAci;

Dk: Dakika;

Kats: Katsayi;

Tip tanımından sonra yukarıdaki gibi o tipten değişken tanımlanarak kullanılır. Bu tipteki değişkenlere sınırlar haricinde bir değer ataması yapılamaz.

Kats:=1300; // Doğru

Kats:=200; // Yanlış, sınır dışı

Ancak bir işlemin sonucu bu değerin dışına çıkıyorsa bu kontrol edilmez.

Örneğin:

Kats:=800;

ataması sınır dışı olduğu için kabul edilmez. Ancak dolaylı olarak bu sonuç şöyle alınabilir.

Kats:=1300; Kats:=Kats-500;

Bu işlemin sonucunda Kats:=800 olacaktır.

3.7.3. Küme Tipleri

Küme tipi matematikte bildiğimiz kümelerin karşılığı olan bir tiptir. Ve bu tipte tanımlanan değişkenler üzerinde küme işlemleri yapılabilir. Bir küme **Byte**, **Boolean**, **Char** veya sayılabilir bir tipten olabilir. Küme tanımı **Type** kısmında şöyle yapılır;

Type

Küme_adi = Set Of Tip

Bu tipten değişken tanımı var kısmında yapılır. Daha sonra kümenin elemanları [] içinde belirlenir.

Örneğin:

Type

Kume = set of Byte;

Harfkume = set of 'a' .. 'z';

Turkceharfkume = set of char;

Var

A:Kume;

B:Kume;

h:Harfkume;

th.turkceharfkume;

Begin

A:= [5,6,10,2,4];

B:= [250,123,6];

th:= ['ı', 'İ', 'ü', 'Ü', 'ş', 'Ş', 'ö', 'Ö', 'ğ', 'Ğ', 'ç', 'Ç'];

End;

Bu şekilde kümenin elemanları belirlenebilir.

Bir küme üzerinde yapılabilecek işlemler ve operatörleri şunlardır.

Birleşim (+)

$A + B = A \text{ birleşim } B$

$[5,6,10,2,4] + [250,123,6] = [5,6,10,2,9,250,123]$

Kesişim (*)

$A * B = A \text{ kesişim } B$

$[5,6,10,2,4] * [250,123,6] = [6]$

Fark (-)

$A - B = A \text{ fark } B$

$[5,6,10,2,4] - [250,123] = [5,10,2,4,250,123]$

Kümeler üzerinde yapılabilecek karşılaştırma işlemleri ise şunlardır.

Eşit (=), Eşit Değil (<>)

$A = B$; A kümesi ile B kümesi aynı elemanlara sahipse True üretir.

$A <> B$; A kümesi ile B kümesi aynı elemanlara sahip değilse True üretir.

Kapsama (>=,<=)

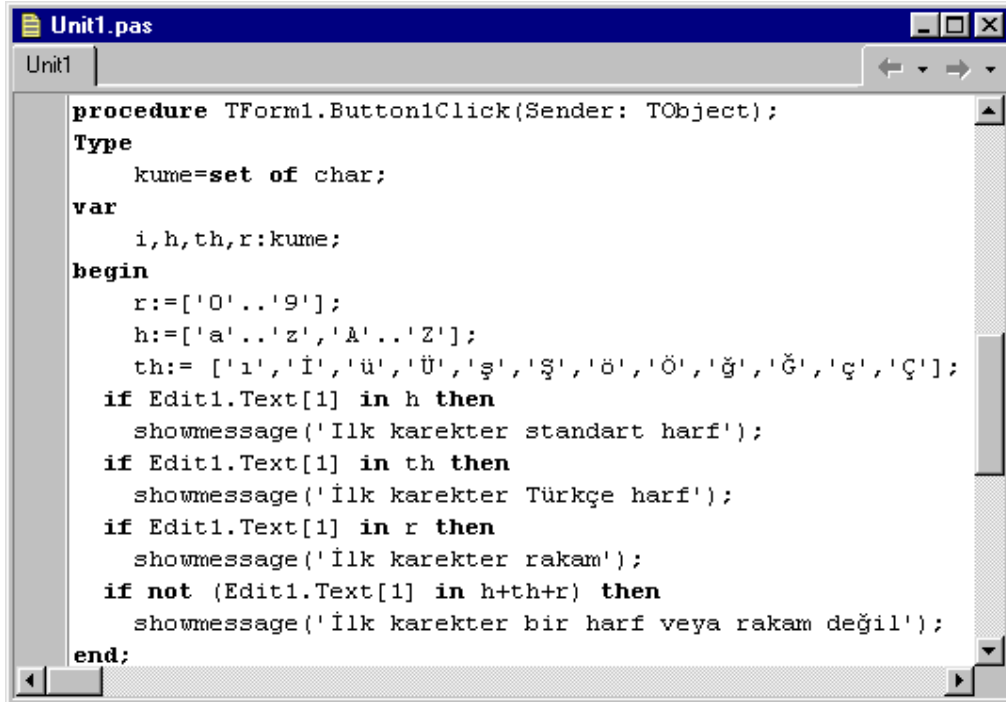
$A <= B$; B kümesi A kümesini kapsıyorsa True üretir. Yani A kümesinin bütün elemanları B kümesinde mevcuttur.

$A >= B$; A kümesi B kümesini kapsıyorsa True üretir. Yani B kümesinin bütün elemanları A kümesinde mevcuttur.

İçinde Bulunma (In)

$x \text{ in } A$; In operatörü A kümesi içinde x mevcutsa True üretir. Yani x'in A kümesinin elemanı olup olmadığını bildirir.

Örneğin bir **Edit** kutusundaki ilk karakterin standart harf, Türkçe harf veya rakam karakterlerinde hangisi olduğunu bulacak bir program yazalım. Örneğimiz için form üzerine bir Edit kutusu ve bir Komut düğmesi yerleştirdikten sonra kod penceresini Şekil 3.16.a'daki gibi düzenledim. Programın örnek çıktısı Şekil 3.16.b'de verilmiştir.



Şekil 3.16. a. Program kod penceresi



Şekil 3.16. b. Programın örnek çıktıları

3.7.4. Kayıt Tipleri

Yukarıda verilen standart tiplerden kendi tipinizi de türetebilirsiniz. Kullanıcı tanımlı kayıt tipleri şöyle tanımlanır:

Type

TipIsmi=Record

Deg_ismi:Tipi

Deg_ismi :Tipi

.....

End;

Bu tipte bir değişken ise şöyle tanımlanır :

Var

Deg_ismi: tip_ismi

Kullanıcının kendi tiplerini tanımlamasının bir çok avantajı vardır. Şöyle bir örnekle önemini anlatmaya çalışalım:

Bir okuldaki öğrencilerle ilgili işlemleri yapacak bir programımız olsun. Bir öğrenciyi tanımlamak için öğrencinin adına, soyadına, doğum yerine, doğum tarihine, numarasına, bölümüne, sınıfına ... ihtiyacımız olacaktır. Bunları standart olarak tanımlarsak,

Var

Ad, Soyad, Dogum_Yeri, Dogum_Tarihi, No, Bolum:String;

Sinif:Integer

Yukarıdaki gibi bir tanım yapabiliriz. Şimdi de veri tabanımıza yeni bir kayıt eklemek istediğimizde bu öğrencinin daha önce listede olup olmadığına bakmamız gerekebilir. Bu durumda öğrencinin adını, soyadını, doğum yerini ve doğum tarihini kayıtlı öğrencilerle karşılaştırmamız gerekir. Bu durumda dört ayrı değişkeni kontrol etmemiz gerekecektir. Dosyaya yazma veya okuma sırasında da, bütün değişkenleri tek tek dosyaya yazmak ve okumak gerekecektir.

Görüldüğü gibi işlemler oldukça uzun ve sıkıcıdır. Bunun yerine öğrenciyi bir tip olarak tanımlayalım:

Type

Ogrenci=Record

Ad, Soyad, Dogum_Yeri, Dogum_Tarihi, No, Bolum: String;

Sinif:Integer;

End;

Burada **Ogrenci** değişkenini değil **Ogrenci** tipini tanımladık. Artık **Ogrenci** yapısında değişkenler tanımlayabiliriz.

Var

Ogr, Ogr2: Ogrenci;

Ogrenci tipinden **Ogr** ve **Ogr2** değişkenleri tanımlandı, bu değişkene atama yapmak için ise "." Operatörünü kullanacağız.

Ogr.Ad := "Ali";

Ogr.Sinif := 2;

.....

Evet artık iki öğrenciyi karşılaştırmak için,

If Ogr = Ogr2 Then

gibi bir komutla tek işlemde yapabiliriz. Dosyaya yazmak veya okumak için de

Write (dosya_deg, Ogr);

Read (dosya_deq, Ogr);

gibi tek satırlık bir işlemle öğrenciyle ilgili bütün değişkenleri dosyaya yazıp okuyabiliriz.

Bu tip değişkenlere atamayı daha kolay yapmak için **With** deyimini şöyle kullanabiliriz.

With Ogr do

Begin

Ad:='Hasan';

Soyad:='Yılmaz';

No:='95530018';

Sinif:=4;

end;

Ayrıca tip tanımında önemli olabilecek bir olay da, tip içinde tanımlanan string tipindeki değişkenlerin sınırlarının verilmesidir. Bu tanımladığınız tipin byte olarak büyüklüğünün sabit olmasını sağlar, bu da dosyaya yazma ve okuma işlemlerinde kolaylık sağlayacaktır. Öğrenci tipimizi bu şarta göre yeniden tanımlayalım:

Type

Ogrenci=Record

Ad, Soyad, Dogum_Yeri: String [20];

No: String [10];

Sinif: Integer;

End;

Yukarıda tanımladığımız tip ile bu tip arasında yapılan işlemler açısından bir fark yoktur. Ancak son olarak tanımladığımız şekilde Öğrenci tipinin uzunluğu sabit olacaktır.

BÖLÜM 4

BİLGİ GİRİŞ ve MESAJ PENCERELERİ

Delphi kullanıcıyı herhangi bir konuda bilgilendirmek, herhangi bir işlem girişiminde kullanıcının onayını almak, bazı konularda kullanıcının dikkatini çekmek için mesaj pencereleri sunar. Bunların bir kısmı Delphi tarafından sağlanan bir kısmı da Windows tarafından sağlanan standart mesaj pencereleridir. Bunun yanında kullanıcının bilgi girişi yapabilmesi bilgi giriş fonksiyonu mevcuttur.

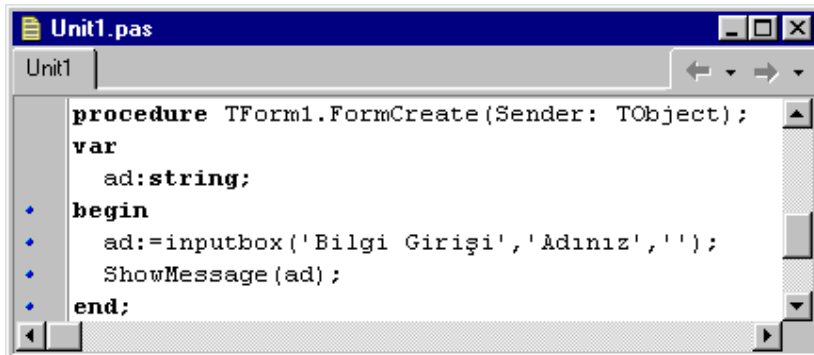
4.1. InputBox () Fonksiyonu

InputBox() fonksiyonu, Delphi projesinin aktif form veya penceresinden bağımsız olarak bir diyalog kutusu içinde kullanıcının dışarıdan bilgi girmesine imkan sağlar. InputBox() fonksiyonu dışarıdan bilgi olarak üç parametre almaktadır.

InputBox(Diyalog Kutusu Başlığı, Metin Kutusu Başlığı, Varsayılan Bilgi)

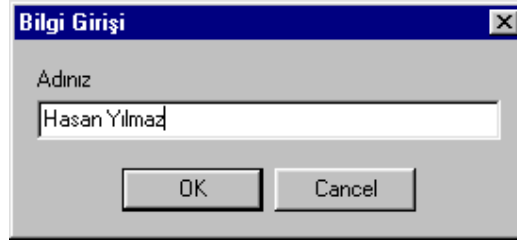
İlk parametre ile diyalog kutusunun başlığı belirtilmektedir. İkinci parametre ile bilgi girişi için diyalog kutusunda hazırlanan metin kutusunun başlığı ve üçüncü parametre ile metin kutusuna varsayım olarak getirilmek istenen bilgi belirtilmektedir. InputBox() fonksiyonun nasıl kullanıldığını anlatmak için üzerinde çalıştığım projenin formu üzerinde çift tıklama yaparak program kodu yazılan pencereyi ekrana getirdim.

Tasarım anında formun üzerinde çift tıklama yapınca hem program kodu girişi yapılabilen pencere ekrana geldi hem de FormCreate adında bir yordam (procedure) hazırlandı. FormCreate yordamındaki program satırları proje çalıştırıldığı zaman otomatik olarak işletilmektedir. Proje çalıştırıldığı zaman bilgi girişi için ekrana bir diyalog kutusunun getirilmesini sağlamak için FormCreate yordamını Şekil 4.1'deki gibi düzenledim.



Şekil 4.1 InputBox() Fonksiyonu

Bu örnekte önce InputBox() fonksiyonu aracılığı ile dışarıdan girilecek bilginin aktarılacağı “Ad” değişkeni **Var** bildiri deyimi ile tanımlanmaktadır. **Begin** deyiminden sonra ilk satırda InputBox() fonksiyonu ile “Ad” değişkenine bilgi aktarılmaktadır. Programın işletimi InputBox() fonksiyonunun olduğu satıra geldiği zaman aşağıdaki gibi bir diyalog kutusu ekrana getirilir.



Şekil 4.2

Bu diyalog kutusundaki “Adınız” başlıklı metin kutusuna istediğiniz gibi bilgi girebilirsiniz. Bu metin kutusuna genişliğinden daha fazla bilgi girebilirsiniz. Bilgi girişini tamamladıktan sonra OK düğmesine tıklama yaparsanız, InputBox() fonksiyonu metin kutusuna girdiğiniz bilgiyi geriye döndürür. OK düğmesine tıklamak yerine doğrudan Enter tuşuna basabilirsiniz. Eğer girilen bilginin geri gönderilmesini istemiyorsanız Cancel düğmesine veya Esc tuşuna basmalısınız. Esc tuşuna basarak diyalog kutusunu ekrandan kaldıracak olursanız InputBox() fonksiyonu geriye boşluk değerini döndürür.

InputBox() fonksiyonu ile ekrana getirilen diyalog kutusundaki metin kutusuna hiçbir şey yazmadan diyalog kutusunu **OK** düğmesi ile veya **Cancel** düğmesi ile kaparsanız programda kullandığımız ShowMessage() deyiminden dolayı Şekil 4.3.a’deki gibi bir mesaj penceresi ile karşılaşırız. Eğer diyalog kutusundaki metin kutusuna bilgi girişi yaptıktan sonra **OK** düğmesine basarsak Şekil 4.3.b’deki gibi bir mesaj penceresi ekrana gelir.



Şekil 4.3.a. Bilgi girişi yapılmış

4.2. ShowMessage() Deyimi

Kullanıcıya herhangi bir anda sadece bilgi vermek için **ShowMessage** kullanılır.

ShowMessage(Mesaj);

ShowMessage prosedürü sadece bir Ok düğmesi içeren bir mesaj penceresi görüntüler. Çalışan uygulamanın adı mesaj penceresinin başlığını oluşturur.

Örneğin program içinde **ShowMessage('İyi Günler');** şeklinde bir kod yazarsak programımız bu kodu işlettikten sonra Şekil 4.4'teki gibi bir mesaj penceresi görüntüler.



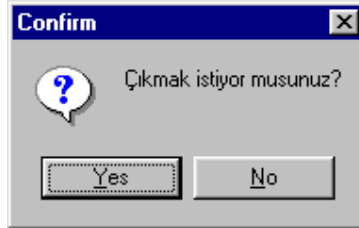
Şekil 4.4.

ShowMessagePos(Mesaj, x, y);

ShowMessagePos prosedürünün, ShowMessage prosedüründen tek farkı mesaj penceresini ekranın ortasında değil verilen x,y koordinatında görüntülemesidir.

4.3. MessageDlg Fonksiyonu

MessageDlg fonksiyonu ekranın ortasında bir mesaj penceresi görüntüler. Bu pencere ShowMessage prosedürü ile oluşturulan mesaj penceresinden farklı olarak kullanıcıdan onay alınması gereken durumlarda kullanılır.



Şekil 4.5. Bir programdan çıkışta kullanılan mesaj penceresi

Secilen := MessageDlg(Mesaj; Tip; Düğme; HelpCtx)





Secilen := MessageDlgPos(Mesaj; Tip; Düğme; HelpCtx,x,y)

Bu pencerede standart Delphi düğmeleri ve ikonları yer alır. Pencerenin içindeki mesaj, düğmeler ve ikonlar şu parametrelerle belirlenir:

Mesaj: Pencerede çıkacak olan mesajı belirler.

Tip: Mesaj penceresinin tipini belirler. Yani gösterilen pencerenin uyarı, bilgilendirme, dikkat vb. ikonlarından birini içerdiğini belirtir.

Buna göre Tip parametresinin aldığı değerler şunlardır:

İkon İsmi	İkon Şekli
MtWarning	
MtError	
mtInformation	
mtConfirmation	

Düğme: Pencerede çıkacak olan düğmeyi belirler. Düğme parametresinin alabileceği değerler şunlardır: mbYes, mbNo, mbOK, mbCancel, mbAbort, mbRetry, mbIgnore, mbAll, mbHelp.

MessageDlg fonksiyonunda düğmelerin hangilerinin bulunacağı köşeli parantez '[']' arasında verilir.

Örneğin :[mbYes] veya [mbYes, mbNo, mbCancel] gibi.

Ayrıca birden fazla düğmeyi görüntülemek için **mbYesNoCancel**, **mbOkCancel**, **mbAbortRetryIgnore** kullanılabilir.

HelpCtx: Yardım dosyasındaki ilgili konu numarası. Bu numara yardım dosyası tasarlanırken belirlenir ve yardım tuşuna basıldığında bu numaralı konu **Winhelp** programı aracılığı ile görüntülenir.

MessageDlg fonksiyonu kullanıcının seçtiği düğmeyi bildirmek için aşağıdaki değerleri gönderir

Seçilen Düğme	Dönen Değer
Yes	MrYes
No	MrNo
OK	MrOK
Cancel	MrCancel
Abort	MrAbort
Retry	MrRetry
Ignore	MrIgnore
All	MrAll

Örnek olarak programdan çıkarken kullanıcının onayını almak isteyelim. Programdan çıkarken formun **OnClose** olayı çalışacağı için kodumuzu buraya yazacağız.

```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
```

```
Var
```

```
    c:Word;
```

```
Begin
```

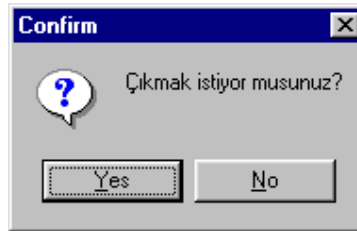
```
    c:=MessageDlg('Çıkmak istiyor musunuz?',MtConfirmation,[mbYes,mbNo],0);
```

```
    if c=mrNo then //No seçildiyse
```

```
        Action:=caNone; //Çıkışı iptal et
```

```
End;
```

Yukarıdaki program kodlarını yazarak programımızı çalıştırdıktan sonra çıkış için **program reset** komutunu işlettiğimizde aşağıdaki mesaj penceresi ekrana gelir.



Şekil 4.6.

4.4. Standart Mesaj Pencereleeri

Windows mesaj kutuları konusunda programlara standart pencereler sunar. Bunu İngilizce bir programda çıkan Türkçe mesajlardan fark etmişsinizdir. Siz de Delphi'nin sağladığı pencereler yerine standart pencereleri kullanmak istiyorsanız **MessageBox** metodunu kullanmalısınız. Genel yazılışı aşağıdaki gibidir.

Secilen := Application.MessageBox(Mesaj; Başlık; Tip)

Mesaj: Gösterilecek mesajı ifade eder.

Başlık: Mesaj penceresinin başlığını ifade eder.

Tip: Mesaj penceresinde gösterilecek düğme, icon, varsayılan düğme ve öncelik sırasını temsil eder. Buna göre;

Tip= Düğme + Icon+ Varsayılan Düğme+ Öncelik Sırası

şeklinde ifade edersek buradaki her parametrenin alabileceği değerler ve anlamları da şöyledir:

	Sayı	Sembolik	Anlamı
Düğme	0	MB_OK	Tamam
	1	MB_OCANCEL	Tamam ve İptal
	2	MB_ABORTRETRYIGNORE	İşlemi Durdur, Yeniden Dene, Gözardı Et
	3	MB_YESNOCANCEL	Evet, Hayır, İptal
	4	MB_YESNO	Evet, Hayır
	5	MB_RETRYCANCEL	Yeniden Dene, İptal
Icon	16	MB_ICONSTOP	Dur
	32	MB_ICONQUESTION	Soru
	48	MB_ICONEXCLAMATION	Dikkat
	64	MB_ICONINFORMATION	Bilgi
Varsayılan Düğme	0	MB_DEFBUTTON1	İlk Düğme
	256	MB_DEFBUTTON2	İkinci Düğme
	512	MB_DEFBUTTON3	Üçüncü Düğme
Öncelik Sırası	0	MB_TASKMODAL	Sadece çalışan program cevap verilene kadar askıya alınır
	4096	MB_SYSTEMMODAL	Cevap verilene kadar çalışan bütün programlar askıya alınır

Mesaj kutusu açıldığı anda kullanıcı bir seçim yapana kadar program çalışmayı durdurur, Öncelik sırası yerine **MB_SYSTEMMODAL** verilirse sadece size ait program değil bütün programların çalışması durdurulur.

Varsayılan düğme parametresi de kullanıcının yanlış seçim yapmasını önlemek için açılan pencerede hangi düğmenin aktif olduğunu belirler. Örneğin önemli bir dosyayı silmek için kullanıcıdan onay almak için açtığınız pencerede varsayılan düğmeyi **Hayır** düğmesi yaparak kullanıcının yanlışlıkla silmesini önleyebilirsiniz.

Geri dönen değere bakılarak hangi düğmenin tıklandığına karar verilir. Buna göre aşağıda seçilen düğmeler ve dönen değerler şöyledir.

Dönen Değer	Seçilen Düğme
IDOK	Tamam
IDCANCEL	İptal
IDABORT	İşlemi Durdur
IDRETRY	Yeniden Dene
IDIGNORE	Gözardı Et
IDYES	Evet
IDNO	Hayır

Delphi’de hem sembolik isimler hem de sayısal değerler kullanılabilir. Örneğin Evet-Hayır-İptal düğmeleri ve ? ikonu bulunan bir pencere oluşturmak için

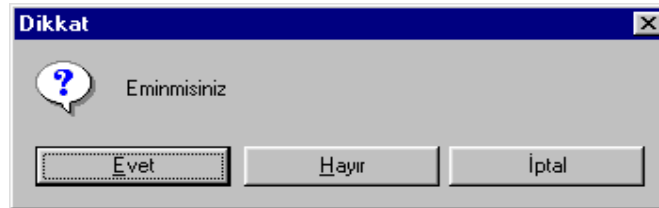
```
Application.MessageBox( 'Eminmisiniz', 'Dikkat', MB_YESNOCANCEL +  
MB_ICONQUESTION);
```

şeklini kullanabileceğimiz gibi,

```
Application.MessageBox( 'Eminmisiniz', 'Dikkat', 3 + 32);
```

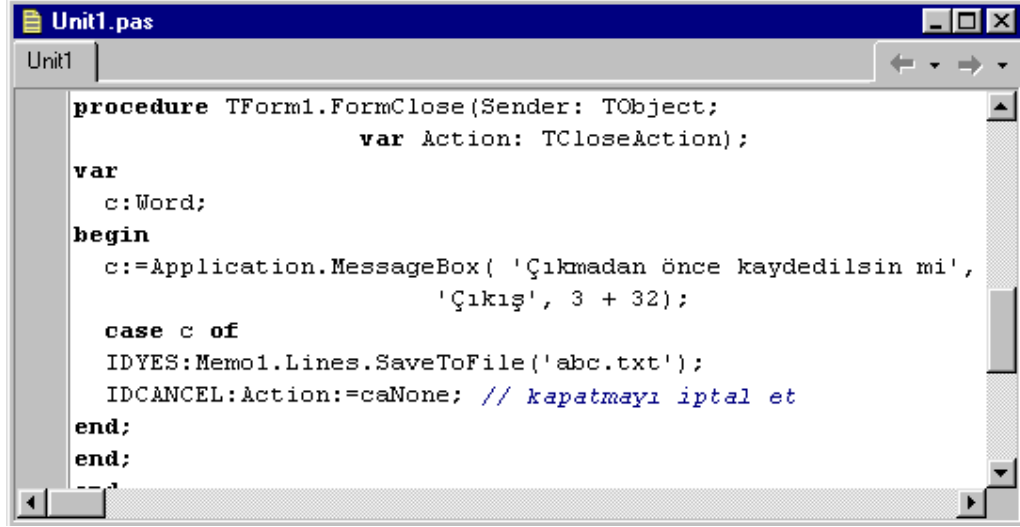
formatını da kullanabiliriz. Birinci yöntem daha anlaşılırken ikinci yöntem yazım kolaylığı sağlamaktadır.

Bu komutlarla oluşacak pencere aşağıdaki gibi olacaktır.



Şekil 4.7. Örnek mesaj penceresi

Örneğin programdan çıkarken “kaydedilsin mi?” diye soracak ve Evet-Hayır-İptal seçenekleri bulunacak bir mesaj kutusu ile kullanıcının onayını alalım. Programdan çıkarken Formun OnClose olayı meydana geleceği için kodumuzu bu olaya yazacağız. Program satırları Şekil 4.8’deki gibi olacaktır.



Şekil 4.8. Çıkış için yazılan program kodları

Programdan çıkarken Formun **OnClose** olayı meydana gelecektir. Kullanıcıdan aldığımız onay; **Evet** ise **Memo1** kontrolü içindeki bilgiyi dosyaya yazacak kodu yazıyoruz. **Hayır** ise herhangi bir koda gerek kalmaz. Çünkü form zaten kapatılıyor. **İptal** ise formun kapatılmasını durdurmamız gerekir. Bunu da **Action** değişkenine atayacağımız **caNone** değeri yapar.



Şekil 4.9. Programdan çıkışa ait mesaj penceresi

BÖLÜM 5

PROGRAM KONTROL DEYİMLERİ

5.1. Şartlı Çalışma Deyimleri

Programlar normalde satır satır çalışırlar. İstenirse belirli şartlar aranarak programın bir kısmının çalıştırılmasını veya çalıştırılmamasını sağlayabiliriz.

5.1.1. If Yapısı

If- şart yapısı bütün programlama dillerinde olan, bazı şartların gerçekleşmesi durumunda ve gerçekleşmemesi durumunda ayrı-ayrı kodların çalıştırılmasına imkan veren yapıdır.

If Şart Then

Komutlar

Else

Komutlar;

Şartın gerçekleşmesi halinde **Then** deyiminden sonraki satır işletilir. Gerçekleşmemesi durumunda ise **Else** deyiminden sonraki satırlar işletilir. Şartın gerçekleşmemesi durumunda çalışacak Else bloğu istenirse verilmeyebilir.

If Şart Then

Komutlar;

Buradaki Komutlar kısmı birden fazla komut içeriyorsa *Begin-End* deyimleri kullanılmalıdır. Tek satırlık ifadelerde **Begin-End** deyimini kullanılmaz.

If Şart Then

Komut

Else

Komut;

Birden fazla satır kullanılacaksa **Begin-End** deyimleri kullanılır.

If Şart Then

Begin

Komutl;

Komut2:

....

End;

Else

Begin

Komut1;

Komut2;

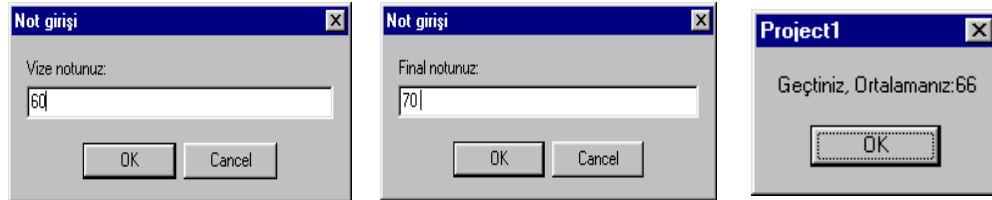
End;

Aşağıda, Şekil 5.1.a’da If – Then – Else yapısının kullanımına ait, girilen vize, final, ve bütünleme notlarına göre not ortalamasını hesaplayan bir program, Şekil 5.1.b’ de ise program çalıştırıldıktan sonra alınan ekran görüntüleri verilmiştir.

```
Unit1.pas
Unit1

procedure TForm1.Button1Click(Sender: TObject);
var
    v,f,b,ort:real;
begin
    v:=StrToInt(InputBox('Not girişi','Vize notunuz:', ''));
    f:=StrToInt(InputBox('Not girişi','Final notunuz:', ''));
    ort:=v*0.4+f*0.6;
    //Ort 49.5 den küçükse veya final 50 den küçükse bütünleme sor
    if (Ort<49.5) or (f<50) Then
    Begin
        b:=StrToInt(InputBox('Not girişi','Bütünleme notunuz:', ''));
        ort:=v*0.4+b*0.6;
    End;
    if Ort>49.5 then
    ShowMessage('Geçtiniz, Ortalamanız:'+FloatToStr(ort))
    Else
    ShowMessage('Kaldınız, Ortalamanız:'+FloatToStr(ort));
end;
```

Şekil 5.1.a. If –Then – Else yapısının kullanımı



Şekil 5.1.b. Programın çıktıları (Soldan sağa doğru sırasıyla verilmiştir.)

5.1.2. Case Yapısı

Bir değişkenin aldığı bir çok değere göre ayrı komutların çalıştırılması gereken durumlar için **If** yapısını kullanmak yerine **Case** yapısını kullanmak daha avantajlıdır.

Case Değişken Of

Durum1:Komutlar;
Durum 2, Durum3:Komutlar;
Durum4 .. Durum7:Komutlar;
.....

Durum N:Komutlar;
Else Komutlar;

End;

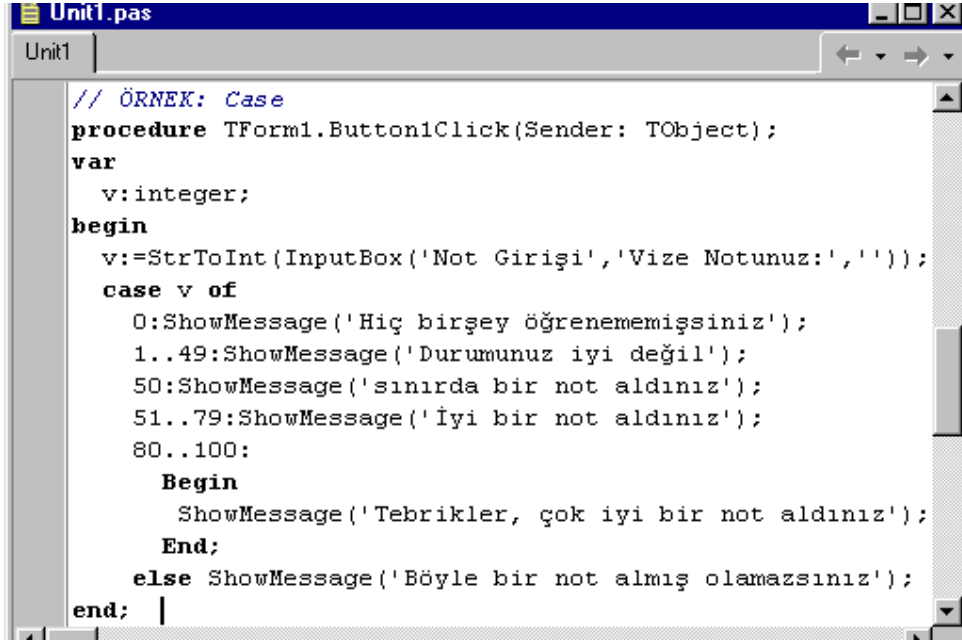
Burada **Değişken** parametresi ile belirlenen değişkenin aldığı duruma göre **DurumN** değerinin karşısındaki komutlar işleme girer. Eğer değişkenin değeri durumlardan hiçbirine uymuyorsa **Else** kısmındaki komutlar çalışacaktır.

Case x of

1: // x= 1 iken yazılacak kod
2, 3, 5: // X= 2, 3 veya 5 iken yazılacak kod
6 ..10: // X= 6, 7, 8, 9, 10 iken yazılacak kod
else: // Bunlardan biri değilse yazılacak kod

End;

Case yapısındaki durumu karşılaştıran değişken **string** tipte veya kullanıcı tipte bir değişken olamaz. Bu değişken değeri sayı olarak ifade edilebilen bir değişken olmak zorundadır.



```
// ÖRNEK: Case
procedure TForm1.Button1Click(Sender: TObject);
var
  v:integer;
begin
  v:=StrToInt(InputBox('Not Girişi','Vize Notunuz:', ''));
  case v of
    0:ShowMessage('Hiç birşey öğrenememişsiniz');
    1..49:ShowMessage('Durumunuz iyi değil');
    50:ShowMessage('sınırdan bir not aldınız');
    51..79:ShowMessage('İyi bir not aldınız');
    80..100:
      begin
        ShowMessage('Tebrikler, çok iyi bir not aldınız');
      end;
    else ShowMessage('Böyle bir not almış olamazsınız');
  end;
```

Şekil 5.2. Case yapısının kullanımına ait örnek program kod penceresi

5.2. Döngü Deyimleri

Belirli bir komut parçasının belli şartlar gerçekleşinceye kadar defalarca çalışması gerekebilir. Çalışma sayısının belli olduğu durumlarda **For** döngüsü diğer durumlarda yani döngü sayısının değişken olduğu durumlarda ise **Repeat-Until**, **While-Do** blokları kullanılır.

5.2.1. FOR Döngüsü

For *Sayaç:=BaşlangıçDeğeri to BitişDeğeri do*
Begin
 komutlar;
End;

Burada **Sayaç** değişkeni tamsayıya çevrilebilen tipte (**Integer**, **LongInt**, **ShortInt**, **Byte**, **Word**, **Boolean**, **Char**) bir değişken olması gerekir. **For** döngüsü **Sayaç**'ın **BaşlangıçDeğeri**'nden başlayarak **BitişDeğeri**'ne kadar sayacı birer arttırarak blok içindeki komutları çalıştırır.

BaşlangıçDeğeri, **BitişDeğeri**'nden küçükse döngüye hiç girilmeyecektir.

Sayaç'ın artarak değil azalarak çalışması için **to** yerine **downto** deyimini kullanılır.

For Sayaç:= BaşlangıçDeğeri downto BitişDeğeri do

Begin

Komutlar;

End;

BaşlangıçDeğeri, **BitişDeğeri**'nden büyükse döngüye hiç girilmeyecektir.

Örnek olarak 100 kişinin ismini sormamız gerektiğini farz edelim. Bu durumda:

Var

ad: array [1 ..100] of String;

Begin

ad[1]:=InputBox ('Ad girişi', '1. kişinin adı', '');

ad[2]:=InputBox('Ad girişi', '2. kişinin adı', '');

ad[3]:=InputBox('Ad girişi', '3. kişinin adı', '');

.....

End;

Yukarıdaki gibi 100 satırlık kod yazmak yerine 1 den 100'e kadar bir döngü kurarak aynı işi çok daha kolay yapabiliriz.

Var

ad:array [1 ..100] of String;

i:Integer;

Begin

For i:=1 to 100 do

ad[i]:=InputBox('Ad girişi', IntToStr(i) + ' . kişinin adı', '');

end;

Yukarıdaki örnekte for döngüsünün altında bir satırlık kod yazdığım için **Begin-End** bloğunu kullanmadım.

Örnek olarak ekranda rastgele koordinatlara iç içe dikdörtgenler çizecek bir program yazalım. Programımız için formun üzerine bir komut düğmesi yerleştirdikten sonra click yordamına aşağıdaki program kodlarını yazalım.

ÖRNEK : For – DownTo

```
procedure TForm1.Button1Click(Sender: TObject);
Var
    mx,my,i:Integer;
Begin
    Randomize;
    mx:=Trunc(random*Form1.Width);
    my:=Trunc(random*Form1.Height);
    for i:=10 downto 0 do
        Canvas.Rectangle(mx+i*10, my+i*10, mx-i*10, my-i*10);
    End;
```

Başka bir örnek olarak ta kullanıcının gireceği 3x3 boyutlu iki matrisi toplayıp, çarpacak bir program yapalım. Örneğimiz için Şekil 5.3'deki formu oluşturalım.

Şekil 5.3. Oluşturulacak Form

Yukarıdaki formu oluşturduktan sonra kod penceresinde aşağıdaki kodları yazarak programımızı **F9** tuşuna basarak çalıştıralım. Programın çıktısı Şekil 5.4'deki gibi olacaktır.

```
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.Button3Click(Sender: TObject);
var
    i,j:Integer;
begin
    Memo1.Text:=' '; {A Matrisini sor}
    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) + '. elemanın değeri:', '1'));
                    Memo1.Text:=Memo1.Text+' '+IntToStr(a[i,j]);
                end;
            Memo1.Text:=Memo1.Text+#13#10 {Satırbaşı ekle}
        end;
    end;

    procedure TForm1.Button4Click(Sender: TObject);
    var
        i,j:integer;
    begin
        Memo2.Text:=' '; {B Matrisini sor}
        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) + '. elemanın değeri:', '1'));
                    end;
                end;
            end;
        end;
    end;

```



```

    Memo2.Text:=Memo2.Text+' '+ IntToStr(b[i,j]);
end;
Memo2.Text:=Memo2.Text+#13#10 {Satır başı ekle}
end;
end;
procedure TForm1.Button1Click(Sender: TObject);
var
    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.Button2Click(Sender: TObject);
Var
    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.

```

Şekil 5.4. Toplama işlemi için programın çıktısı

For döngüsünde sayaç değişkeni birer artıp/azalarak değişir. Artımın daha büyük aralıklarla yapılması için artırma işlemi blok içerisinde **Sayaç** değişkeninin değeri arttırılarak yapılamaz.(1.0 versiyonunda yapılabilir). Bu gibi durumlarda **While – Do** döngüsü kullanılmalıdır.

5.2.2. While – Do Döngüsü

Bir şart gerçekleştiği sürece çalışması gereken program bloklarında kullanılır.

While Şart Do

Begin

Komutlar;

End;

Burada; şart gerçekleştiği sürece döngü çalışmaya devam edecektir. For döngüsünün getirdiği bazı kısıtlamalar vardır. Sayaç değişkenimiz tamsayı olmak

zorunda ve birer artıp/azalmak zorunda idi. Bu sınırlamaları aşmak için **While – Do** döngüsünü **For** döngüsüne uyarlayabiliriz.

For Sayaç:=BaşlangıçDeğeri to BitişDeğeri do

Begin

Komutlar;

End;

Aynı işi While – Do ile yaparsak;

Sayaç:=Başlangıç;

While Sayaç<=BitişDeğeri do

Begin

Komutlar;

sayaç:=sayaç+1;

End;

Artımı birden fazla yapmak içinde

sayaç:=sayaç+1; yerine

sayaç:=sayaç+art; kullanabiliriz.

While TRUE do

Begin

.....

If koşul Then

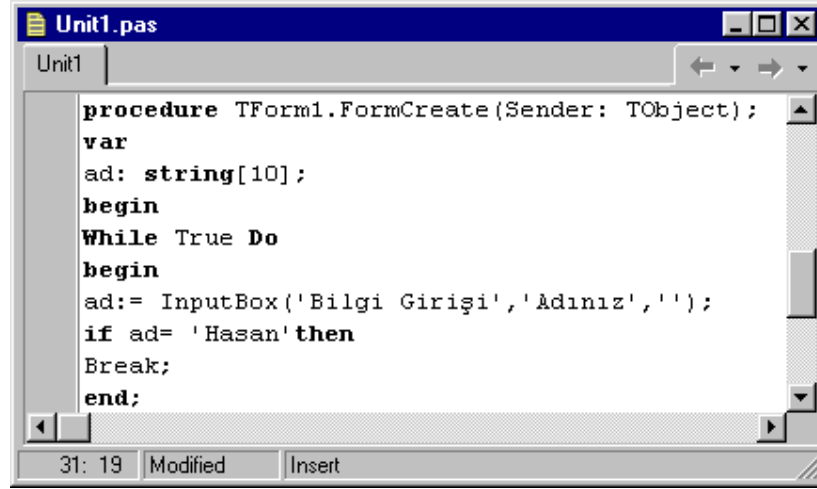
Break;

End;

Kullanılması isteğe bağlı olan **Break** yardımcı deyimini döngüden belirtilen şarta bağlı kalmadan çıkmayı sağlar. Genel alışkanlıkla yalnızca Break deyimini ile çıkılacak şekilde oluşturulan While – Do döngülerinde “şart” olarak mantıksal sabit bir değer veya değişken kullanılır.

Yukarıda verilen While – Do döngüsünün her turunda If değimi ile “koşul” test edilir. Eğer koşul mantıksal doğru (True) değerini içeriyorsa programın işetimi döngüden çıkılarak blok sonunu işaret eden End deyimini izleyen satıra geçer.

Döngü kontrol ifadesi olarak mantıksal sabit yani True kullanılması halinde döngüden **Break** deyimi ile çıkılmadığı sürece döngü sonsuz olur. Bu nedenle yukarıdaki gibi döngülerde mutlaka döngüden çıkabilmek için belirli şartlara bağlı olarak **Break** deyiminin işletilmesi sağlanmalıdır.



Şekil 5.5. While – Do döngüsü

5.2.3. Repeat – Until Döngüsü

Bu döngü yapısı da şart gerçekleşene kadar çalışması gereken program bloklarında kullanılır.

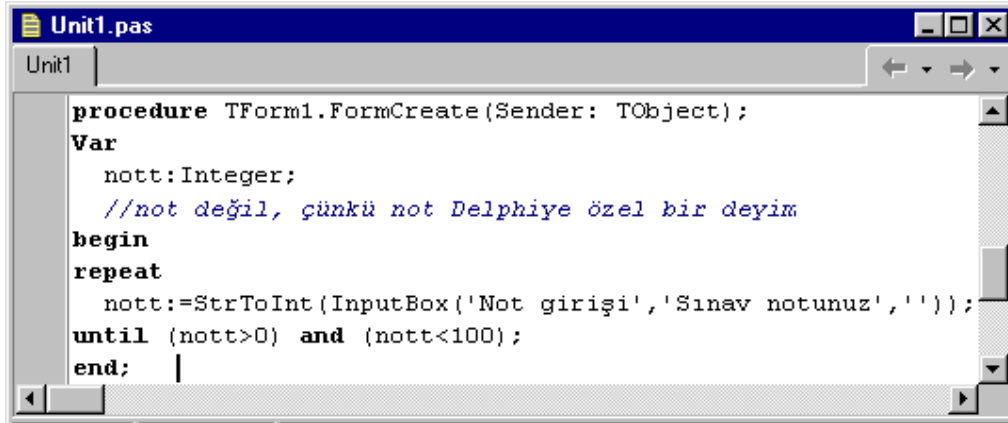
Repeat

Komutlar;

Until Şart;

While – Do döngüsünden farklı olarak döngüye girerken değil çıkarken şart kontrol edilir. Böylece Repeat – Until döngüsü içindeki komutlar en az bir kere çalışır.

Örneğin kullanıcıya sınav notunu sorduğumuzu düşünelim. Gireceği not 0 – 100 aralığı dışında ise; notu bu aralıkta girinceye kadar tekrar tekrar sorulması gerekir. Bu iş için **Repeat – Until** yapısı uygundur. Çünkü döngü içerisinde not sorulduktan sonra not 0-100 aralığı dışında ise tekrar sorulması gerekir. Bununla ilgili örnek Şekil 5.6'da verilmiştir.



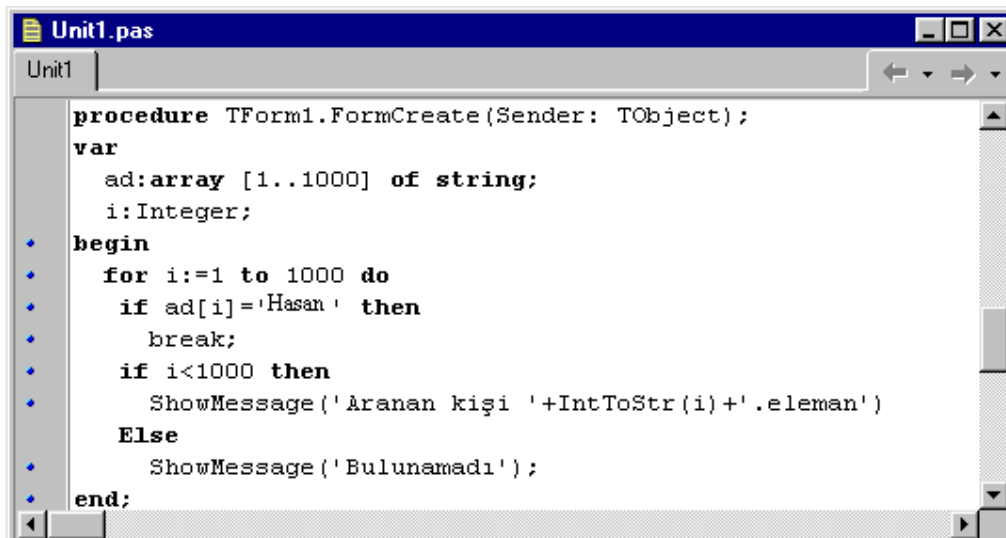
5.3. Döngü Kontrol İfadeleri

Bazen döngü bitmeden döngüden çıkmak gerekebilir veya bazı durumlarda döngü içindeki bir miktar kodun çalıştırılmaması istenebilir. Bu gibi durumlarda döngü kontrol deyimlerini kullanmak gerekir.

5.3.1. Break

Break prosedürü; **For**, **While** veya **Repeat** döngülerinden birinde bazı şartların gerçekleşmesi durumunda döngüden çıkmak için kullanılır.

Örnek olarak 1000 kişinin bulunduğu `ad[1000]` dizisinde bir kişinin adını **for** döngüsü ile aradığımızı kabul edelim. Aranılan kişi bulunduktan sonra döngünün çalışmaya devam etmemesi için **break** kullanılması gerekir.



Şekil 5.7. Break deyim

5.3.2. Continue

Continue prosedürü; **For**, **While** veya **Repeat** döngülerinde bazı şartlar gerçekleştiğinde döngünün sonuna gitmeden tekrar başa dönmesini sağlar.

Var

i,s:Integer;

Begin

s:=0;

For i:=1 to 100 do

Begin

If odd(i) Then

Continue;

s:=s+i;

End;

ShowMessage(IntToStr(s));

End;

Burada normalde 1 den 100'e kadar olan sayıları toplamalı gerekir. Ancak şart satırımızda sayı tek ise başa dön diyerek toplama işlemini yapmadan devam etmesini sağlıyoruz. Böylece program 1'den 100'e kadar olan çift sayıları toplayacaktır. Çift sayıları **odd()** deyimi ile tespit ediyoruz.

5.3.3. Exit

Exit prosedürü mevcut program bloğundan, bloğun sonuna ulaşmadan çıkmaya yarar. Çalışan blok bir prosedür ise prosedürden, bir döngü veya şart bloğu ise o bloktan çıkar.

procedure TForm1.Button1Click(Sender: TObject);

Var

i,j:integer;

Begin

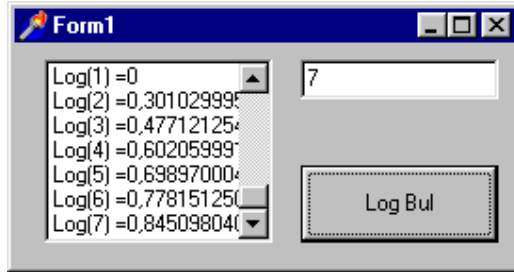
j:=StrToInt(Edit1.Text);

If j<=0 Then exit;

```

For i:=1 to j do
    ListBox1.Items.Add('Log('+IntToStr(i)+' ) =' +FloatToStr(Ln(i) / Ln(10)));
End;

```



Şekil 5.8.

Burada da 1'den, **Edit** kutusuna girilen sayıya kadar olan sayıların logaritmasını listeye ekliyoruz. Ancak kullanıcı kutuya 0 veya daha küçük bir sayı girmişse prosedürden çıkıyoruz.

5.3.4. Halt

Halt prosedürü programdan çıkışı sağlar. Burada istenirse bir parametre ile çıkış kodu programdan geriye dönebilir.

5.4. With – Do Deyimi

Herhangi bir kontrol elemanının birden fazla özelliğini değiştirmek yada metotlarına ulaşmak için **With – Do** deyimi kolaylık sağlar. Kullanım biçimi şöyledir.

With KontrolAdı Do

Begin

.....

End;

Artık bu blok içerisinde o kontrolün özelliklerine kontrolün ismi kullanılmadan ulaşılabilir.

```
procedure TForm1.Button1Click(Sender: TObject);
```

Begin

```
    Combobox1.clear;
```

```
    Combobox1.items.add('T.E.F.');
```

```

Combobox1.items.add('Bilgisayar');
Combobox1.items.add('Öğretmenliği');
Combobox1.items.add('Bölümü');
Combobox1.Font.Style:=[fsUnderline];
Combobox1.Font.Name:='Courier New';
Combobox1.ScaleBy(150,100);
End;

```

Yukarıdaki program kodunu **With – Do** deyimini kullanarak şu şekilde yazabiliriz:

```

procedure TForm1.Button1Click(Sender: TObject);
Begin

```

```

    With combobox1 do
    Begin
        clear;
        items.add('T.E.F. ');
        items.add('Bilgisayar');
        items.add('Öğretmenliği');
        items.add('Bölümü');
        Font.Style:=[fsUnderline];
        Font.Name:='Courier New';
        ScaleBy(150,100);
    End;

```

```

End;

```

Böylece blok içerisinde, her seferinde kontrolün ismini kullanmak zorunda kalmıyoruz.

Aynı işlem **Record** type olarak tanımlanmış değişkenlerin alt değişkenlerine ulaşılırken de geçerlidir.

BÖLÜM 6

PROSEDÜR ve FONKSİYONLAR

Prosedür ve fonksiyonlar program kodlarının modüler kısımları olup, özel görevler üstlenirler. Bunlar aynı zamanda “**Altprogram**” olarak ta bilinirler. Delphi, uygulamaları geliştirmek için çok sayıda hazır prosedür ve fonksiyonu kullanıcının hizmetine sokar. Her biri özel bir amaca yönelik olan bu prosedür ve fonksiyonları Delphi uygulamalarında kullanmak için onları çağırmak gerekir.

Prosedüre ve fonksiyonlar nesnelerle birlikte kullanıldığı zaman **Methods** adını alırlar. Fonksiyon, Procedure’den farklı olarak geriye bir değer gönderebilir. Bu değer gönderme işlemi fonksiyon ismine geri dönecek değerin atanmasıyla olur.

6.1. Prosedür

Bir prosedür şu şekilde tanımlanır.

Procedure ProsedürAdı (GirişParametreleri:Tipi);

Sabit,değişken, Tip tanımı

Begin

Program Kodları;

[Exit;]

End;

ProsedürAdı: Prosedüre verilecek addır. Prosedür bu isim kullanılarak tanımlanır. Değişken tanımı için geçerli olan kısıtlamalar bu ad için de geçerlidir.

GirişParametreleri: Prosedüre, çağrıldığı yerden gönderilen bilgilerdir. Araya virgül konarak birden fazla giriş parametresi tanımlanabilir. Parametrenin tanımından sonra o parametrenin tipi iki nokta işaretinden sonra belirtilir.

Sabit, Değişken, Tip Tanımı: Prosedürün kullanacağı değişkenler, sabitler veya tipler bu blokta tanımlanır. Bu blokta tanımlanan değişkenler **local** değişkendir. Yani bu değişkenleri sadece bu prosedür kullanabilir.

Program Kodlar: Bu kısımda prosedürün yapması gereken işi belirleyen kodlar bulunur.

Normalde bir prosedür bloğun sonuna kadar çalışır. Ancak belirli şartlar gerçekleştiğinde tamamı çalışmadan çıkmak için **Exit** komutu kullanılır. Örneğin koordinatı, genişliği ve yüksekliği verilen evi çizecek bir prosedür yazalım.

Örnek : Ev çizdirme prosedürü

Procedure EvCiz(x,y,h,w:Integer);

Var

ortax: Integer;

Const

CatiYuk = 3; {Duvarın 1/3 ü }

Kapigen = 5; {Tabanın 1/5 i }

KapiYuk = 2; {Duvarın yarısı}

Begin

Ortax:= x + w div 2;

Form1.Canvas.Rectangle(x, y, x + w, y-h); {Evin dışı çiziliyor}

{Çatıyı çiz}

Form1.Canvas.MoveTo(x, y-h);

Form1.Canvas.LineTo(ortax, y-h- h div catiyuk);

Form1.Canvas.LineTo(x + w, y-h);

{kapıyı çiz}

Form1.Canvas.Rectangle(ortax-w div Kapigen, y, ortax + (w div Kapigen), y-h div kapiyuk);

Form1.Canvas.MoveTo(ortax, y);

Form1.Canvas.LineTo(ortax, y-h div kapiyuk);

End;

Prosedürümüzün **Const** kısmında tanımladığımız **CatiYuk**, **Kapigen**, **KapiYuk** sabitleri kod bloğunda değiştirilemezken, **Var** kısmında tanımladığımız **OrtaX** ise bir değişkendir ve prosedürde değiştirilebilir.

Prosedürü çağırma işlemi de prosedür ismi ile parametreleri verilerek yapılır.

evciz(100, 100, 60, 80);

Bu kodu bir komut düğmesinin **Click** olay alt programına yazarsak düğme tıklandığında ev çizilecektir.

Procedure TForm1.Button1Click(Sender: TObject);

Begin

evciz(100, 100, 60, 80);

End;

Formumuza dört tane metin kutusu ekleyerek koordinat belirleme işleminde kullanalım. Aynı zamanda rasgele veya belirlediğimiz koordinatlarda çizim yaptırmak için iki tane komut düğmesini (button) formumuza ekleyelim. Bu komut düğmelerine aşağıdaki gibi **Click** alt programı yazarak yukarıdaki **evciz** prosedürüne ekleyelim.

Örnek: Eklenen komut düğmelerine ait prosedürler.

Procedure TForm1.Button1Click(Sender: TObject);

Var

x,y,w,h: Integer;

Begin

x:= StrToIntDef(Edit1.text,0);

y:= StrToIntDef(Edit2.text,0);

w:= StrToIntDef(Edit3.text,0);

h:= StrToIntDef(Edit4.text,0);

evciz(x, y, w, h); // evciz prosedürü çağırılıyor

end;

Procedure TForm1.Button2Click(Sender: TObject);

Var

x,y,w,h: Integer;

Begin

Randomize;

x:= Random(width);

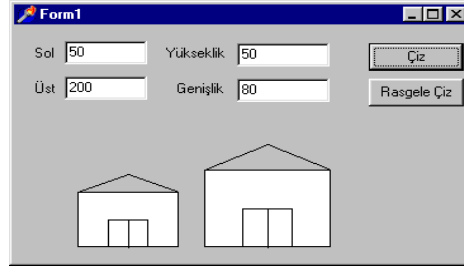
y:= random(Height);

w:= random(width) div 2;

h:= random(height) div 2;

evciz(x, y, w, h); // evciz prosedürü çağırılıyor

end;



Şekil 6.1. Ev çizme programının çıktısı

Şekil 6.1’de ev çizme programının ekran çıktısı verilmiştir. Ortak işler yapan kodlarımızı bir prosedür halinde yazarsak kodlamamız daha anlaşılır olur ve kod yazma işlemi kolaylaşır.

6.2. Fonksiyon

Bir Fonksiyon ise şu şekilde tanımlanır.

Function FonksiyonAdı (GirişParametreleri:Tipi):FonkTipi;

Sabit,değişken, Tip tanımı

Begin

Program Kodları;

[Exit;]

FonksiyonAdı:=Deger;

End;

Prosedür tanımından farklı olarak fonksiyon geriye bir değer göndereceği için bu değer tip fonksiyon tanımından sonra **FonkTipi** parametresi ile belirlenir. Ayrıca geri dönecek değer fonksiyon adına yada **Result** ifadesine atanan değer ile yapılır.

Function FonksiyonAdı (GirişPazametreleri:Tipi): FonkTipi;

Sabit,değişken, Tip tanımı

Begin

Program Kodları;

[Exit;]

Result:= Deger;

End;

Örneğin fonksiyona giren sayının faktöriyelini alacak bir fonksiyon yazalım.

```

Function Fak(x: Integer):LongInt;
Var
    Sonuc: LongInt;
    i: Integer;
Begin
    If x<0 Then
        Begin
            Fak:= -1;
            Exit;
        End;
    Sonuc:= 1;
    If x= 0 Then
        sonuc:= 1;
    Else
        For i:= 1 to x do
            sonuc:= sonuc * i;
        Fak:= sonuc; {sonuc gönderiliyor}
    End;

```

Fonksiyonu çağırma işlemi bir prosedür çağırma işlemi gibidir ancak fonksiyondan geriye değer döneceği için sonucun bir değişkene aktarılması gerekir.

```
Edit1.Text:= Fak(10);
```

Bir komut düğmesiyle sonucu Edit1 kutusuna yazmak için:

```

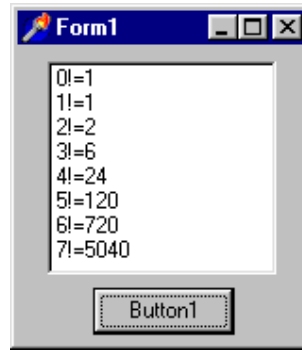
Procedure TForm1.Button1Click(Sender: TObject);
Begin
    Edit1.Text:= IntToStr(Fak(10));
End;

```

Şeklinde bir prosedür hazırlayabiliriz.

Örnek olarak kullanıcının gireceği bir sayıya kadar olan sayıların faktöriyelini bu fonksiyonu kullanarak bulduralım.

Örneğimiz için formumuz üzerine bir liste kutusu birde komut düğmesi yerleştirelim. Şekil 6.2.b'deki komut kümesini yazarak programımızı çalıştırsak Şekil 6.2.a'daki gibi bir sonuç elde ederiz.



Şekil 6.2a

```
Unit1.pas
Unit1

Function Fak(x: Integer): LongInt;
Var
  Sonuc: LongInt;
  i: Integer;
Begin
  If x < 0 Then
  Begin
    Fak := -1; Exit;
  End;
  Sonuc := 1;
  If x = 0 Then
    sonuc := 1
  Else
    For i := 1 to x do
      sonuc := sonuc * i; Fak := sonuc; {sonuc gönderiliyor}
    End;
  End;
procedure TForm1.Button1Click(Sender: TObject);
var
  n, i: Integer;
begin
  n := StrToIntDef(InputBox('Sayı girişi', 'Bir sayı gir', ''), 0);
  ListBox1.Clear;
  for i := 0 to n do
    ListBox1.Items.Add(IntToStr(i) + '!=' + IntToStr(Fak(i)));
  end;
```

Şekil 6.2.b. Faktöriyel hesaplayan program

Örnek Fonksiyon 1: Girilen sayıları binlik basamaklara ayırmak için bir fonksiyon kullanalım. Fonksiyonu aşağıdaki gibi hazırlayabiliriz.

Function Binlik(s: Real): String;

Begin

Result:= FormatFloat('###.###.###', s);

End;

Procedure TForm1.Button1Click(Sender: TObject);

Begin

Edit2.Text:=Binlik(StrToFloat(Edit1.text));

End;

Örnek Fonksiyon 2: Yazılan bir ifadeyi ters çevirmek için aşağıdaki fonksiyonu yazalım. Bu fonksiyonu çalıştırdığımızda şekil 6.3'deki gibi bir sonuç elde ederiz.

Function TersCevir(s : string) : string;

Var

i: integer;

s2 : string;

Begin

s2:= ' ';

for i:= 1 to Length(s) do

s2:= s[i] + s2; // her harfi s2'nin başına ekle

Result:= s2;

End;

Procedure TForm1.Button1Click(Sender: TObject);

Begin

Caption:= TersCevir(Edit1.Text);

End;

Geriye birden fazla değer dönmesi

Fonksiyondan geriye tek bir değer dönmektedir. Peki bir, iki veya daha fazla değer dönmeli gerekiyorsa ne olacak. Bu durumda, iki yöntem var. Birincisi geriye dönecek değerlerin bir **Record** tipinde olması ve dolayısıyla bu tipteki değişkenin alt değişkenleri ile çok sayıda değer geri dönebilmesi. İkinci yöntem ise bu tip tanımına

gerek kalmadan giren parametrelerle çıkışta yapılabilmesi. Giren bir parametre ile çıkışta yapılabilmesi için o parametrenin önünde **Var** deyimi kullanılır. Tabi ki bu durumda bir fonksiyon kullanmak ta mecburi değildir bir prosedürün giriş parametreleri ile de değer çıkışı yapılabilir.

Örnek Fonksiyon: Bir reel bir sayının tam ve ondalık kısmını ayıracak bir prosedürü aşağıdaki gibi yazalım.

Procedure cevir (s:Real; var t:LongInt; var v:Real);

Begin

t:=Trunc(s);

v:=Frac(s);

End;

Prosedüre giren s sayısının tam kısmını, t değişkeninde ve virgülden sonraki ondalık kısmını da v değişkeninde saklıyoruz. Bu değişkenler prosedür başlığında **var** deyimi ile tanımlandıkları için de prosedürün çağrıldığı yere bu değişkenlerin değeri dönecektir.

Prosedürümüzü bir düğmenin **Click** olayından çağıralım ve sayının tam kısmını **Edit1** ve virgülden sonraki kısmını da **Edit2** de gösterelim. Program satırları aşağıdaki gibi olacaktır.

Procedure TForm1.Button1Click(Sender: TObject);

Var

x: LongInt;

y: real;

Begin

cevir(1.9, x, y);

Edit1.text:= IntToStr(x);

Edit2.text:= FloatToStr(y);

End;

6.3. Prosedür ve Altprogram Bildirisi

Bir prosedürü veya fonksiyonu çağırmadan önce bazı durumlarda o prosedürün varlığını derleyiciye bildirmek gerekir. Bir prosedür veya fonksiyon tanımlandığı yerden daha önce çağrılıyorsa derleyici **Undeclared Identifier** hatasını verecektir.

Örneğin x ve y isimli iki prosedürümüz olsun ve bunlar sırasıyla şöyle yazılmış olsunlar.

Procedure x;

Begin

.....

End;

Procedure y;

Begin

X; // y prosedüründen x prosedürü çağrılıyor

.....

End ;

Bu durumda y prosedüründen x çağrılabilir.

Ancak x prosedüründen y çağrılamaz. Çünkü x prosedürü y'den önce tanımlanmıştır ve varlığının derleyiciye bildirilmesi gerekmektedir.

Procedure x;

Begin

y; // x prosedüründen y prosedürü çağrılmaya çalışılıyor

.....

End;

Procedure y;

Begin

.....

End;

Bu durumda önce y prosedürünü tanımlamış olmanız gerekir. Ancak y prosedürü de x prosedürünü çağırıyorsa hangisini öne alırsanız alın sonuç fark etmez.

Procedure x;

Begin

y; // x prosedüründen y prosedürü çağrılmaya çalışılıyor

.....

```

End;
Procedure y;
Begin
    X; // y prosedüründen x prosedürü çağrılıyor
    .....
End;

```

Bu durumu aşmak için, x prosedüründe y prosedürü çağrılmadan önce y prosedürünün varlığını **forward** deyimi ile derleyiciye bildirmemiz gerekir.

```

Procedure y; forward; // y prosedürünün varlığı bildirildi
Procedure x;
Begin
    y;
    .....
End;
Procedure y;
Begin
    X;
    .....
End;

```

6.4. Olay Altprogramları

Visual dillerde diğerlerinden farklı olarak kullanıcının tanımlayacağı alt programlardan başka kontrollerin kendi olay alt programları vardır. Ve kontrolle ilgili bir işlemde bu olay alt programlarından biri veya birkaçı aktif hale gelir. Bir olay alt programı şu şekildedir:

```

Procedure FormAdi.KontrolAdiOlay(Sender: TObject [,GirisParametreleri: Tipi]);
Örneğin
Procedure TForm1.Edit1MouseMove (Sender: TObject; Shift:

```

TShiftState; X, Y: Integer);

Bu olay altprogramını Form1 üzerinde bulunan **Edit1** kontrolünün **MouseMove** olayını temsil eder. **Shift**, **X** ve **Y** parametreleri ise mouse'un basılan tuşları ve koordinatlarını verir.

Olay alt programları, diğer altprogramlar gibi programcı tarafından çağırılması gerekmez, alt programın ait olduğu kontrol tarafından çağırılır. Örneğin bir kontrolün KeyPress olayı o kontrol üzerinde iken bir tuşa basılması durumunda veya MouseMove olayı kontrolün üzerinden mouse geçerken Delphi tarafından (dolaylı olarak Windows tarafından) çağırılır. Dolayısıyla olay altprogramlarının hangi sırada çağrılacağı veya kaç defa çağrılacağı tamamen kullanıcının yaptığı hareketlere bağlıdır. Ayrıca Windows'un bir kaç programı aynı anda çalıştırdığını biliyorsunuz. Bu bir programın altprogramları için de geçerlidir. Aynı anda aynı programın bir kaç altprogramı da çalışabilir. Bu yapı ilk etapta biraz karışık gelebilir ancak edineceğiniz program tecrübesiyle yapının oldukça kullanışlı olduğunu fark edeceksiniz. Çünkü artık programınızdaki kontrollerin nasıl ve hangi sırada çalışacağını belirlemek zorunda kalmayacaksınız. Her kontrol kendisinin ne zaman çalışacağını bilir ve bunun için programcının kod yazmasına ihtiyaç duymaz.

Olay alt programları istenirse diğer alt programlar gibi de o olay gerçekleşmeden de çağrılabilir. Bu çağrı o olayı meydana getirmez, sadece o olay için yazılmış alt programın çalışmasına sebep olur.

Olay altprogramlarında bulunan **Sender:TObject** parametresi bu olay alt programının hangi kontrol tarafından çağrıldığını belirler. Şöyle bir şey düşünülebilir:

Procedure TForm1.Button1Click (Sender: TObject);

satırındaki **Button1Click** bu altprogramın **Button1** kontrolüne ait olduğunu gösterir. O halde **Sender** parametresiyle bunu bildirmenin ne faydası olabilir?

Delphi'de bir olay altprogramını birden fazla kontrol ortak olarak kullanabilir. Bu durumda hangi kontrolün bu olay altprogramını çağırdığı bu parametre ile öğrenilir. (Delphi kontrolleri dizi olarak tanımlayamadığı için kontrol dizileri de bu yöntemle oluşturulur.)

Örneğin iki komut düğmesinin aynı **Click** olayını kullanmasını isteyelim. Bu işi **Button1** ve **Button2**'nin **OnClick** olayının karşısına, **Object Inspector** penceresinde **Button1Click** yazalım. Bu işlem hangi düğme tıklanırsa tıklansın **Button1Click**

olayının çalışacağını gösterir. Hangi düğmenin bu olayı çağırdığını ise **Sender** parametresi ile öğreneceğiz.

```
Procedure Tmain1.Button1Click(Sender: TObject);
```

```
Begin
```

```
if Sender = Button1 then
```

```
ShowMessage('Birinci düğme tıklandı')
```

```
else
```

```
ShowMessage('İkinci düğme tıklandı')
```

```
End;
```

Bu yöntem kullanılarak ortak işler yapan kontrollere aynı olay altprogramını gösterip daha az kodla daha çok iş yapılması sağlanmış olur.

6.5. Program Bloklarının Yapısı

Bloklar Delphi'de önemli bir yer işgal etmektedirler. Program kodlarına. Bloklar sayesinde modülerlik sağlanır. Delphi'de bloklar iki parçadan meydana gelir. Bunlardan birincisi Değişken, sabit, tip vb. tanımlama bloğu, diğeri ise program komutlarının doğrudan tanımlandığı bölümdür.

Tanımlama blokları aşağıdaki tanımlama bloklarından meydana gelmektedir.

- Prosedür / Fonksiyon tanımlama bloğu
- Değişken (Variable) tanımlama bloğu
- Sabit (Constant) tanımlama bloğu
- Tip (Type) tanımlama bloğu
- Kod bloğu

Aşağıda verilen örnekte **Begin...End** satırları arasındaki bölüm bir blok oluşturur. Bu blokta tanımlama bloğu bulunmayıp, Procedure ve kod bloğu içermektedir.

```
Procedure TForm1.Button1Click(Sender: TObject);
```

```
Begin
```

```
Showmessage ('İyi Çalışmalar');
```

```
End;
```

Procedure, Sabit, Değişken ve Kod bloğu içeren kod ise şöyle olacaktır:

```

Procedure Tform1.Button1Click(Sender: TObject);
Const {Sabit Tanımlama bloğu}
    pi= 3;
Var {Değişken tanımlama bloğu}
    r: integer;
    alan: integer;
Begin {Kod bloğu başlangıcı}
    r:= 5;
    alan:= pi * r * r;
    Showmessage('Dairenin Alanı' + inttostr(alan));
End; {Kod bloğu sonu}

```

6.5.1. Bir UNIT İçindeki Blok Yapısı

Bir unit tanımlama ve deyim bölümlerinden meydana gelir.

Tipik bir **Unit** yapısı aşağıdaki gibidir.

```

unit Unit1;
interface
uses
SysUtils, Windows, Messages, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls;
type // tip tanımlama bölümü
    TForm1 = class(TForm)
        Button1: TButton;
        procedure Button1Click(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
    end;
var
    Form1: TForm1; {Global Değişken tanımlama bloğu}
implementation

```

```

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject); { Prosedür bloğu başlangıcı}

Const

    pi=3;

var

    i:integer;

    alan:integer;

begin

    i:=5;

    alan:=pi*i*i;

    Showmessage('Dairenin Alanı'+ inttostr(alan));

end; // Prosedür bloğu sonu

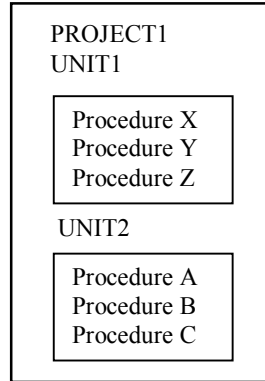
end. // Unit bloğunun sonu

```

Tanımlanan ifadelerin geçerlilik aralığı

Değişken, sabit, metod, tip vs. ifadelerinin aktif oldukları bölgeler, onların geçerlilik alanlarını tanımlar. Bu alanlar ilgili ifadeler için birer **local** bölgedir. Aşağıda verilen şekilde (Şekil 6.4) procedurlerin geçerli oldukları alanlar gösterilmiştir.

Şekil 6.4’de görüldüğü gibi her bir dikdörtgen bir bloğu temsil etmektedir. Eğer bir değişkeni **Procedure Y** içinde tanımlarsanız bu değişkene sadece **Procedure Y** ulaşabilir. Diğerleri ulaşamaz. Çünkü değişken bulunduğu bölge itibariyle **Local** bir değişkendir.



Şekil 6.4.

Eğer bir değişkeni UNIT1 de tanımlarsanız Procedure X, Y, Z için bu değişken global olduğundan bunların tümü o değişkeni kullanabilecektir. Fakat Procedure A, B, C bundan yararlanamayacaktır. Bunlar için değişken hala local dir.

Bir aralığın dışındaki bir değişkene ulaşmak

Her bir UNIT bir değişken aralığı teşkil ettiğinden dolayı bir unitten diğerinin adını kullanmak gerekir. Örneğin Unit1 de iken Unit2’de tanımlı olan Hesapla(sayi1, sayi2: integer); prosedürünü çağıralım:

Unit2. Hesapla(sayi1, sayi2: integer);

Tabi ki Unit1’in **Uses** deyiminden sonra Unit2 yazılmalıdır.

Şimdi local ve global değişkenlere bir örnek verelim. Örnek için Şekil 6.5’deki formu üzerindeki kontrollerle birlikte oluşturalım.

Şekil 6.5. Oluşturulacak form

Yukarda görüldüğü gibi form üzerinde Label1, Label2, Label3, SpinEdit1, SpinEdit2, Edit1 ve başlığı Çarp olan Button1 bulunmaktadır.

Procedure TForm1.Button1Click(Sender: TObject);

Var

Sayi1, Sayi2: integer;

Begin

Sayi1:= Strtoint(Spinedit1.text);

sayi2:= Strtoint(Spinedit2.text);

*Edit1.Text:= IntToStr(sayi1 * Sayi2);*

End;

Yukarıdaki program kodundan anlaşılabacağı gibi SpinEdit kontrolleriyle yada direkt spineditlere girilmek şartıyla belirlenen Sayi1 ve Sayi2 Button1Click prosedürü aracılığıyla çarpılarak sonuç Edit1’de gösterilmektedir.

Şimdi Form dizaynını aşağıdaki gibi değiştirelim:

Şekil 6.6.

Forma **Button2** eklenerek başlığı Topla yapıldı. Topla butonuna karşılık aşağıdaki kod girilir.

Procedure TForm1.Button2Click(Sender: TObject);

Begin

sayi1:= Strtoint(Spinedit1.text);

sayi2:= Strtoint(Spinedit2.text);

Edit1.Text:= IntToStr(sayi1 + sayi2);

End;

F9 tuşuna basılıp program çalıştırıldığında **Unknown Identifier** hata mesajıyla karşılaşılır. Çünkü Sayi1 ve Sayi2 değişkenleri sadece Button1click prosedüründe tanımlı olduğundan dolayı Button2click prosedürü bunları kullanamaz.

Şimdi bu problemi çözmek için iki metod gözüküyor. Bunlardan birincisi sayi1 ve sayi2 değişkenlerini Button2click prosedüründe tanımlamak. Bu durumda Button2click prosedürü için yazılan program kodumuz şu şekilde olacaktır:

Procedure TForm1.Button2Click(Sender: TObject);

Var

sayi1, sayi2: integer;

Begin

sayi1:= Strtoint(Spinedit1.text);

sayi2:= Strtoint(Spinedit2.text);

Edit1.Text:= IntToStr(sayi1 + sayi2);

End;

İkinci metod ise Sayi1 ve Sayi2 değişkenlerini Button1click ve Button2click prosedürlerinin dışında tanımlamak. Buda şu şekilde mümkün olmaktadır:

Implementation

*{ \$R *.DFM }*

Var

sayi1, sayi2: integer;

Procedure TForm1.Button1Click(Sender: TObject);

Begin

sayi1:= Strtoint(Spinedit1.text);

sayi2:= Strtoint(Spinedit2.text);

*Edit1.Text:= IntToStr(sayi1 * sayi2);*

End;

Procedure Tform1.Button2Click(Sender: TObject);

Begin

sayi1:=Strtoint(Spinedit1.text);

sayi2:= Strtoint(Spinedit2.text);

Edit1.Text:= IntToStr(sayi1 + Sayi2);

End;

İkinci metod seçildiğinde sayi1 ve sayi2 değişkenlerine Button1click ve Button2click prosedürlerinin dışındaki prosedürler de ulaşacaklardır. Çünkü sayi1 ve sayi2 değişkenleri local değil artık **global**'dır.

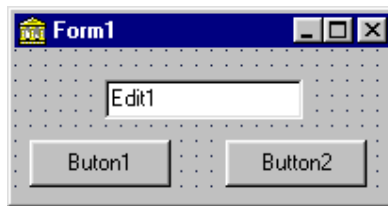
6.5.2. Bir Prosedür veya Fonksiyonun Program Kodu İçindeki Yeri

Prosedürler ve Fonksiyonlar bir Unitin **Implementation** bölümünde yer alırlar. Sadece içinde bulunulan UNIT tarafından kullanılacak tüm tanımlı ifadeler **Implementation** bölümünden deklare edilirler.

Başka Unit'ler tarafından kullanılacak Prosedür ve Fonksiyonlar tanımlamak için şu yollar takip edilmelidir:

- Fonksiyon veya Prosedür başlığı Unit'in **Interface** bölümüne yerleştirilir.
- Fonksiyon veya Prosedüre ait olan tüm kod **Implementation** bölümünde yer almalıdır.
- Başka Unit'ler tarafından çağrılan Unit'in adı **Uses** deyiminden hemen sonra yer almalıdır.

Şimdi vereceğimiz bir örnekle birden fazla unit tarafından kullanılacak fonksiyon çağrısını açıklığa kavuşturalım:



Şekil 6.7.

Şekil 6.7'deki form dizaynı kullanılarak Button1'e basıldığı zaman Edit1 kutusunun içeriği fonksiyon aracılığıyla büyük harfe çevrilerek Form1'in başlığına

yansıtılıyor. Button2 aracılığıyla Fom2'ye geçiş yapılarak aynı fonksiyon unit2 tarafından kullanılıyor.

Yukarıdaki form dizaynına ait unitin içeriği şu şekilde olacaktır:

Unit Unit1;

Interface

Uses SysUtils, Windows, Messages, Classes, Graphics, Controls, Forms, Dialogs,
StdCtrls;

Type

Tform1 = class(TForm)

Button1: TButton;

Edit1: TEdit;

Button2: TButton;

Procedure Button1Click(Sender: TObject);

Procedure Button2Click(Sender: Tobject);

Private

{ Private declarations }

Public

{ Public declarations }

End;

Var

Form1: Tform1;

Function Buyukharf(a:string):string;

Implementation

*{ \$R *.DFM }*

Uses unit2;

Function Buyukharf(a:string):string;

Begin

Result:= uppercase(a);

End;

Procedure Tform1.Button1Click(Sender: TObject);

Begin

Form1.caption:=Buyukharf(Edit1.Text);

End;

Procedure TForm1.Button2Click(Sender: TObject);

Begin

form2.show;

End;

End.

Buna karşın Unit2'nin içeriği de aşağıdaki gibi olacaktır:

Unit Unit2;

Interface

Uses SysUtils, Windows, Messages, Classes, Graphics, Controls, Forms, Dialogs,
StdCtrls;

Type

TForm2 = class(TForm)

Button1: TButton;

Edit1: TEdit;

Procedure Button1Click(Sender: TObject);

Private

{ Private declarations }

Public

{ Public declarations }

End;

Var

Form2: TForm2;

Implementation

Uses Unit1;

*{ \$R *.DFM }*

Procedure TForm2.Button1Click(Sender: TObject);

Begin

Form2.caption:= Buyukharf(Edit1.Text); // Buyukharf fonksiyonu direk çağrılıyor.

End;

End.

6.5.3. Public ve Private Deklarasyonları

Yeni bir form oluşturulduğu zaman Delphi o forma ait unit içerisine boş olan **Public** ve **Private** alanları koyar. Aşağıdaki gibi gözüktür:

Type

TilkForm = class(TForm)

Button1: TButton;

Edit1: TEdit;

Procedure Button1Click(Sender: TObject);

Private

{ Private declarations }

Public

{ Public declarations }

End;

Public bölümü şu amaçlar için kullanılır:

1. Diğer Unitlerin nesnelerinin içindeki metodların ulaşabileceği veri alanları deklare edilir.
2. Diğer Unitlerin nesnelerinin ulaşabileceği metodlar deklare edilir.

Private bölümü Public bölümüne nazaran sınırlı bir alana hitap etmektedir. Veri alanları ve metodların bu bölümden tanımlanması halinde diğer unitlerin erişim şansını tamamen ortadan kaldırmaktadır.

Private bölümü şu amaçlar için kullanılır:

1. Sadece içinde bulunulan unitin metodlarının erişeceği veri alanları deklare edilir.
2. Sadece içinde bulunulan unitin nesnelerinin erişeceği metodlar deklare edilir.

BÖLÜM 7

DOSYA İŞLEMLERİ

Daha önceki bölümlerde Formlara nasıl Label ve metin kutusu nesnesi(Edit) eklendiğini anlatmış, ShowMessage deyimi ve InputBox() fonksiyonu hakkında bilgi vermiştik. Diğer taraftan Pascal programlama dili hakkında bilgi verilen bölümde Pascal programları dahilinde girişi yapılan bilgilerin disk dosyasına nasıl yazıldığı veya kayıt yapıldığı konusunda bilgi verilmişti. Şimdi aynı işlemleri Delphi ile yapacağız. Verilecek örneğin kolay izlenebilmesini ve anlaşılmasını sağlamak için tek proje yerine, her işlem için ayrı projeler hazırlayacağım. Kayıt Girişi işlemleri ayrı, Kayıt düzeltme, Kayıt Silme ve kayıt arama işlemleri için önce ayrı projeler hazırlayacağım. Daha sonra bu ayrı olarak hazırlanan projeleri bir tek projede birleştireceğim. Bu bölümde en basit şekliyle anlatılan işlemler iyi anlaşılırsa, kitabın geri kalan kısımları çok daha kolay anlaşılır.

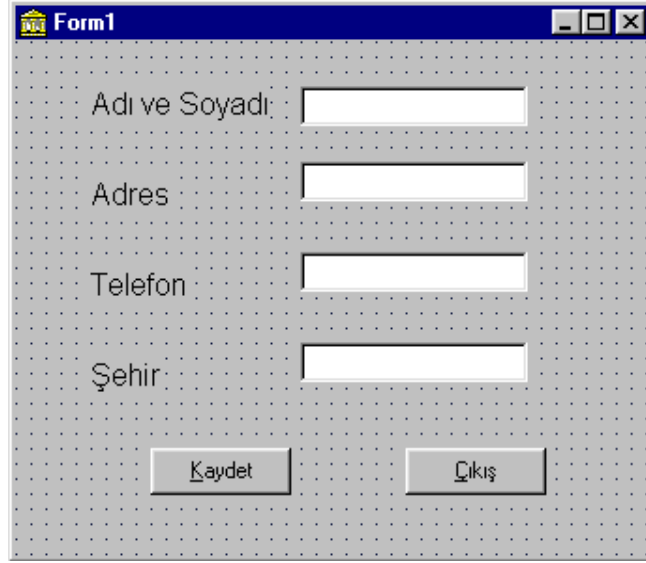
Diğer taraftan Delphi dahilinde dBASE ve Paradox formatında veri tabanı dosyası hazırlayıp erişim sağlanabildiği için bu bölümde an~atıldığı şekilde dosyalar hazırlayıp kayıt girişi yapmanın pek fazla pratik değeri yoktur. Bu bölüm özellikle daha önce Pascal ile çalışmış olanların Delphi'ye alışmaları amacıyla hazırlandı. Ancak basit ve az sayıda kayıt içeren uygulamalar için bu bölümde anlatılan veri tabanı dosyası hazırlama tekniğini kullanabilirsiniz.

7.1 Dosyaya Kayıt Girişi

Bütün programlama dillerinde programcılar öncelikle hard diske yani veri tabanı dosyalarına yazılacak bilgileri belirlerler. Veri tabanı dosyalarına bir kişi, bir kuruluş veya ticari işlemle ilgili olan bilgilerin hepsine birden 'kayıt' adı verilmektedir. Bu nedenle öncelikle dosyaya kayıt olarak hangi bilgilerin yazılacağı belirlenmelidir. Bizim örneğimizde veri tabanı dosyasına yazılacak bilgiler şunlar olacak:

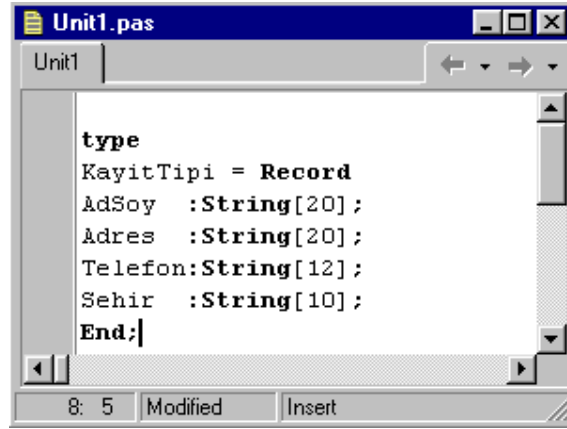
- Ad ve Soyad
- Adres
- Telefon
- Şehir

Kayıt bilgilerinin her birine alan adı verilmektedir. Konuyu basitleştirebilmek için veri tabanı dosyasına yazılacak bilgileri veya kaydın içeriğini sınırlı tuttum. Kayıta bulunan bütün bilgiler karakterse olacak. Bu amaçla hazırlanacak Giriş adlı projenin formuna kayıta bulunacak alanlarla aynı ada sahip metin kutuları ve bu metin kutuları için gerekli olan Label nesnelerini hazırladım. Kayıt girişi amacıyla kullanılacak Formun hazırlanmış halini aşağıda verdim.

The image shows a screenshot of a Windows application window titled "Form1". The window has a blue title bar with standard Windows window controls (minimize, maximize, close). The main area of the window has a light gray background with a fine grid pattern. There are four text input fields arranged vertically. Each field is preceded by a label: "Adı ve Soyadı:", "Adres:", "Telefon:", and "Şehir:". Below these fields, at the bottom of the form, are two buttons. The left button is labeled "Kaydet" and the right button is labeled "Çıkış".

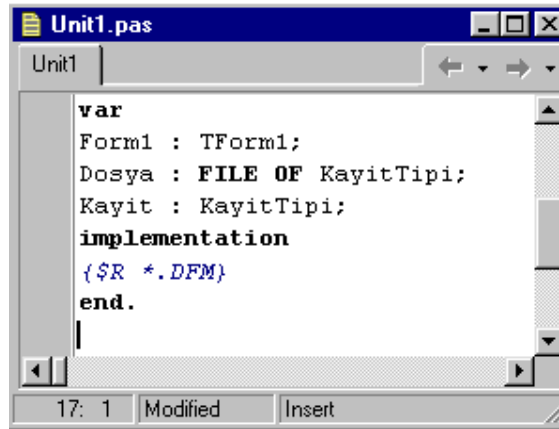
Şekil 7.1

Formun başlığını 'Kayıt Girişi' ve Proje içindeki projenin adını 'KayGiris' olarak seçtim. Ardından gireceğimiz kayıtları yazacağımız dosyanın kayıt yapısını belirlemek üzere projedeki tek Unit'te önce KayıtTipi adında bir Record tipi tanımladım.



Şekil 7.2

Ardından tanımladığım bu Record tipinden yararlanarak Unit'in değişken tanımlama işlemi yapılan kısmında FILE ve Record tipinde iki değişken tanımladım. Tanımladığım FILE tipindeki değişkene göre, kayıt girişi yapacağım bilgi dosyasının bu proje dahilindeki adı **Dosya** olacak.



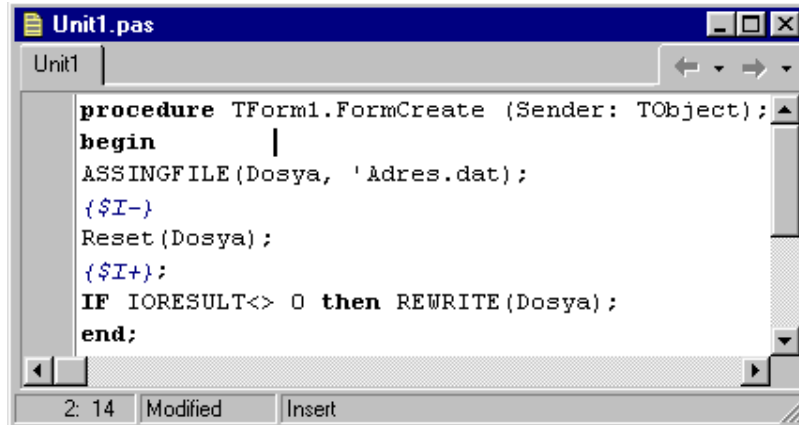
Şekil 7.3

Form üzerinde hazırlanan metin kutularına girişi yapılan bilgileri her seferinde kayıt olarak diske yazmak için önceden bazı hazırlıklar yapılmalıdır. Girişi yapılan bilgileri diske bir kayıt olarak yazabilmek için önceden ilgili dosyanın açılması veya dosya diskte yoksa oluşturulması gerekir.

Daha önce belirtildiği gibi en başta bilgi kayıt edilecek dosyanın açılması gerekir. Ancak dosyayı açmadan önce, açılacak veya bilgi girişi yapılacak dosyanın adının daha

önce FILE OF bildiri deyimi ile tanımlanan değişkene aktarılması gerekir. Bu amaçla Delphi dahilinde ASSIGNFILE deyimi kullanılmaktadır. Pascal'da aynı işlem ASSIGN deyimi ile yapılyordu. AssignFile deyimi ile üzerinde işlem yapılacak dosyanın adı FILE tipindeki değişkene aktarıldıktan sonra dosya RESET() deyimi ile açılabilir.

Henüz diskte hazırlanmamış bir dosyanın RESET() ile açılmak istenmesi halinde hata meydana geldiğini daha önce Pascal hakkında bilgi verilen bölümden biliyorsunuz. Bu nedenle RESET() deyimi ile dosya açılmadan önce programa {\$I-} satırı eklenerek Delphi'nin dosya açma işlemi sırasında meydana gelme olasılığı olan hatalara aldırması sağlanır. Eğer dosya açma işlemi sırasında açılmak istenen dosya diskte bulunamazsa, meydana gelen hatanın kodu otomatik olarak IORESULT adlı sistem değişkenine aktarılır. Bu değişkenin başlangıç değeri 0'dır. Eger RESET() deyimi ile yapılan dosya açma işlemi sırasında herhangi bir hata meydana gelmişse bu değişkenin içeriği 0'dan farklı olur ve söz konusu dosya REWRITE() deyimi ile diskte oluşturulur. Bu işlemleri yapacak program satırlarını proje başlatılır başlatılmaz işletilen FormCreate yordamına dahil ettim.

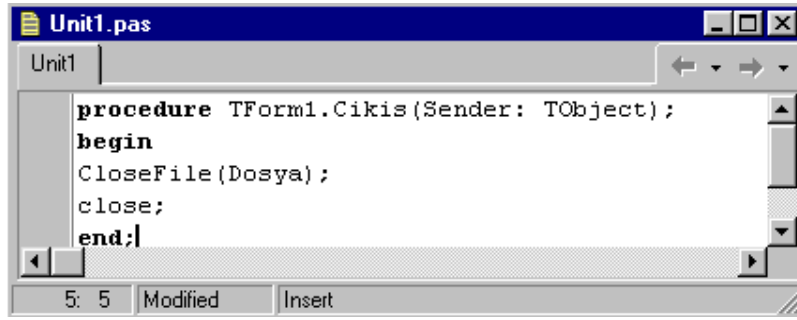


Şekil 7.4

ASSIGNFILE() deyimi ile açılacak dosyanın adı 'Adres.Dat'. Bu dosyanın diskteki yeri belirtilmediği için dosyanın aktif sürücünün aktif dizininde olduğu varsayılır. Açılmak istenen dosya aktif dizinde bulunamazsa, oluşturulur. Programın daha sonraki çalıştırılmalarında dosya aktif dizinde olacağı için, dosya açılmakla yetinilir. Dosyaların diskteki adlarından başka bir de proje içinde geçerli olan adları var.

Daha sonra dosyaya Write deyimi ile kayıt işlemi yapılırken, kaydın yazılacağı dosyayı belirlemek için dosyanın diskteki adı yerine, proje dahilindeki adı kullanılır.

Ekran görüntüsü daha önce verilen KAYGIRIS adıyla kayıt edilen ve Kayıt Girişi başlıklı form incelenirse, Label ve Edit nesnelerinden başka iki adet düğme var. Bu düğmelerden Çıkış başlığına sahip olanı programın çalışmasını sona erdirmek için kullanılacaktır. Ancak programın çalışması sona erdirilmeden daha önce RESET deyimi ile açılan dosyanın kapatılması gerekir. Dosya kapatmak amacıyla kullanılan deyim ise CCloseFile(). CCloseEile() deyimine parametre olarak kapatılmak istenen dosyanın proje içindeki adı verilmelidir.

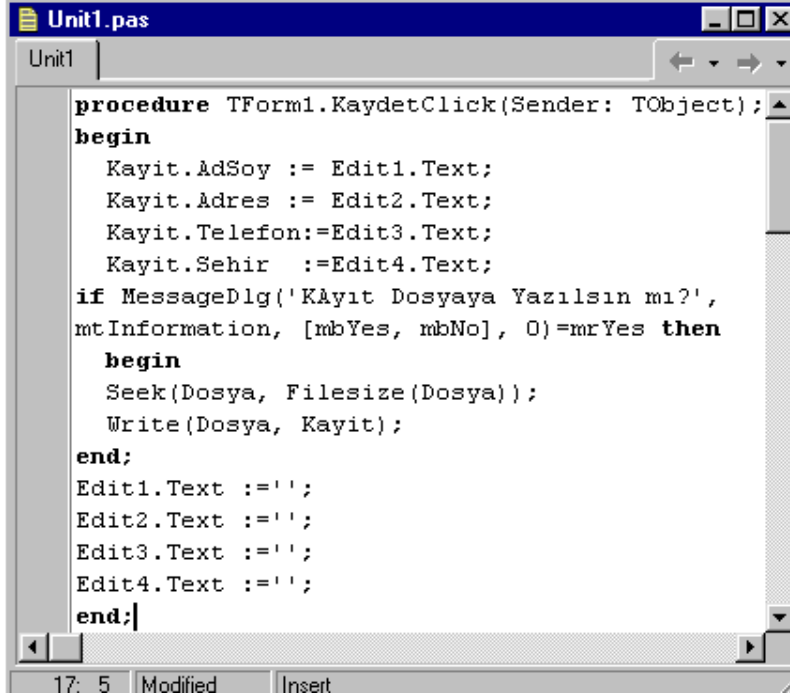


Şekil 7.5

Bu projede CCloseFile() deyimini kullanmasaydık her hangi bir sorunla karşılaşılmazdı. Çünkü Proje sona erdirildiği zaman, daha önce açılan bütün dosyalar otomatik olarak kapatılmaktadır. Proje dahilinde çok sayıda dosya açılıp kullanılıyor ve o an için artık kullanılmayacak dosyalar varsa, bu dosyaların CCloseFile() deyimi ile kapatılması önerilir. CClose methodu tek başına kullanıldığında, çalışma anında üzerinde çalışılan projenin formu kapatılır. Üzerinde çalıştığımız proje yalnızca bir forma sahip olduğu için, projenin çalışması sona erer. Projede birden fazla form olsaydı; CClose metodundan önce, kapatılmak istenen formun belirtilmesi gerekirdi.

Proje başlatılır başlatılmaz FormCreate yordamının içine yazılan program satırları işletilerek kayıtların yazılacağı Adres.Dat dosyası açılır. Bu işlemten sonra imleç formdaki ilk metin kutusunun içinde hazır olarak bekler. Kullanıcı kayda ait bilgileri Form'daki diğer metin kutuların girdikten sonra, girişi yapılan kayıt bilgilerini dosyaya

yazmak için Kaydet düğmesinde tıklama yapar. Kaydet düğmesinde tıklama yapıldığı zaman, girişi yapılan bilgilerin dosyaya yazılabilmesi için Kaydet düğmesi için Write deyimini de içeren program kodunun hazırlanması gerekir. Kaydet düğmesine ait program kodunu aşağıda ekran görüntüsü verilen kod penceresinde görebilirsiniz.



Şekil 7.6

Şimdi size ekran görüntüsü verilen bu program kodu penceresindeki program satırları hakkında bilgi vereceğim. Öncelikle ‘Adres.dat’ adındaki dosyaya kayıt girişi yapmak amacıyla hazırlanan bu Delphi projesinde kullanılan tek Forma ait UNIT'in diske KAYGIRIS.PAS adıyla kayıt edildiğini ve projedeki tek forma Form1 adının verildiğini anımsatmak istiyorum. Diğer taraftan, dosyaya yazılacak bilgileri temsil etmek üzere hazırlanan kayıt(Record) tipli değişkenin adı Kayıt olarak seçilmişti. Kaydet düğmesi için verilen program kodu, Click olayına bağlandığı için Kaydet düğmesinde tıklama yapıldığı zaman dosyaya kaydetme işlemi yapılır.

Formdaki metin kutularına girişi yapılan bilgileri dosyaya kaydedebilmek için, öncelikle metin kutularına(Edit nesneleri) girilen bilgilerin tek tek Kayıt adlı Record tipindeki değişkenin elemanlarına aktarılması gerekir. Çünkü Write deyimine 2. parametre olarak verilen ‘Kayıt’, daha önce tanımladığımız Record tipli bir değişkendir.

Metin kutularına girişı yapılan bilgiler aşıađıda verilen program satırları ile Record tipindeki deęiřkenin elemanlarına aktarılmaktadır.

```
Kayit.AdSoy := Edit1.Text;
```

```
Kayit.Adres := Edit2.Text;
```

```
Kayit.Telefon := Edit3.Text;
```

```
Kayit.Sehir := Edit4.Text;
```

Bu iřlemten sonra, kaydın ierięini Write deyimi ile dosyaya yazmadan nce MessageDlg() fonksiyonu ile kullanıcıdan onay alınmaktadır. Burada kullanılan MessageDlg() fonksiyonuna dıřarıdan 3 parametre verilmektedir. Birincisi kayıt ncesi onay almak iin diyalog kutusuna yazılan mesajdır. İkinci parametrede ise MessageDlg() fonksiyonu ile ekrana getirilen diyalog kutusuna yazılacak mesajdan nce, kullanıcıya uyarıcı bilgi vermek zere hangi resmin bulunacaęını belirleyen sabit bilgi var. 3. parametre ile diyalog kutusunda yer alacak dęmeler belirlenmektedir.

```
if MessageDlg('Kayıt Dosyaya Yazılsın mı?',  
mtlInformation, [mbYes, mbNo], 0)=mrYes then
```

Form zerinde bulunan metin kutularına istedięiniz bilgilerin giriřini yapıp Kaydet dęmesinde tıklama yaparsanız, programın iřletimi iinde MessageDlg() fonksiyonunun bulunduęu satıra gelir ve MessageDlg() fonksiyonu diyalog kutusunu ekrana getirir.

MessageDlg() fonksiyonu tarafından ekrana getirilen diyalog kutusunda Yes dęmesinde tıklama yapar veya Evet dęmesi seili durumda iken Enter tuřuna basarsanız, MessageDlg() onksiyonu geriye mantıksal True deęerini, No dęmesinde tıklama yaparsanız mantıksal False deęerini geriye dndrr. MessageDlgQ fonksiyonu ile ekrana getirilen diyalog kutusunda Yes dęmesinde tıklama yaparsanız, blok iine alınan Seek ve Write deyimlerinin kullanıldıęı program satırları iřletilir.

```
Seek(Dosya, Filesize(Dosyay));
```

```
Write(Dosya, Kayit);
```

Seek() deyimi ile kaydı dosyaya yazmadan nce,kayıt yazma kafası dosyanın istenen yerine konumlandırılır. Seek() deyimi dıřarıdan iki parametre almaktadır. İlk parametrede sz konusu dosyanın program iindeki adı belirtilmektedir. İkinci parametrede ise dosyanın zerine gidilmek istenen kaydın numarası belirtilir. Yeni

kaydı dosyanın sonuna yazmak istediğimiz için önce FileSize() fonksiyonu ile dosyadaki kayıt sayısını bulup Seek() deyimine parametre olarak verdim. Ardından Write() deyimi ile Record tipli değişkenin içeriği dosyanın sonuna yazılır. Record tipli Kayıt değişkeninin içeriği Write() deyimi ile dosyanın sonuna yazıldıktan sonra formdaki metin kutularının içeriği silinmektedir. Bu şekilde dosyaya arka arkaya istediğiniz sayıda kaydın girişini yapabilirsiniz.

7.2 Dosyadan Kayıt Okumak

Bir önceki konuda hazırlanan proje örneği. ile diskteki adı Adres.Dat olan dosyaya nasıl kayıt yapılacağı konusunda bilgi verildi. Şimdi aynı dosyadan kayıtların nasıl okunup ekrana yazıldığı konusunda örnek bir proje üzerinde bilgi verilecektir. Bu amaçla yeni bir Delphi projesi hazırladım. Yukarıdaki sayfalarda hazırlayıp kayıt girişi yaptığımız dosyadan kayıt okumak amacıyla hazırladığım bu projeye ARAMA.DPR adını verdim. Bu proje ile daha önce diske kaydettiğimiz bilgileri okuyup ekrana getireceğimiz için ilk olarak projenin tek UNIT'inde, AdSoy, Adres, Telefon ve Şehir adında String tipindeki 4 değişkeni kendi içinde barındıran bir değişken tipini tanımladım.

```
Type;  
KayıtTipi = Record  
AdSoy : String[20];  
Telefon : String[12];  
Sehir : String[10];  
End;
```

Ardından KayıtTipi adını verdiğim bu Record tipi ile 'Kayıt' adında bir değişken tanımladım, Record tipine ve bu tip içinde yer alan değişkenlere ad verirken daha önceki projedeki değişken adlarına bağlı kaldım. Değişkenlerin özellikleri aynı olmak şartıyla değişkenlere istediğiniz adı verebilirsiniz.

```
Var
```

Form1 :Tform1;

Dosya :FILE OF KayıtTipi;

Proje çalıştırılır çalıştırılmaz otomatik olarak işletilen FormCreate yordamına yazılan program kodları ile, daha önce hazırladığımız Proje ile kayıt yaptığımız Adres.dat dosyası açılır. Dosyadan bilgi okumak amacıyla hazırladığım bu projenin FormCreate yardamı ile kayıt girişi amacıyla hazırladığımız projenin FormCreate yordamı birbirinin aynısıdır.

Projenin çalıştırılıp FormCreate yordamı aracılığı ile bilgi okunmak istenen dosyanın açılması işlemi tamamlanınca, ekleme noktası projenin formundaki ilk metin kutusunda hazır olarak bekler. Bu sırada dosyada aranıp bulunmasını istediğiniz kişinin(kaydın) adını ve soyadını girmeniz gerekir. Okunup ekrana getirilecek kaydın Ad ve Soyad bilgileri girilip Formdaki Kayıt Ara başlıklı düğmede tıklama yapılınc, kayıt dosyadan okunacak ve kaydın detayları Formda bulunan diğer metin kutularına yazılacak. Proje dahilinde kullanılacak form, bir önceki konuda kayıt yazmak amacıyla kullanılan forma oldukça benzemektedir. Aşağıda verilen ekran görüntüsünü kayıt okumak amacıyla hazırlanan projeyi çalıştırıp aranacak kaydın Ad ve Soyad bilgilerini formdaki ilk metin kutusuna girdikten hemen sonra aldım.



Şekil 7.7

```

Procedure TForm1.KayitAraClick(Sender :TObject);
Var
I: Integer;
For I := 0 To FileSize (Dosya) – 1 Do
Begin
Seek (Dosya,I);
Read(Dosya,Kayit);
If Kayit.AdSoy = Edit1.Text Then
Begin
Edit2.Text := Kayit.Adres
Edit3.Text := Kayit.Telefon;
Edit4.Text := Kayit.Sehir;
End;

```

Bu yordamda ,dosyadaki kayıttan başlamak üzere , sıra ile bütün içerikleri Read ile deyimi okuyup, daha önce tanımlanan Record tipindeki değişkene aktarılmaktadır. Ardından, okunan kaydın Ad-Soyad bilgisi formdaki ilk metin kutusunun(Edit1) içeriği ile karşılaştırılmaktadır. Formdaki ilk metin kutusunun içeriği, diskten okunup Record tipindeki kaydın ad-soyad bilgisi ile aynı ise, kaydın içeriği formdaki diğer metin kutuları aracılığı ile görüntülenir.

7.3 Kayıt Düzeltmek

Kayıt okumak amacıyla bir önceki konuda verilen Arama adlı projenin formundan farklı olarak Kayıt Düzeltme başlıklı bu formda fazladan ‘Değiştir’ başlığına sahip olan bir düğme var. Değiştir başlıklı düğme ile değişikliğe ugratılan kaydın tekrar dosyaya yazılması sağlanacaktır. Program çalıştırılıp Form yüklenildikten sonra kullanıcıdan ilk olarak düzeltilecek kaydın Ad ve Soyad bilgisi istenir.

Düzeltilecek kaydın Ad ve Soyad bilgisi, formdaki ilk metin kutusuna yazılıp Kayıt Ara düğmesinde tıklama yapılırsa, söz konusu kayıt dosyada aranır. Kayıt dosyada bulunursa içeriği ekrana getirilir.



Şekil 7.8

Okunup içeriği ekrana getirilen kayıta istediğiniz değişikliği yapabilirsiniz. Yapılan değişikliklerden sonra kaydın yeni şeklini diske tekrar yazmak için projenin formuna ‘Değiştir’ başlıklı bir düğme eklendi. Bu düğmede tıklama yapılanca, bu düğmeye ait Click yordamı aktive olacağından, kaydın yeni şeklini diske yazmada kullanılacak Write() deyimini bu yordama dahil ettim.

```
Procedure TForm1.ButtonClick(Sender: TObject);  
Begin  
  Kayit.AdSoy := Edit1.Text ;  
  Kayit.Adres := Edit2.Text;  
  Kayit.Telefon :=Edit3.Text;  
  Kayit.Sehir := Edit4.Text;  
  Write(Dosya, Kayit);  
  Edit1.Text := ‘’;  
  Edit2.Text := ‘’;  
  Edit3.Text := ‘’;  
  Edit4.Text := ‘’;  
  Edit1.SetFocus;  
End;
```

7.4 Dosyadan Kayıt Silmek



Şekil 7.9

Diğer konularda olduğu gibi Form yüklenince, başka bir deyişle proje çalıştırılınca, silinecek kaydı içeren dosya açılır ve ekleme noktası formdaki ilk metin kutusunda silinecek kaydın Ad ve Soyad bilgisini girmek üzere bekler. Silmek istediğiniz kaydın ad ve soyad bilgisini girip Kayıt Ara düğmesinde tıklama yapmanız halinde, söz konusu kayıt dosyadan okunup ekrana getirilecek.



Şekil 7.10

Dosyadan okunup ekrana getirilen kaydı dosyadan silmede kullanılacak yordamı ‘Kayıt Sil’ başlığına ve ‘Sil’ adına sahip düğmede yapılacak tıklama(Click) olayına bağladım. Sil düğmesinde tıklama yapılırca, Delphi, proje dahilinde SilClick adında bir yordam arar. Bu nedenle dosyadan kayıt silmek için gerekli program kodları SilClick adlı yordamın içine yazılmalıdır.

```
Procedure TForm1.SilClick (Sender: TObject);  
var  
I Integer;  
Begin  
ASSINGFILE (Dosya1, 'C:\Temp.Dat');  
REWRITE(Dosya) – 1 Do  
Begin  
Seek(Dosya,I);  
Read(Dosya,Kayit);  
If Kayit.AdSoy <> Edit1.Text then  
Write(Dosya1, Kayit)  
End;  
CloseFile(Dosya);  
CloseFile(Dosya1);  
DeleteFile('c:\adres.dat');  
RenameFile('c:\Temp.Dat', 'c:\adres.dat');  
Edit1.Text := '';  
Edit2.Text := '';  
Edit3.Text := '';  
Edit4.Text := '';  
End;
```

BÖLÜM 8

FORMLARA UYGULANAN METODLAR

Delphi ile geliştirilen uygulamaların en temel bileşenleri Formlar ve formlara dahil edilen nesnelerdir. Uygulamalara dahil edilen her nesnenin varsayım olarak bazı özellikleri bulunmaktadır. Söz konusu nesnenin varsayılan özellikleri tasarım anında Object Inspector penceresinde veya çalışma anında program kodu yazılarak değiştirilebiliyor. Nesnelerin özelliklerinden başka, nesnelerle ilgili olarak önceden tanımlı olan çok sayıda olay var. Önceden tanımlı olan olaylar nesneden nesneye değişiklik göstermektedir. Bir önceki bölümde Form nesnesine uygulanabilen olaylar hakkında bilgi verildi. Bir nesne için anlatılması gereken bir diğer önemli konu ise, nesneler üzerinde işlem yapmada kullanılan Metod'lardır. Kitabın bundan önceki kısımlarında Formların üzerine bilgi yazmak için her seferinde TextOut() methodundan yararlandık. Su bölümde Formlarla ilgili olarak en çok kullanılan Method'lar hakkında bilgi verilecektir.

8.1 Show Metodu – Formları Görüntülemek

Üzerinde çalışılan projede yalnızca bir form varsa, bu form proje çalıştırıldığında otomatik olarak ekrana getirilmektedir. Bu nedenle projedeki ilk formun yüklenmesi konusu ile ilgilenmemize gerek yoktur. Ancak üzerinde çalıştığınız projede birden fazla form varsa gerek duymanız halinde bu formları Show methodu ile ekrana getirmemiz gerekir.

```
Procedure TForm1.FormClick(Sender: TObject);  
Begin  
Form2 . Show;  
End;
```

Show methodu ile ekrana getirilmek istenen form önce ekrana getirilir, ardından aktif form yapılır. Daha önce ekranda görüntülenmesine rağmen pasif duruma getirilmiş olan formlar Show methodu ile aktif form durumuna getirilebilir. Bunun dışında daha

önce Hide methodu ile gizlenmiş olan formlar, Show methodu ile tekrar ekrana getirilebilirler.

8. 2 Close ve Hide Methodları - Formları Kapatmak ve Gizlemek

Daha önceki konularda belirtildiği gibi Close methodu ile daha önce ekrana getirilmiş olan Formlar istenildiği zaman kapatılabilmekte veya ekrandan kaldırılmaktadır. Üzerinde çalışılan projede bir tek form varsa veya o sırada belleğe yüklenmiş durumda olan bir tek form varsa, Close methodu tek başına kullanılabilir.

Ancak projede birden fazla form varsa, Close methodundan önce kapatılmak istenen formun adı belirtilmelidir. Diğer taraftan Close methodu ile kapatılan form, projenin başlangıç formu ise, form kapatma işlemi projenin çalışmasının sona ermesine neden olur.

```
Procedure TForm1.FormClick(Sender: TObject);  
begin  
    Focm2.Close;  
End;
```

Hide methodu, formları masaüstünden kaldırıp gizler. Eğer masaüstünde o sırada gerek duymadığınız formlar varsa, banları Hide methodu ile gizleyebilirsiniz. Gizlenen form, Show methodu ile tekrar ekrana getirilene değin orijinal görünümünü korur.

```
Procedure TForm2.FormClick(Sender: TObject) ;  
begin  
    Form1.Hide;  
end;
```

Hide methodu ile formu içerdiği bütün nesnelerle birlikte gizlemek yerine, forma daha önce eklemiş olduğunuz nesnelerden istediğinizi Hide methodu ile gizleyebilirsiniz. Bu işlemin nasıl yapıldığını göstermek için üzerinde çalıştığım projenin tek formuna bir Memo nesnesi ve iki düğme dahil ettim.



Şekil 8.1

Çalışma anında Gizle başlıklı düğmede tıklama yapıldığı zaman Memol adındaki Memo nesnesi gizlenecek ve Göster başlıklı düğmede tıklama yapıldığı zaman ise Memo nesnesi formun üzerinde tekrar görüntülenecek. Bu nedenle Gizle başlıklı düğmenin Click yordamında Hide methodunu kullandım.

```
Procedure TForm1.GizleClick(Sender: TObject);  
begin  
Memol.Hide;  
End;
```

Bu yordam ile çalışma anında gizlenen Memo nesnesi Göster başlıklı düğme için hazırlanan yordam ile tekrar görüntülenir.

```
Procedure TForm1.GösterClick(Sender: TObject);  
Begin  
Memol.Show;  
End;
```

8.3 SetFocus Methodu

Daha önceki konularda belirtildiği gibi formlara eklenen her nesneye bir sıra numarası verilmektedir. Nesnelerin sıra numaraları söz konusu nesnenin TabOrder değişkeninde saklanmaktadır. Forma eklenen ilk nesnenin TabOrder özelliğine 0, ikinci sırada, eklenen nesneye ait TabOrder özelliğine ise 1 değeri aktarılmaktadır. Ancak

Object Inspector penceresinden yararlanarak forma eklemiş olduğunuz nesnelerin TabOrder değişkenlerin içeriklerini değiştirebilirsiniz. Çalışma anında Tab tuşu ile Formdaki nesnelerin arasında dolaşırken, nesnelerin TabOrder değişkenlerin içeriklerine göre hareket edilir. Diğer taraf tan, SetFocus methodu ile daha önce ekrana getirilmiş olan formlardan herhangi birini, aktif form duruma getirebilirsiniz.

```
Procedure TFoxm2.FormClick(Sender: TObject);  
begin  
Form1.SetFocus;  
End;
```

Daha önceki konularda belirtildiği gibi Formun yüzeyine Canvas adı verilmekteydi. Canvas nesnesine TextOut() methodu ile bilgi yazma denemeleri yapmıştık. Şimdi Canvas nesnesi veya Formun yüzeyi üzerinde işlem yapılırken kullanılan diğer methodlar hakkında bilgi vereceğiz.

8. 4 Refresh Methodu

TextOut() methodu veya başka bir teknikle Formun üzerine yazılanları veya çizilenleri silmek için Refresh methodundan yararlanabilirsiniz. Refresh methodundan Formdaki diğer nesneler etkilenmez. Eğer üzerini temizlemek istediğiniz Form aktif değilse, Refresh methodu ile birlikte Formun adını belirtmeniz gerekir.

```
Procedure TForm1.Button1Click (Sender: TObject);  
begin  
Form1.Refresh;  
End;
```

8. 5 LineTo() ve MoveTo() Methodları

LineTo() methodu ile aktif veya belirtilen formun yüzeyine(Canvas) çizgi çizilir. Delphi'deki LineTo() methodu, Pascal'daki Line komutundan farklı değildir. LineTo()

metodu ile çizgi çizilmek istenen Form aktif durumda değilse, LineTo() methodundan önce üzerinde çizgi çizilecek Form veya nesne belirtilmelidir.

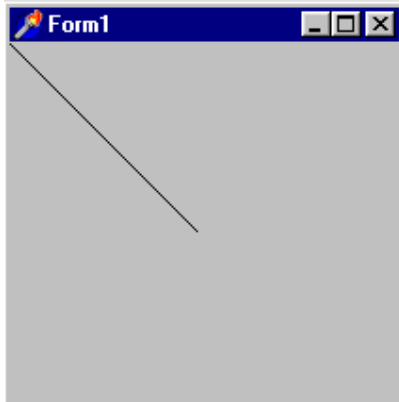
Genel Yazılışı

LineTo(X, Y)

Genel yazılıştan görüleceği gibi LineTo() methodu dışarıdan 2 parametre almaktadır. LineTo methodu ile bulunulan yerden, belirtilen yere kadar çizgi çizilir. Aşağıda verilen örnekte çalışma anında fare ile formun üzerinde tıklama yapılması halinde, Formun sol üst köşesinden itibaren düz bir çizgi çizilir.

```
Procedure TForm1.FormClick (Sender: TObject);  
begin  
Canvas.LineTo(100,100); end;
```

Bir Form aktive edildiği zaman LineTo() methodu ile çizgi çizme işlemi formun sol üst köşesinden itibaren yapılır. Çalışma anında formun üzerinde tıklama yapıp FormClick yordamının işletilmesi halinde, formun sol üst köşesinden yani (0,0) koordinatından (100, 100) koordinatına kadar düz çizgi çizilir.



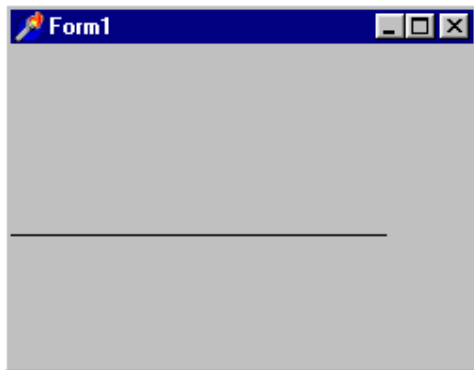
LineTo() methoduna çizgi çizme işleminin sona erdirileceği koordinat belirtilirken, bitiş yerinin önce sütun değeri ardından satır değeri Piksel birimi ile yazılır. Eğer çizgi çizme işlemine Formun sol üst köşesinden itibaren başlatmak istemiyorsanız, MoveTo methodundan yararlanmanız gerekir. Aşağıda verilen iki satırlık program kodu ile formun (0,100) koordinatından (200,100) koordinatına kadar düz çizgi çizilir.

```

    proceduce TFotml.FormClick(Sender: TObject);
    Begin
Canvas.MoveTo(0,100);
    Canvas.LineTo(200,100);
    End;

```

Çalışma anında formun üzerinde tıklama yapıp bu yordamı işletecek olursanız projenin formuna aşağıdakine benzer düz bir çizgi çizilir.



Eğer formun üzerine farklı bir kalınlıkta çizgi çizmek istiyorsanız Pen.Width komutunu kullanabilirsiniz.

```

Pccedure TForm1.FormClick(Sender: TObject);
    beqin
Canvas.Pen.Width := 3;
    Canvas.MoveTo(0,100);
    Canvas.LineTo(200,100);
    End;

```

Bu değişiklikten sonra proje çalıştırılıp formun üzerinde tıklama yapılp Form1Click yordamı işle tilirse, formun üzerine 3 piksel kalınlığında çizgi çizilir.

LineTo() methodu ile formun üzerine yani Canvas'a çizgi çizerken farklı bir çizgi rengi kullanmak istiyorsanız Pen nesnesine ait Color özelliğinden yararlanabilirsiniz. Çizgi rengini değiştirmek üzere Pen nesnesine ait Color özelliği için renk seçimi

yapılırken, renk adlarının önüne 'cl' harfleri eklenmektedir. Bu kitap dahilinde farklı renkleri kullanmadığım için sarı rengi seçtim. Ayrıca, forma sarı renkte çizgi çizmeden önce formun Ctl3D özelliğini False yaptım.

```
procedure TForm1.FormClick(Sender: TObject);
begin
Focml.Ctl3D:=False;
Canvas.Pen.Width:=4;
Canvas.MoveTo(50,50);
Canves.Pen.Color := clYellow;
Canvas.LineTo(200,100);
end;
```

Çalışma anında formun üzerinde tıklama yapıldığı zaman MouseDown ve formun üzerinde tıklama yapıp farenin tuşu tam serbest bırakılırken MouseUp olayı meydana gelmektedir. Bu iki olaydan yararlanarak çalışma anında PaintBrush gibi boyama programlarında olduğu gibi, formun istediğiniz yerine istediğiniz uzunlukta düz çizgi çizebilirsiniz. Bu nedenle ilk olarak üzerinde çalıştığımız projenin Unitine FormMouseDown() yordamının kalıbını hazırlayıp dahil etmemiz gerekir. Bunun için Object Inspector penceresinde form için önceden tanımlı olan olaylar listelenirken fare ile OnMouseDown olayının üzerinde çift tıklama yapılır.

```
Procedure TForm1.FormMouseDown(Sender: TObject);
Button: TmouseButton;
Shift: TshiftState;
X, Y: Integer;
begin
Carivas.MoveTo(X, Y);
End;
```

Çizilecek çiiğinin başlayacağı yeri bu şekilde belirledikten sonra, çizginin biteceği yeri belirtmek gerekir. Çizginin biteceği yeti belirhnek için MouseUp olayından yararlandım.

```
Procedure TForm1.FormMouseUp(Sender: TObject);
```



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

```
begin
```

```
Canvas.Pen.Width := 4;
```

```
Canvas.LineTo(X,Y);
```

Hazırladığım program kodlarına göre çalışma anında çizginin başlayacağı yerde farenin sol tuşuna basacaksınız. Elinizi tuşun üzerinden kaldırmadan fare işaretini çizginin biteceği yere kadar götürüp bırakacaksınız.

Şimdi PaintBrush gibi boyama programlarındaki gibi nasıl serbest çizim yapabileceğimizi anlatacağım. Serbest çizim için MouseDown ve MouseMove olaylarından yararlanacağım. MouseMove olayı fare işareti formun üzerinde hareket ettirildikçe meydana gelmektedir. Çalışma anında fare sürükleme pozisyonunda hareket ettirildikçe üzerinde geçilen yerlere çizgi çizilecektir. Fare işareti normal pozisyonda formun üzerinde hareket ettirilirken, çizim işlemi yapılmayacak. Bunu sağlamak için en başta 'Ciz' adında Boolean tipinde bir değişken tanımladım.

Çizim işlemi farenin sol tuşu üzerinde yapılacak tıklama işlemi ile başlayacağı için önce üzerinde çalıştığım projenin tek Unitine MouseDown olayı için FormMouseDown yordamını hazırladım. Çalışma anında MouseDown olayı meydana gelince çizim işlemine başlanılacağı için 'Ciz' değişkenine True değerini aktardım. Ayrıca MoveTo methodu ile çizimin başlayacağı yeri belirledim.

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

```
Button: TMouseButton;
```

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

```
Begin
```

```
Ciz := True;
```

```
Canvas.MoveTo(X, Y);
```

```
end;
```

Çalışma anında formun üzerinde fare işaretinin yeri 1 piksel kadar değiştirilirse, bir kez MouseMove olayı meydana gelir. 2 piksel kadar hareket ettirilirse, MouseMove olayı iki kez meydana gelir. Bu özellikten yararlanmak için FormMouseMove()

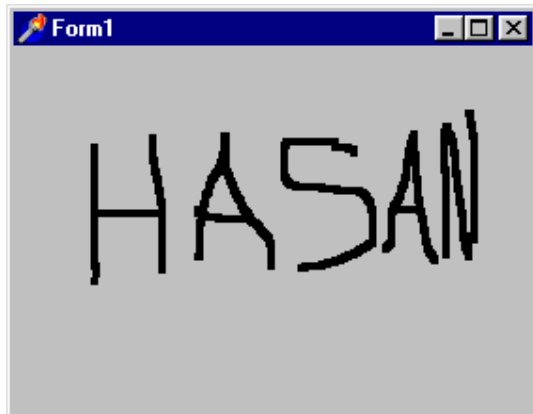
yordamında LineTo() methoduna yer verdim. MouseMove olayının meydana geldiği sırada 'Ciz' değişkeni True değerini içeriyorsa 1 piksel boyundaki çizgi çizilir.

```
procedure TForm1.FornMouseMove(Sender: TObject);  
Shift: TShiftState; X, Y: Integer);  
begin  
if Ciz then  
Canvas.LineTo (X, Y);
```

Çalışma anında farenin sol tuşu serbest bırakıldığında veya MoveUp olayı meydana geldiğinde çizim işleminin sona erdirilmesi için FormMoveUp yordamında 'Ciz' değişkeninin içeriğini False yaptım.

```
Button: TMouseButton;  
Shift: TShiftState; X, Y: Integer);  
begin  
Ciz := False;
```

Bu işlemlerden sonra çalışma anında sürükleme pozisyonunda formun üzerinde farenin gezdirildiği yerlerde serbest çizgi çizilmiş olunur.



8. 6 Elipsle Methodu

Bu method ile istenen Forma istenen çap, renk ve kalınlıkta çember ve elipsler çizilir.

Genel Yazılışı

Ellipse (Başlama X, Başlama Y, Yükseklik, Genişlik)

8. 7 Rectangle Metodu – Dikdörtgen Çizmek

Genel Yazılışı:

Rectangle(Başlama X, Başlama Y, Genişlik, Yükseklik)

8. 8 TextWidth ve TextHeight Metodları

TextWidth metodu ile sabit bir bilginin geçerli olan Font ve Puntuya değerine bağlı olarak yazma işlemi sırasında Form üzerinde kaplayacağı genişliği piksel cinsinden bulmaya yararmaktadır. TextHeight ise metnin yüksekliğini bulmada kullanılmaktadır.

Örnek:

Genislik :=Canvas.TextWidth('Borland Delphi');

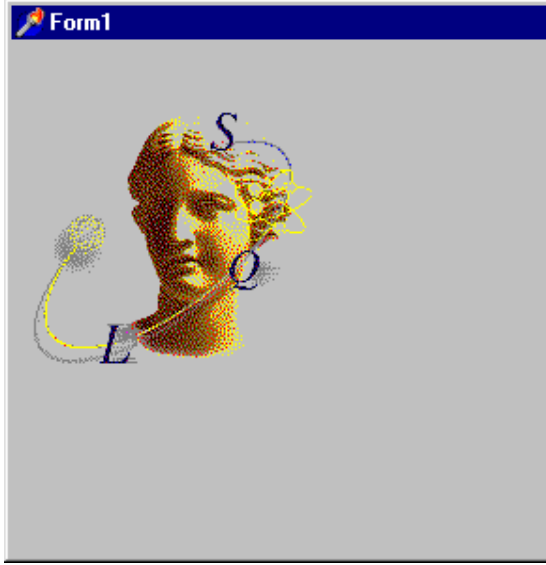
Yukseklik := Canvas.TextHeight('Borland Delphi');

8. 9 Draw ve Create Metodları

Delphi'de Tbitmap , Ticon, TmetaFile, Tpicture adında başka hazır nesnelerde bulunmaktadır. Create metodu ile bu tip nesneler çalışma anında program kodu yazılarak hazırlanan Tbitmap ve TmetaFile gibi nesnelerin içerikleri forma veya Image nesnesi içinde görüntülenmektedir.

```
Procedure TForm1.FormClick(Sender : TObject);
Var
Bitmap1 :TBitmap;
begin
Bitmap1 := Tbitmap.Create ;
Bitmap1.LoadFromFile('c:\athena.bmp');
Form1.Canvas.Draw(0, 0,Bitmap1;
End;
```

Create metodu ile Tbitmap tipinde nesne hazırlandıktan sonra LoadFromFile metodu ile diskteki BMP uzantılı bir resim dosyası okunup, hazırlanan Bitmap nesnesine aktarılmaktadır. LoadFromFile metodu ile diskten Bitmap nesnesini temsil eden değişkene aktarılan resmi projenin formu üzerinde görüntülemek için Draw metodu kullanılmaktadır. Draw metodu dışarıdan 3 parametre almaktadır. İlk iki parametre ile Bitmap nesnesinin içeriğinin form üzerinde görüntülenecek yeri belirtilmektedir. 3. Parametre ile içeriği görüntülenecek nesne belirtilmektedir.

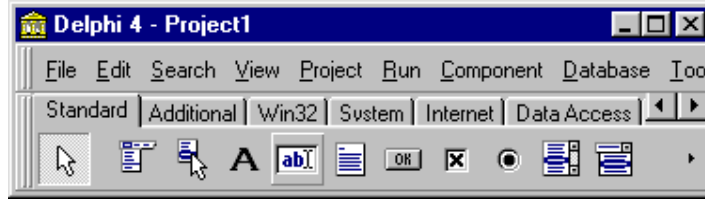


BÖLÜM 9

EDİT KONTROLÜ

9.1 Metin Kutuları

Formlara eklenebilen bir diğer nesne veya kontrol metin kutularıdır. Tasarım anında aktif durumdaki Forma bir metin kutusu eklemek için önce Toolbars araç çubuğunda Edit(metin kutusu) kontrolünü temsil eden düğme seçili duruma getirilir. Toolbars Palette araç çubuğunda Edit nesnesini temsil eden düğme seçili durumda iken formun üzerinde tıklama yapılarak forma Edit nesnesi eklenir. Diğer nesnelerde olduğu gibi fare ile Edit nesnesinin form üzerindeki yerini ve boyutlarını istediğiniz gibi belirleyebilirsiniz.

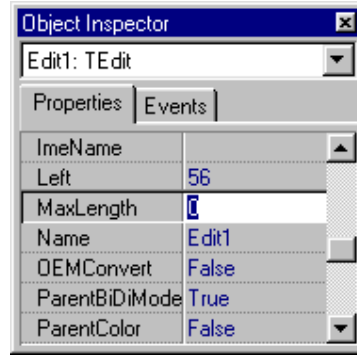


Şekil 9.1

Edit nesnesini temsil eden düğme seçili durumda iken formun üzerinde tıklama yapılarak forma ait edit nesnesi eklenir. Forma eklenen ilk metin kutusuna Edit1 adı verilmektedir. Metin kutularının içeriği çalışma anında kullanıcı tarafından belirlendiği için genel alışkanlıkla tasarım anında metin kutularının içerikleri boşaltılır.

9.2 Metin Kutusuna Girilecek Bilgi Miktarı

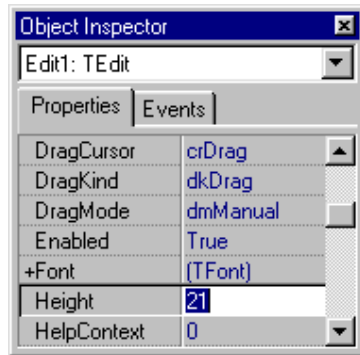
Bir metin kutusuna en fazla 32KB kadar metin girişi yapılabilir. Metin kutusuna girilebilecek karakter sayısı **MaxLength** değişkeni aracılığıyla belirlenir.



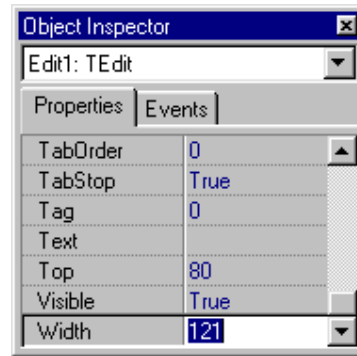
Şekil 9.2

9. 3 Metin Kutularının Genişlik ve Yükseklikleri

Forma yerleştirilmiş olan bir metin kutusunun güncel genişliği metin kutusuna ait Width Değişkeninde, metin kutusunun yüksekliği ise Height adındaki değişkenle saklanmaktadır.



Şekil 9.3a

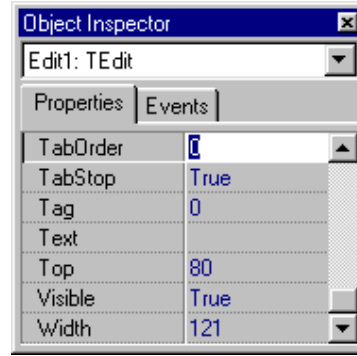


Şekil 9.3b

Bunun dışında, metin kutularının form üzerindeki konumu, metin kutusu nesnesine ait **Top** ve **Left** değişkenleri ile izlenmektedir.

9. 4 TabOrder Özelliği

Çalışma anında Forma dahil edilmiş olan bir sonraki nesnenin üzerine gitmek için Tab tuşuna, sıradaki bir önceki nesnenin üzerine gitmek içinde Shift + Tab tuşlarına basılır. Nesnelerin sıra numarası nesneleri TabOrder değişkeni aracılığı ile izlenir.

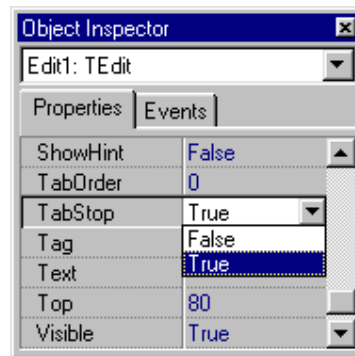


Şekil 9.4

Bu genellemeye Label nesneleri dahil değildir.

9.5 TabStop Özelliği

Bazı durumlarda Formlara eklenen metin kutularına kullanıcının müdahale etmesi istenmez. Örneğin kayıt düzeltmek amacıyla hazırlanan bir Formdaki Kayıt Numarası yazılı olan metin kutusunun içeriğinin kayıt okuma işleminden sonra değiştirilmemesi gerekir. Bu ve benzeri bir nedenle bir Edit nesnesinin atlanması isteniyorsa Edit nesnesine ait TabStop özelliğinden yararlanılmalıdır. Forma eklenen Edit nesnelerinin TabStop değişkeni varsayım olarak True değerini sahiptir.

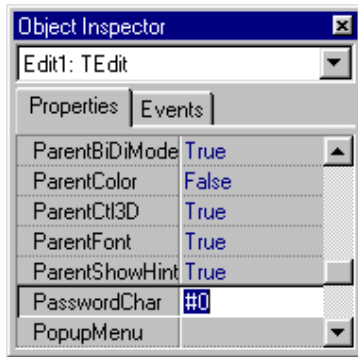


Şekil 9.5

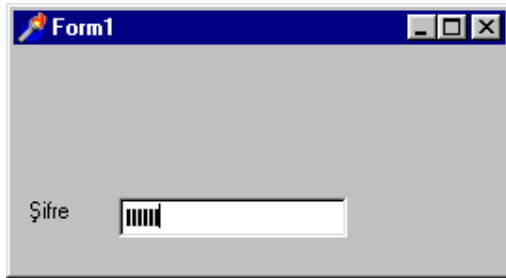
Edit nesnelerinin TabStop değişkeni tasarım anında Object Inspector penceresinde veya çalışma anında değiştirilebilir. Sırada TabStop değişkeni False alan bir Edit nesnesi olduğu zaman, Tab tuşuna basıldığında söz konusu Edit nesnesi atlanır.

9. 6 Şifre Karakteri

Özellikle ticari amaçlı programlarda programa her girişte şifre istenir. Delphi programcıları ek bir çaba harcamadan şifre girişinde metin kutularından yararlanabilirler. Şifre olarak metin kutusuna girilen karakterler, giriş işlemi sırasında ekranda görünmemelidir. Girilen her karakterin yerine '*' gibi bir şifre karakterinin ekranda ilgili yere yazılması gerekir. Forma eklediğiniz metin kutusunu şifre girişinde kullanmak istiyorsanız Edit nesnesine ait PasswordChar özelliğinden yararlanmanız gerekir.

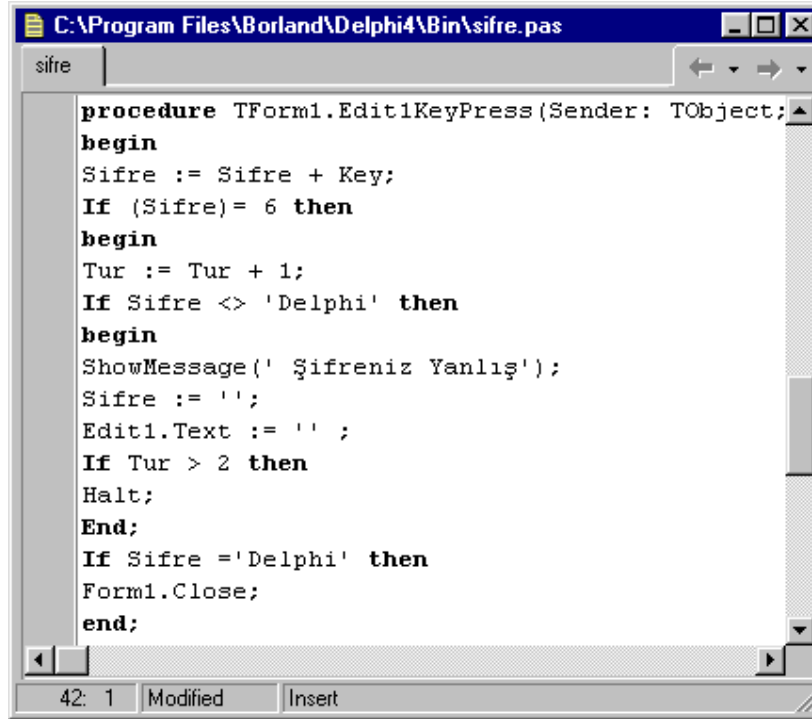


Varsayım olarak başlangıçta PasswordChar değişkeni '#0' bilgisini içermektedir. Bu değişkene tasarım veya çalışma anında istenen karakter atanabilir. Eğer forma eklemiş olduğunuz metin kutusunu 5 karakterlik şifre girişinde kullanmak istiyorsanız Object Inspector penceresinde PasswordChar değişkenine "#5" değerini aktarmanız gerekir. Aşağıda verilen ekran görüntüsünü formdaki tek metin kutusu nesnesinin PasswordChar özelliğine '#5' değerini aktarıp projeyi çalıştırdıktan sonra aldım.



Ayrıca şifre olarak girilecek bilgileri saklamak için Unite ait değişken tanımlama bloğunda(Var) iki değişken tanımladım. Kullanıcı şifre girmek üzere herhangi bir tuşa bastığında, Edit nesnesine ait olan EditKeyPress yordamı ile şifrenin karakter ve tekrar

sayısı hesaplanır. Girilen karakterler belirlenen sayıda ve belirlenen şifre ile aynı ise kullanıcı programa girebilecek. Değilse programın çalışması sona erdirilecek.



9. 7 Metin Kutusuna Girilen Bilgiyi Büyük veya Küçük Harfe Çevirmek

İstenilen karakterin büyük veya küçük harfe çevrilmesi için **CharCase** özelliğinden yararlanılır. Eğer metin kutusuna girilen bütün harflerin büyük harfe çevrilmesini istiyorsak CharCase özelliğine **ecUpperCase** değerini aktarmak gerekir. Eğer küçük harfe çevrilmesini istiyorsak bu kez **ecLowerCase** değerini aktarmamız gerekir.

9.8 ReadOnly Özelliği:

Eğer metin kutusu aracılığı ile ekrana getirdiğiniz değişiklik yapılmasını önlemek istiyorsanız metin kutusuna ait ReadOnly özelliğine True değerini aktarmanız gerekir.

9.9 Metin Kutularını Gizlemek – Visible Özelliği:

İstediğiniz metin kutusunun formun üzerinde görünmesini engellemek istiyorsanız, metin kutusuna ait **Visible** özelliğine False değerini aktarmanız gerekir.

9.10 Modified Özelliği:

Modified özelliğinden, metin kutusu nesnesinin içeriğinin değiştirilip değiştirilmediği öğrenilmek istendiği zaman yararlanılmaktadır.

BÖLÜM 10

DELPHİ KONTROLLERİ

Visual dillerin kullanıcıya sunduğu yeniliklerin başında şüphesiz ki kontrol elemanları (components) gelmektedir. Kullanıcı artık küçük bir kontrol için uzun uzadıya program kodu yazmak zorunda kalmayacak, kontrol elemanları vasıtasıyla program modülleri daha kısa ve daha anlaşılır bir şekilde kontrol edilebilecektir.

Her kontrol elemanı kendisine ait olan özellikleri (Properties) sayesinde kullanıcıya çok sayıda seçenek sunmaktadır. Bunlar kullanılarak programın gerek tasarımı, gerekse çalışma zamanı esnasında programa istenen şekil ve estetik verilebilmektedir. Object Inspector penceresinde bulunan özelliklerin bir kısmı hem tasarım hem de çalışma zamanında değiştirilebilir. Tasarım anında değiştirilemeyecek olan özellikler Object Inspector penceresinde yer almamaktadır. Bunlar program modüllerinde kod yazılarak değiştirilebilir. Bazı özellikler ise hem çalışma zamanı hem de tasarım zamanı değiştirilemez, yalnız okunabilirler.

Bir kontrolün propertiesini çalışma zamanı değiştirmek için o propertiese aşağıdaki gibi atama yapılır.

KontrolAdi.PropertiesAdi := Değer;

Örneğin:

Edit1.Text := “ELAZIĞ”

Buradaki KontrolAdi o kontrolün **Name** propertiesine verilen isimdir. Kullanıcı her bir kontrol elemanını seçmek suretiyle formun istenilen yerine ve istenilen boyutta farenin sol tuşunu basılı tutarak çizebilir. Form üzerine alınan bir kontrol elemanı kesme, kopyalama, yapıştırma, taşıma ve bir çoğu boyutlandırma işlemlerini destekler. Form üzerindeki kontrol elemanlarını mouse ile çerçeveleyerek bloklamak suretiyle yerleşim şekillerini bozmadan konumlarını, veya ortak olan özelliklerini değiştirmek mümkündür. Seçme işlemini **shift** tuşuna basılı tutup seçilmek istenen elemanlar tıklanarak da yapılabilir.

Eğer aynı kontrol elemanından birden fazla kullanılacaksa kopyalama metodu kullanılabilir. Bu yöntemle çoğaltılan elemanların olay alt programları da aynı olacaktır.

Image, Frame, Panel gibi kontrol elemanlarının içine birden fazla kontrol elemanı yerleştirmek mümkündür. Ana kontrol elemanı yer değiştirdiği zaman içindeki

elemanlar da onunla birlikte yer deřiřtirecektir. Bu tip kontroller diđer kontrolleri gruplamak için kullanılır ve birçok yararlı özellikleri vardır.

Bu bölümde sık kullanılan bazı kontrol elemanlarının açıklanması, özellikleri (Properties), olayları (Events) ve o nesneye uygulanabilecek yöntemler (Methods) örneklerle açıklanmıştır. Özelliklerin bir kısmı her kontrol elemanı için aynı anlam ifade ettiğinden dolayı sadece bir eleman için verilmektedir. Ortak olmayan özellikler ise her eleman için ayrı ayrı ele alınmaktadır.

10.1. Standart Kontrol Elemanları



Şekil 10.1. Standart kontrol elemanları paleti.

Bu palette program yaparken en çok kullanacağınız aşağıdaki kontroller bulunur. (Programlarınızda sık kullanacağınız kontrollerden bazıları da **Additional** ve **System** kısmındadır.)

Standart kontrol elemanlarının kullanım amaçlarını kısaca verelim:



TMainMenu : Menü çubuğu tasarlamada kullanılır.



TPopupMenu : Sağ fare tuşuyla çalışan menüler hazırlamada kullanılır.



TLabel (Etiket) : Kontrollere açıklama yazmada kullanılır.



TEdit (Metin Kutusu): Kullanıcının bilgi girebilmesi için kullanılır.



TMemo : Kullanıcının birden fazla satıra sahip bilgileri girmesi için kullanılır.



TButton (Komut Düğmesi): Kullanıcının bir işi yaptırabilmesi için kullanılır.



TCheckBox (İşaret Kutusu): Bir seçeneği aktif veya pasif yaptırmak için kullanılır.



TRadioButton (Seçenek Kutusu): birden fazla seçenektan birinin seçilmesi gereken durumlarda kullanılır.



TListBox (Liste Kutusu): Birden Fazla elemanı listelemek ve düzenlemek için kullanılır.



TComboBox (Aşağı Doğru Açılan Liste): Kullanıcının hazır değerlerden birini seçebilmesi için kullanılır.



TScrollBar (Kaydırma Çubuğu): Kaydırma işlemlerinde veya değer arttırıp azaltma işlemlerinde kullanılır.



TRadioGroup: Seçenek düğmelerini tasarım zamanı oluşturabilmek için kullanılır.



TPanel: Diğer kontrolleri gruplamak ve durum çubuğu oluşturmakta kullanılır.



TGroupBox (Gruplama Kutusu): Diğer kontrolleri (daha çok seçenek düğmelerini) gruplamakta kullanılır.

10.1.1. TEdit (Metin Kutusu)

Bu kontrol elemanı kullanıcının bilgi girişi yapmasına imkan veren ve programlarımızda en çok kullanacağımız kontrollerden biridir.

Properties (Özellikler)

Text :

Text özelliği Edit kontrolünün en önemli özelliğidir. Edit kutusu içerisinde yazılması istenen ifade ve kullanıcının kutuya girdiği ifade bu özellik vasıtası ile öğrenilir.

Edit1.Text:= 'İyi Çalışmalar...';

Örnek olarak kullanıcının **Edit** kutusuna gireceği para miktarının içindeki banknotların sayısını bulacak bir program yazalım. Programımız için forma büyükçe bir **Label** kontrolü, altına bir **Edit** kutusu ve bir komut düğmesi yerleştirerek aşağıdaki formu (Şekil 10.2.a) hazırlayalım.

a) Hazırlanan form**b) Programın ekran çıktısı**

Programdaki **Edit** kutusuna yazılan para miktarındaki banknotların sayısını yazacağımız programla **Label** kontrolü içinde göstereceğiz. Komut düğmesinin **Click** olayına aşağıdaki kodu yazalım.

// Örnek:Text

procedure TForm1.Button1Click(Sender: TObject);

var

i,j: Integer;

begin

i := StrToInt(Edit1.Text); // Edit kutusundakini sayıya çevir

j := i div 5000000; // kaç tane tam 5000000 var

Label1.Caption:= ' '; // Label1 içeriğini sil

Label1.Caption := Label1.Caption + IntToStr(j) + 'tane 5000000 TL' + #13#10;

i := i mod 5000000; // kalanı bul

j := i div 1000000; // kaç tane tam 1000000 var

Label1.Caption := Label1.Caption + IntToStr(j) + 'tane 1000000 TL' + #13#10;

i := i mod 1000000; // kalanı bul

j := i div 500000; // kaç tane tam 500000 var

Label1.Caption := Label1.Caption + IntToStr(j) + 'tane 500000 TL' + #13#10;

i := i mod 500000; // kalanı bul

j := i div 250000; // kaç tane tam 250000 var

Label1.Caption := Label1.Caption + IntToStr(j) + 'tane 250000 TL' + #13#10;

i := i mod 250000; // kalanı bul

j := i div 100000; // kaç tane tam 100000 var

Label1.Caption := Label1.Caption + IntToStr(j) + 'tane 100000 TL' + #13#10;

i := i mod 100000; // kalanı bul

j := i div 50000; // kaç tane tam 50000 var

Label1.Caption := Label1.Caption + IntToStr(j) + 'tane 50000 TL' + #13#10;

i := i mod 50000; // kalanı bul

j := i div 20000; // kaç tane tam 20000 var

```

Label1.Caption := Label1.Caption + IntToStr(j) + 'tane 20000 TL' + #13#10;
i := i mod 20000; // kalanı bul
j := i div 10000; // kaç tane tam 10000 var
Label1.Caption := Label1.Caption + IntToStr(j) + 'tane 10000 TL' + #13#10;
i := i mod 10000; // kalanı bul
Label1.Caption := Label1.Caption + 'Kalan' + IntToStr(i) + 'TL nin banknot
değeri yok';
end;

```

Yukarıdaki programda kullandığımız iki değişkenimiz var. **i** değişkeni para miktarını **j** değişkeni ise bir banknottan kaç tane olduğunu belirtiyor. Bir banknottan kaç tane olduğunu bulabilmek için mod para miktarının o banknota bölünmesi gerekir.

j:=i div 10000 işlemi bu işi yapmaktadır. Burada / yerine **div** kullanmamızın sebebi tam sayıları bölmek için / operatörünün kullanılamayışındandır. Delphi’de tam sayı bölme işlemlerinde **div** kullanılır.

Para miktarındaki bir banknottan kaç tane olduğunu bulduktan sonra kalan miktarı bulmak için de **mod** operatörü kullanılabilir. **i := i mod 20000** işlemi para miktarındaki 20000’lik banknotları çıktıktan sonra kalan para miktarını verir.

Örnekte kullanılan **#13#10** karakterleri ise **enter** tuşu işlevi görür. Yani satırların alt alta yazılmasını sağlar.

CharCase:

Bu özellik vasıtası ile isterseniz kullanıcının Edit kontrolüne girilecek harflerin tümünün büyük harfli, yada tümünün küçük harfli, yada normal harfli (hem büyük hem de küçük) olmasını sağlayabilirsiniz. Bu özellik hem çalışma hem de tasarım zamanında Object Inspector penceresinde değiştirilebilir:

ecUpperCase :Tümü büyük harfe çevrilecek.

ecNormal : Normal giriş.

ecLowerCase : Tümü küçük harfe çevrilecek.

```

procedure TForm1.Button1Click (Sender: TObject);

```

```

begin

```

```

    Edit1.CharCase:= ecLowerCase;

```

end;

Font :

Font özelliği bir kontrol elemanının font biçimini ve font büyüklüğünü belirler. Açılan font penceresiyle bütün özellikler görülerek değiştirilebileceği gibi aşağıdaki alt özelliklerle tek tek de yapılabilir.

Tek tek yapmak için **Font** özelliğinin aşağıdaki özelliklerini kullanmamız gerekir.

Font.Color : Yazı rengini belirler.

Edit1.Font.Color := clRed; // yazı rengi kırmızı

Font.Height : Fontun piksel olarak boyu.

Font.Size : Fontun points olarak boyu.

Font.Name : Fontun adı. Sistemde yüklü olan fontlardan herhangi birinin adı olması gerekir. Bu font bulunmazsa Windows varsayılan fontu kullanacaktır.

Edit1.Font.Name := 'Courier New';

Font.Style : Fontun **kalin**, **eğik**, **altıçizili**, **üstüçizili** özellikleri bununla belirlenir.

Şu değerler tek tek veya birlikte atanarak font özellikleri değiştirilebilir:

fsBold, fsItalic, fsUnderline, fsStrikeOut.

Örneğin Edit kutusunun fontunu kalın ve eğik yapmak için;

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

Diğer özelliklerini değiştirmeden kalın özelliğini de vermek için;

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

Sadece kalın özelliğini kaldırmak için;

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

Komutlarını yazmak gerekir.

Burada görüldüğü gibi diğer özellikleri değiştirmeden bir özelliği vermek istediğimizde +, o özelliği kaldırmak istediğimizde – operatöründen yararlanıyoruz.

Font.Pitch: Bu özellik fonttaki her bir karakterin eşit genişlikte olup olmayacağını belirler. Alabileceği değerler ve anlamları şöyledir:

fsFixed: Her harf aynı genişlikte.

fsVariable: Her harf kendi genişliğinde.

fpDefault: Fontun orijinal hali kullanılacak. Yani font Courier New tipi bir fontsa her harf aynı genişlikte, Times New Roman tipi bir fontsa her harf kendi genişliğinde.

PasswordChar:

PasswordChar özelliği **Edit** kutusu şifre girişi için düşünülüyorsa kullanılır. Girilen şifrenin etraftakiler tarafından görülmesi istenmez. Bu yüzden de bir çok programda girilen şifre çoğunlukla * işaretleriyle gösterilir. PasswordChar özelliğine bir karakter atarsanız kullanıcının bu kutuya girdiği karakterler yerine burada belirlediğiniz karakterler görülür.

Edit1.PasswordChar := '';*

PasswordChar özelliğini iptal etmek için de bu özelliğe girdiğiniz karakteri silin veya #0 vererek boş olduğunu belirtin. Bu işlemi program kodu ile yapmak için;

Edit1.PasswordChar := #0;

kodunu kullanabilirsiniz.

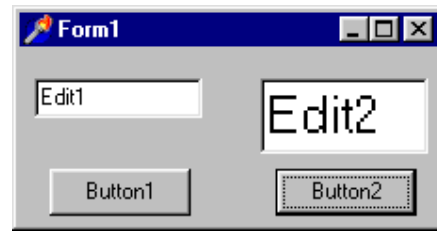
AutoSize:

AutoSize özelliğinin **True** olması halinde Edit1'in font büyüklüğü değiştiği zaman Edit1 de aynı oranda değişecektir. Aksi taktirde Edit1'in boyu sabit kalacak ve büyük fontların bir kısmı görülmeyecektir. Ancak alt alta bir kaç kontrolünüz varsa ve bunlardan birinin AutoSize özelliği **True** ise yazı boyu büyüdüğünde kontrolün yüksekliği de büyüyeceği için diğer kontrollerin üzerini kaplayacaktır.

AutoSize özelliğinin etkisini görmek için aşağıdaki programı hazırlayıp çalıştıralım. Ekran çıktısı Şekil 10.3'teki gibi olacaktır.



a) Buton 1'e basıldığında



b) Buton 2'ye basıldığında

Şekil 10.3. AutoSize özelliği

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    Edit1.AutoSize := false;
    Edit1.Font.Size := 20;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
    Edit2.AutoSize := true;
    Edit2.Font.Size := 20;
end;
```

Görüldüğü gibi Buton1'e basıldığında AutoSize özelliği **false** olduğu için birinci Edit kutusu aynı boyda kalmakta ve içeriği tam olarak görülememektedir. Buton2'ye basıldığında ise ikinci Edit kutusu otomatik olarak uygun boyuta getirileceğinden içeriği tam olarak görülebilecektir.

Eğer Edit kutusunun boyutunun her zaman sabit kalmasını istiyorsanız **AutoSize** özelliğini **false** yapın. Böylece Edit kutusunun form üzerindeki yerleşimi bozulmayarak diğer kontrollerin üzerine taşması engellenmiş olacaktır.

AutoSelect:

Bu özelliğinin **True** olması, **Tab** tuşu ile ulaşılması halinde içindeki bilginin seçilmesini sağlar. Bu kullanıcının sürekli bilgi girdiği durumlarda, örneğin personelle ilgili bilgilerin girildiği durumlarda kullanıcıya hız kazandırır. Kullanıcı isterse tek tuşla siler isterse hiçbir değişiklik yapmadan geçer.

MaxLength:

Kutuya girilebilecek karakter sınırını belirler. 0 verilirse bu sınır kaldırılır. Kullanıcının belirli sayıdan daha fazla karakter girmesini önlemek için bu özellik kullanılır. Örneğin kullanıcının 3 harften daha fazla bilgi girişi yapmasını önlemek için bu özelliğe 3 değerini verebilirsiniz.

ReadOnly: Bu özellik **true** ise kullanıcı Edit kutusunda bulunan bilgiye herhangi bir müdahalede bulunamaz. Yani ne içindeki bilgi değiştirilebilir ne de bilgi girişi gerçekleştirilebilir.

Modified:

True veya False değerlerinden birini alan bu özellik kontrol kutusunun Text özelliğinde herhangi bir değişme olduğu zaman True değerini alır. Örneğin programdan çıkarken “Değişiklikler kaydedilsin mi?” gibi bir soruyu ancak bir değişme olduğu zaman sormak gerekir. Bir değişme olup olmadığını da bu özellik ile öğrenebiliriz.

// Örnek: Modified

Procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);

Var

c: integer;

Begin

if edit1.modified then // Edit kutusu değişti ise

begin

*c := application.MessageBox ('Değişiklikler kaydedilsin mi?',
'Çıkış', Mb_YesNoCancel + Mb_IconQuestion);*

case c of

IdYes: // Kayıt için gerekli kod;

IdCancel: Action := caNone; {Kapatmayı durdur}

end;

end;

end:

Left, Top, Width, Height :

Bu özellikler çalışma zamanı ekranda görülen bütün kontrollerde bulunur ve nesnenin sol ve üst koordinatlarının , genişliğini ve yüksekliğini belirler.

Align:

Bu özellik, kontrolü formun istenilen kısmına (sağına, soluna, altına, üstüne, tüm form alanını kaplayacak şekilde) alınmasını sağlar. Formun boyutları değiştiğinde Align özelliği verilen kontrolde Formun durumuna göre yeniden boyutlanacaktır. Örneğin formun sürekli en altında bulunmasını istediğiniz bir Edit kontrolüne **alBottom** değerini verirseniz form hangi boyutta olursa olsun Edit kontrolü de formun genişliğine ayarlanacak ve sürekli olarak formun en altında bulunacaktır. Align özelliği aşağıdaki değerlerden birini alır:

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

Şekil 8.4'teki formu oluşturarak kullanıcının, Edit kontrolünü Sola, Sağa, Üste ve Alta yerleştirebilmesi imkanı verelim.

// Örnek: Align

```
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;
```

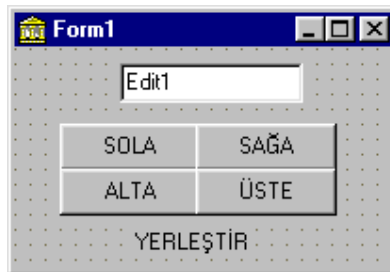
```
end;
```

```
procedure TForm1.Button4Click(Sender: TObject);
```

```
begin
```

```
    edit1.Align:=AlTop;
```

```
end;
```



Şekil 10.4. Align kontrolü

Visible:

Visible özelliği kontrolün ekranda görülür veya görülmez olmasını sağlar. Visible özelliği **False** yapılarak kontrolün ekran üzerinde görülmemesi sağlanabilir.

```
Edit1.visible:= False;
```

satırı ile bir kontrolü gizleyebilir, **True** vererek de tekrar görüntüleyebilirsiniz. Böylece Form üzerinde sadece bazı şartlar gerçekleştiğinde gösterilmesini istediğiniz kontrolleri de gösterebilirsiniz.

Örneğin bir sınav programında cevabın bulunduğu kontrolü gizleyerek ancak sınav bittikten sonra görüntülenmesini sağlayabilirsiniz.

Showing:

Visible özelliğinin **true** yapılması kontrolün her zaman görünür olmasını sağlamaz. Örneğin kontrol başka bir kontrolün (panel gibi) içine yerleştirilmişse (içine yerleştirildiği kontrole parent diyoruz) ve içinde bulunduğu nesnenin Visible özelliği **false** ise bu kontrolde görülmeyecektir. Kontrolün form üzerinde görülüp görülmediği **Showing** özelliği ile öğrenilir. Bu özellik değiştirilemez. Parentin **Visible** özelliğinin değişmesi ile bu Showing özelliği kendiliğinden değişir.

Bir nesneye SetFocus metodu ile kontrol verilmeden önce bu özellikle o nesnenin görünür olduğunu denetlemek gerekir.

Enabled:

Bu özellik kontrolün aktif veya pasif durumda bulunmasını sağlar. Enabled özelliğinin **true** olması kullanıcının kontrol özelliği üzerinde işlem yapabilmesini, **false** olması ise yapamamasını sağlar. Visible özelliğinden farklı olarak kullanıcı kontrolü ekranda görür ancak üzerinde işlem yapamaz.

```
procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
```

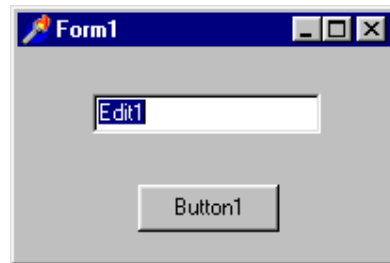
```
    if edit1.enabled = true then
```

```
        edit1.enabled := false
```

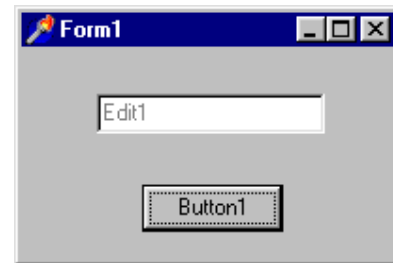
```
    else
```

```
        edit1.enabled := true;
```

```
end;
```



a) True konumu



b) False konumu

Şekil 10.5. Enabled özelliği

BorderStyle:

Bu özellik, kontrolün etrafında bir çerçevenin gösterilip gösterilmemesini sağlar.

bsSingle değerinin verilmesi etrafında bir çerçevenin olmasını,

bsNone olması bu çerçevenin bulunmamasını sağlar.

Hint, ShowHint:

Bir çok programda üzerine gittiğiniz nesnelerin altında küçük balonlarla o nesnenin ne iş yaptığını açıklayan bir mesaj görmüşsünüzdür.

İsterseniz, siz de kontrollere küçük açıklamalar ekleyebilir ve kullanıcının bunun üzerinde kısa bir süre fare ile durduğunda, açıklamanızın görüntülenmesini isteyebilirsiniz. Açıklamanızın ne olduğunu Hint özelliği ile, bu açıklamanın gösterilip gösterilmeyeceğini de ShowHint özelliğini **True** yada **False** yaparak belirleyebilirsiniz.

Aşağıda verilen örnekte form yüklendiği zaman Edit1 kutusunun Hint özelliği belirlenmektedir. Fare ile Edit1 kutusu üzerinde dolaşıldığı zaman açıklama kutusu Edit1'in hemen altında gözükmektedir.

// Örnek : Hint, ShowHint

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

```
begin
```

```
    Edit1.hint := 'Adınızı Yazınız';
```

```
    Edit1.showhint := true;
```

```
end;
```

Bu kodu yazdıktan sonra Edit1 üzerinde fare ile bir müddet beklerseniz küçük bir kutuda açıklama yazıldığını göreceksiniz.

Name:

Name özelliği kontrol elemanlarının ismini ifade eder. Eğer kullanıcı tarafından Name özelliği değiştirilmezse Delphi, form üzerine alınan her elemana bir ad verir. Edit1, Edit2, Button1, Checkbox1, Label1 gibi.

Bu özellik sadece tasarım anında **Object Inspector** penceresinden değiştirilebilir. Bu özellik ile belirlediğiniz isimle o kontrolün özellik ve metodlarına ulaşabilirsiniz. Örneğin Edit1 kutusunun **Text** özelliğine **Edit1.Text** ile ulaşırken, Name özelliğini BilgiGiris yaparsanız artık Text özelliğine **BilgiGiris.Text** ile ulaşmanız gerekir.

TabStop:

TabStop özelliği **True** olan bir kontrol elemanına Tab tuşu ile ulaşılabilir. **False** ise Tab tuşu ile ulaşamaz. Özellikle kullanıcının sürekli olarak bilgi girdiği formlarda her zaman kullanmadığı kontrollerin **TabStop** özelliklerini **False** yaparak kullanıcıya hız kazandırabilirsiniz.

TabOrder:

Bu özellik form üzerinde bulunan kontrollerin Tab tuşu ile erişme sırasını belirler. Delphi form üzerine alınan ilk elemanın **TabOrder** özelliğini 0, ikincisini 1, üçüncüsünü 2 vb. yapar. Fakat bu sıra TabOrder özelliği ile değiştirilebilir.

Color:

Bu özellik kontrolün zemin rengini belirler. Delphi’de kullanılabilen standart renkler şöyledir.

Değer	Anlamı	Değer	Anlamı
ClBlack	Siyah	ClSilver	Açık Gri
ClMaroon	Koyu Kırmızı	ClRed	Kırmızı
ClGreen	Yeşil	ClLime	Açık Yeşil
ClOlive	Zeytin Yeşili	ClBlue	Mavi
ClNavy	Koyu Mavi	ClFuchsia	Pembe
ClPurple	Mor	ClAqua	Aqua
ClTeal	Teal	ClWhite	Beyaz
ClGray	Gri	ClYellow	Sarı

SelStart, SelLength, SelText :

Windows’ta seçme, kopyalama, kesme ve yapıştırma işlemleri çok kullanılan faydalı özelliklerdir. Delphi’de kullanıcının giriş yapabildiği kontroller bu özellikleri standart tuşlarla herhangi bir komuta gerek kalmaksızın desteklerler. Programla da bu işleri yapmak mümkündür. Bu işlemler yapılırken seçili olan kısmı belirlemek için **SelStart** ve **SelLength** özellikleri kullanılır. **SelStart** seçimdeki ilk karakterin yerini

verir. İlk karakter için bu değer 0 dır. **SelText** özelliği ise seçilen kısmın içeriğini öğrenmeye ve değiştirmeye yarar.

Örnek olarak Edit kutusunda seçilen kısmı büyük-küçük harfe çevirecek bir program yazalım. Örneğimiz için Şekil 10.6'daki formu oluşturalım.

// Örnek : *SelLength*, *SelText*

```
procedure Tform1.Button1Click(Sender: TObject);
```

```
begin
```

```
    If Edit1.SelLength = 0 Then
```

```
        ShowMessage ('Çevrilecek kısmı seçiniz.')
```

```
    Else
```

```
        Edit1.SelText := AnsiUpperCase(Edit1.SelText);
```

```
end;
```

```
procedure Tform1.Button2Click(Sender: TObject);
```

```
begin
```

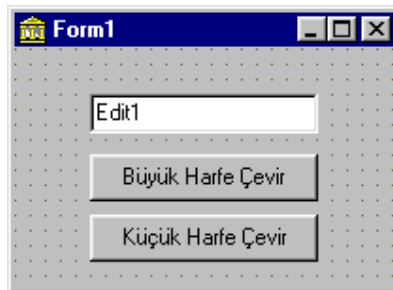
```
    If Edit1.SelLength = 0 Then
```

```
        ShowMessage ('Çevrilecek kısmı seçiniz.')
```

```
    Else
```

```
        Edit1.SelText := AnsiLowerCase(Edit1.SelText);
```

```
end;
```



Şekil 10.6. Oluşturulacak form

Ctl3D:

Bu özellik bir kontrol elemanın üç boyutlu gözüküp gözükmeyeceğini belirtir. Ctl3D'nin **True** olması durumunda ilgili eleman üç boyutlu, **False** olması durumunda

ise ilgili eleman normal gözükecektir. Üç boyutlu görünüm her zaman daha iyi bir görünüm oluştururken hız açısından dezavantajı vardır.

ComponentIndex:

Sadece çalışma zamanı değiştirilebilen bu özellik form üzerindeki elemanların sıra numarasını verir. Form üzerine alınan ilk elemana 0, ikinci elemana 1 indeksi verilerek böylece tüm elemanların index numarası saptanmış olur.

Cursor :

Cursor özelliği farenin o nesne üzerinden geçerken alacağı şekli belirler. Bu şekiller kum saati, ok, çift başlı ok, dört başlı ok ... olabilir.

DragCursor, DragMode :

Bu özellik **Sürükle-Bırak** (Drag-Drop) işlemlerinde kullanılır.

HelpContext:

Yardım dosyasındaki ilgili konu numarası. Bu numara yardım dosyası tasarlanırken belirlenir ve kontrol üzerindeyken F1 tuşuna basıldığında bu numaralı konu **WinHelp** programı aracılığı ile görüntülenir.

Owner:

Sadece çalışma zamanı kullanılan bu metot, hangi elemanın hangi elemana ait olduğunu belirler.

Form üzerindeki tüm elemanlar forma ait iken form da kendisini kullandıran uygulamaya (**TApplication**) aittir. Bir **owner** hafızadan atıldığında onun altında bulunan kontrollerde atılır.

Parent :

Bu özellik üzerinde bulunan kontrolün adını içerir. Bir kontrol elemanının parenti üzerinde yapılan bazı işlemler aynen o kontrol elemanına da yansır. Mesela **GroupBox** içerisine alınan elemanlar, GroupBox'un taşınması, saklanması,

gösterilmesi gibi olaylardan aynen etkilenirler. Normalde bir grup içine konmazsa hepsinin parenti Form'dur.

ParentColor:

Eğer ParentColor özelliği **True** ise bir kontrol elemanı üzerinde bulunduğu parentin rengini alır, değilse kendisine ait renkte kalır.

ParentCtl3D:

ParentCtl3D özelliğinin True olması halinde, kontrol elemanı üzerinde bulunduğu elemanın üç boyutlu özelliğini kullanır. Eğer bu özellik False ise kontrol elemanı kendi **Ctl3D** özelliğini kullanır.

ParentFont:

Bu özelliğin True olması halinde, kontrol elemanı üzerinde bulunduğu elemanın Font özelliklerini kullanır. Eğer bu özellik False ise kontrol elemanı kendi Font özelliğini kullanır.

ParentShowHint:

Bu özellik, bir kontrol elemanının kısa yardım açıklamasını nerede bulacağını belirler. Eğer ParentShowHint True ise form üzerindeki tüm kontroller kendi parentlerinin ShowHint özelliklerini kullanırlar. Eğer ParentShowHint özelliği **False** ise kontroller kendi ShowHint özelliklerini kullanırlar. Bu şu kolaylığı sağlar: Her eleman için ShowHint özelliğini **True** yapmak yerine, sadece kendi parentlerinin ShowHint özelliğini **True** yapmak yeterli olacaktır.

PopupMenu:

PopupMenu özelliği herhangi bir eleman üzerinde iken farenin sağ tuşuna basıldığında aktif olacak olan popup menüyü tanımlar. **Object Inspector** penceresinde iken bu özellik kullanıldığında, form üzerindeki PopupMenu elemanının ismi buradan direkt aktif hale getirilebilir. Menülerin anlatıldığı kısımda bir menünün nasıl oluşturulduğu gösterilecektir.

Tag:

Bu özellikle LongInt tipinde bir sayı saklanabilir. Bu özellik Delphi için hiçbir anlam ifade etmez. Kullanıcı bu özelliği global bir değişken gibi kullanabilir. Formdaki bütün prosedürler kontrolleri kullanabilecekleri için bazı durumlarda global değişken tanımlamak yerine bu özellikleri kullanmak daha faydalıdır.

Örneğin kullanıcı **Başla** komut düğmesine bastığında formun başlığında bir sayıyı sıfırdan başlayarak sürekli olarak artırsın ve formun başlığına yazsın. Ta ki kullanıcı **Dur** düğmesini tıklayana kadar.

Başla düğmesine yazacağımız kod ne zamana kadar çalışacak: Kullanıcı **Dur** düğmesine basana kadar. Bu ikisi farklı olaylar olduğu için birinde saymaya devam ederken, diğerinde dur düğmesine basıldığını bir değişkene değer atayarak karar verebiliriz.

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;

Bu şekilde **Basildi** değişkenini global olarak tanımlayarak Dur düğmesine basılınca kadar kodun çalışmasını sağlayabiliriz. Aynı işi **Basildi** değişkenini

tanımlamadan Button2'nin **Tag** özelliğini bir global değişken gibi de kullanarak yapabiliriz. Bu durumda kodumuz şöyle olacaktır:

```
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;
```

```
Tform1.Button2Click(Sender: TObject);
```

```
begin
```

```
    Button2.Tag := 1; // Basılı olduğunu bildirdik
```

```
end;
```

Böylece global bir değişken tanımlamaya gerek kalmadan aynı işlemi yapmış olduk. Burada Button2 kontrolünün **Tag** özelliğini global bir değişken gibi kullanmış olduk.

Methods

Show, Hide:

Hide metodu kontrolü gizler, Show metodu ise kontrolü gösterir. Yani aşağıdaki iki satır da aynı işi yapar.

```
Edit1.Show;
```

```
Edit1.Visible := True;
```

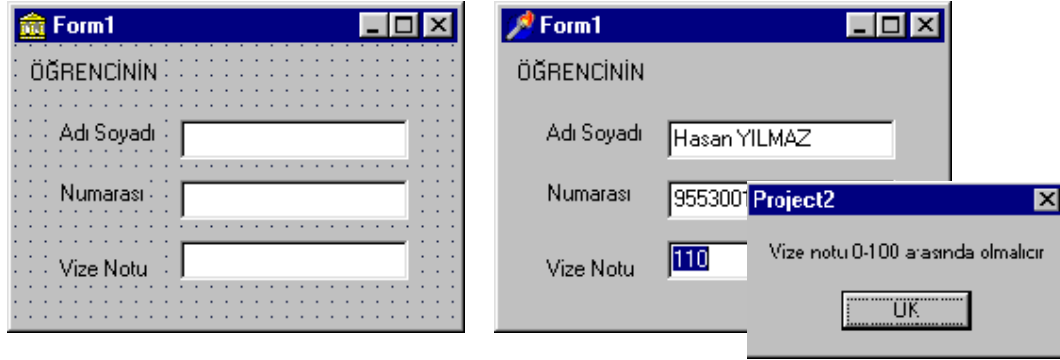
ve bu iki satırda aynı işi yapar.

```
Edit1.Hide;
```

```
Edit1.Visible := False;
```

SetFocus:

Kullanıcı kontroller arasında Tab tuşu ile veya Mouse ile geçiş yapılabilir. Aynı iş SetFocus metodu ile de yapılabilir. Yani bu metotla klavye kontrolü istenen bir kontrole bırakabilir.



Şekil 10.7. Hazırlanan form ve programın çıktısı

Örnek olarak yukarıdaki formda vize notunun girilmesini istediğimiz Edit3 kutusuna geçersiz bir not girilmesi halinde kontrolü tekrar Edit3 kutusuna bırakarak kullanıcıyı geçerli bir not girmeye zorlayalım.

Geçerli bir not girilip girilmediğini kullanıcı Edit3'den başka bir kontrole geçerken yapacağımız için kodumuzu Edit3'ün **OnExit** olayına yazacağız.

// Örnek: SetFocus

Procedure Tform1.Edit3Exit(Sender: TObject);

begin

If (StrToInt(Edit3.Text)<0) or (StrToInt(Edit3.Text)>100) then

Begin

ShowMessage('Vize notu 0-100 arasında olmalıdır');

Edit3.SetFocus;

End;

end;

Programda Vize Notunun girileceği Edit3 kutusuna 0-100 arasında olmayan bir not girmediğimizde program bizi uyaracak ve kontrolü tekrar Edit3 kutusuna bırakacaktır. Böylece bu kutuya 0-100 arası bir not girmeden diğer kutulara geçiş yapılamayacaktır.

CanFocus:

Bir nesneye kontrolü bırakabilmek için o nesnenin Visible özelliğinin ve Enabled özelliğinin **True** olması ayrıca o kontrol bir başka kontrol içinde bulunuyorsa o nesnenin de bu özelliklerinin **True** olması gerekir. Bütün bu özelliklerin aktif olup olmadığını CanFocus metodu ile öğrenebiliriz. Eğer bu metodun sonucu **True** ise kontrolü alabilir, **False** ise alamaz.

Yukarıdaki örnekte bu özellikleri False yapmadığımı bildiğim için bunu kontrol etme ihtiyacı duymadım. Ancak büyük çaplı programlarda bu kontrolü yaptıktan sonra **SetFocus** yapmak gerekir.

```
If Edit1.CanFocus Then Edit1.SetFocus;
```

Focused:

Focused özelliği, şu anda kontrolün bu nesnede olup olmadığını belirler. Kontrol bu nesnede ise sonuç **True**'dur.

Clear, ClearSelection:

Clear metodu kontrolün içeriğini, ClearSelection ise seçili olan kısmı siler.

Yani: *Edit1.Clear ;*

```
Edit1.Text := ' ';
```

ve

```
Edit1.ClearSelection;
```

```
Edit1.SelText := ' ';
```

satırları aynı işi yapar.

SelectAll:

Bu metod TEdit ve TMemo kontrollerinin içindeki tüm veriyi seçer.

```
Edit1.SelStart := 0;
```

```
Edit1.SelLength := Length(Edit1.Text);
```

Satırları ile

```
Edit1.SelectAll;
```

satırı aynı işi yapar.

GetTextLen:

Bu metot Text özelliği içindeki karakter sayısını verir. Length komutu ile aynı işi yapar.

Label1.Caption := 'Edit1 kutusunda' + IntToStr(Edit1.GetTextLen) + 'tane harf var';

Satırı ile Edit1 kutusunda kaç harf olduğunu Label1 içinde yazıyoruz. Aynı işi Length komutu ile aşağıdaki gibi de yapabiliriz.

Label1.Caption := 'Edit1 kutusunda' + IntToStr(Length(Edit1.Text)) + 'tane harf var';

CopyToClipboard, CutToClipboard, PasteFromClipboard :

Delphi'de kullanıcı Text özelliği olan kontrollerde kesme, kopyalama yapıştırma işlemlerini standart tuşlarla (Ctrl-X, Ctrl-C, Ctrl-V) yapabilmektedir. Gerektiğinde program kodları ile de bu işlemleri yapmak mümkündür.

CopyToClipboard: Seçili kısmı panoya kopyalar. Seçili kısım yoksa herhangi bir işlem yapılmaz.

CutToClipboard: Seçili kısmı silerek panoya kopyalar. Seçili kısım yoksa herhangi bir işlem yapılmaz.

PasteFromClipboard: Panodaki Metin bilgisini seçili kısım varsa seçili kısmın üzerine, seçili kısım yoksa kursörün bulunduğu konuma yapıştırır.

SetSelTextBuf (Metin:PChar):

Bu özellik de SelText özelliği gibi seçili kısmı değiştirmeye yarar, tek farkı bu özellik **string** tipinde değil **Pchar** tipindeki metinler kullanılır.

GetSelTextBuf (Buffer; Pchar; BufSize; Integer) :

Bu metot, bir memo yada bir text elemanında seçili olan ifadeyi Buffer olarak gösterilen yere kopyalar. Kopyalanacak maksimum karakter sayısını Bufsize belirler. Metottan geri dönen değer kopyalanan karakter sayısıdır.

BringToFront, SendToBack:

Bu metotlar üst üste gelmiş kontrolleri öne veya arkaya almaya yarar.

SetBounds (Sol, Üst, Genişlik, Yükseklik):

Left, Top propertiesleri ile kontrolün koordinatlarını, Width, Height propertiesleri ile de kontrolün boyutlarını değiştirebileceğimizi daha önce gördük. Bunların hepsini birden yapmak için SetBounds metodu kullanılabilir. Bu yöntemi kullanmak propertieslerle yapmaktan daha hızlıdır.

Edit1.SetBounds (0,0,150,100);

ScaleBy (Çarpan, Bölen):

Bu metod aracılığıyla bir kontrol elemanı belli bir oranda küçültülüp, büyütülebilir. Bu işlemde kontrolün hem yüksekliği hem de genişliği değişir. Burada Çarpan ve Bölen parametreleri tamsayıdır ve kontrolün boyutlarının kaçta kaç büyütülüp küçültüleceğini belirler. Örneğin yarıya indirmek için Çarpan:1 ve Bölen 2 olması gerekir, yani 1/2. %60'ına küçültmek için Çarpan:60 ve Bölen:100 olarak belirlenebilir. Bu yöntem forma uygulanırsa formun içindeki bütün kontroller aynı oranda büyüyüp küçülür.

Free:

Free metodu form üzerinde bulunan bir nesneyi siler hafızadan atar.

Edit1.free; {Edit1 hafızadan atılır}

Refresh :

Refresh metodu ilgili elemanı ilgili kontrolün görüntüsünü yeniler. Örneğin hareket eden bir **Image** kutusunun içindeki resim hareket sırasında görülmeyecektir. Hareket ederken içeriğinin görüntülenebilmesi için Refresh metodu ile kendisini yenilemesi gerektiğinin belirtilmesi gerekir.

// Örnek: Refresh

procedure TForm1.Button1Click(Sender: TObject);

var

i: integer;

begin

for i := 0 to width do


```
Begin
    Image1.Left := Image1.Left + 1;
    Refresh;
End;
End;
```

Yukarıdaki örnekte Refresh metodunu kullanmazsanız Image kutusu hareket ederken içeriği görüntülenmeyecektir.

Repaint:

Repaint metodu ilgili elemanı yeniden çizdirir.

```
Edit1. Repaint;
```

Update:

Update metodu bir kontrolü güncelleştirir. Özellikle FileListBox gibi kontrollerde, diskten bir dosya silindikten sonra bu değişikliğin liste kutusunda etkili olması için bu metotla güncellenmeleri gerekir.

Events (Olaylar)

Nedir bu olaylar? Windows'ta; bir nesneyi tıklamak (Click, DblClick, MouseDown), Mouse'un bir nesne üzerine gelmesi halinde açıklama ifadelerinin görüntülenmesi (MouseMove), Iconların taşınması olayı (Drag), Bir dosyanın fare ile tıklanarak bir yerden başka bir yere alınması olayı (Drag Drop), Mouse'un bazı bölgelerde gezinmesi halinde şeklinin değişmesi (DragOver), Mouse'un tuşları basılı tutularak çizim yapılması (MouseDown, MouseUp) vb. gibi yapılan her hareket bir olayı temsil etmektedir.

- Form ve Form üzerinde bulunan her kontrol elemanı çift tıklanarak o elemana ait default olay prosedürüne ulaşılabilir.
- Diğer olaylar prosedürlerine ise **Object Inspector** penceresinin **Events** tabı kullanılarak ulaşılır. **Object Inspector** penceresinde bir olayın karşısı çift tıklanırsa Delphi olaya default ismini verir.

- İstenirse **Object Inspector** penceresinden, daha önce yazılan olay alt programlarının adı da verilebilir. Bu durumda bir kaç kontrol aynı olay alt programını çalıştırır.
- Her bir olayın kendine özgü giriş parametreleri vardır. Her olayda standart olarak Sender parametresi bulunur. Bu parametre o olayın hangi kontrol tarafından oluşturulduğunu belirler. Bir çok kontrol aynı olayı paylaşıyorsa, olayın hangi kontrol tarafından oluşturulduğu Sender parametresi ile öğrenilir. Örneğin Edit1 ve Edit2 aynı olayı paylaşıyorsa

If Sender = Edit1 Then

If Sender = Edit2 Then

kodları ile olay hangi kontrol tarafından oluşturulduysa ona göre kod yazılabilir.

OnClick(Sender: TObject):

Kontrol üzerinde farenin sol tuşu ile tıklanması durumunda bu olay meydana gelir. Ancak bazı kontrollerde bu olayı boşluk tuşu, enter ve kısa yol tuşu da meydana getirebilir. Genel olarak mouse ile, yapılan tıklama işleminin görevini klavyeden de yapan tuşlar bu olayı meydana getir. Örneğin bir komut düğmesinin kısa yol tuşuna basılması veya kontrol komut düğmesindeyken boşluk veya enter tuşlarına basılması da bu olayı meydana getirir.

OnDblClick (Sender: TObject):

Farenin sol tuşu ile çift tıklanması durumunda bu olay meydana gelir.

Click, DblClick olaylarına örnek olarak bir Edit kutusu üzerine tıklandığında yazı rengi, çift tıklandığında da zemin renginin değişmesini sağlayacak bir program yazalım.

```
procedure TForm1.Edit1Click(Sender: TObject);
```

```
begin
```

```
    Edit1.Font.Color := random (1000000); // rasgele bir renk
```

```
end;
```

```
procedure TForm1.Edit1DblClick(Sender: TObject);
```

```
begin
```

```
    Edit1.Color := random (1000000);
```

end;

OnMouseDown (Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer) :

Fare kontrol elemanı üzerinde iken farenin tuşlarından birine basılmasıyla meydana gelir. Click olayından farklı olarak farenin sadece sol tuşu değil sağ ve orta tuşları da bu olayı meydana getirir. Ve fare koordinatları, shift tuşlarının durumu hakkında da bilgi verir. Button parametresi şu üç değeri alarak hangi tuşa basıldığı hakkında bilgi verir.

mbRight: Sağ fare tuşu.

mbLeft: Sol fare tuşu.

mbMiddle: Orta fare tuşu.

Shift parametresi ise shift tuşlarının durumu hakkında bilgi verir. Bu parametrenin alabileceği değerler OnKeyDown olayında anlatılmış.

X,Y parametreleri de farenin kontrol içindeki koordinatlarını verir. Bu koordinatların değeri kontrolün sol üst köşesi 0,0 kabul edilerek verilir. Form üzerindeki gerçek koordinatları bulmak için kontrolün Left, Top özellikleri ile toplanmalıdır.

OnMouseMove (Sender: TObject; Shift; TShiftState; X, Y: Integer):

Fare kontrol üzerinden geçerken bu olay meydana gelir. Parametrelerin anlamları MouseDown olayındaki gibidir.

OnMouseUp (Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer) : Fare kontrol elemanı üzerinde iken basılan fare tuşu bırakıldığında bu olay meydana gelir. Parametrelerin anlamı MouseDown olayındaki gibidir.

OnKeyPress (Sender: TObject; var Key: Char):

OnKeyPress olayı klavyeden basılan ASCII kodu bulunan bir tuşa basıldığında meydana gelir. Yani bu olay Control, Alt, Shift, CapsLock, F1 vb. gibi tuşlarda meydana gelmez. Bu tuşları ancak OnKeyDown ve OnKeyUp olayları algılayabilir.

Bu olay daha çok kullanıcının girdiği karakterleri kontrol etmek için kullanılır. Hangi tuşa basıldığı **Key** parametresi ile öğrenilir. Olay, basılan karakter kontrol içine gönderilmeden çalıştığı için bu tuş değiştirilebilir.

Örneğin bir **Edit** kutusuna kullanıcının sadece rakam girmesini istiyorsak bu olaya yazacağımız kodla diğer tuşları iptal etmemiz gerekir. Basılan bir karakteri iptal etmek için o karakterin ASCII kodunu 0 yapmak yeterlidir. Buda Key parametresine #0 değeri atanarak yapılır. Örnek olarak Edit1 kutusuna sadece rakam ve Edft2 kutusuna da sadece harf girişi yapılabilecek bir program yazalım.

// Örnek: KeyPress

```
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
```

```
begin
```

```
  If (Key< '0') or (Key> '9') Then
```

```
    Key := #0; {Rakam değilse iptal et}
```

```
end;
```

```
procedure TForm1.Edit2KeyPress(Sender: TObject; var Key: Char);
```

```
begin
```

```
  If (UpperCase(Key)< 'A') or (UpperCase(Key)> 'Z') Then
```

```
    Key := #0;
```

```
end;
```

OnKeyDown(Sender: TObject; var Key: Word; Shift: TshiftState):

Bu olay, klavyeden bir tuşa basıldığında meydana gelir ve basılı tutulduğu müddetçe meydana gelmeye devam eder. KeyPress olayından önce meydana gelir ve KeyPress olayının tanımadığı tuşları da tanır. Bu tuşların genel bir listesi için aşağıdaki tabloya bakınız.

Key: Basılan tuşun Scan kodu. Bu tuşların sembolik isimleri kullanılabilir.

Sembolik isimler ve karşılıkları şöyledir:

Key	Anlamı	Key	Anlamı
Vk_Tab	Tab	Vk_Multiply	*
Vk_Return	Enter	Vk_Add	+
Vk_Pause	Pause	Vk_Subtract	-
Vk_Escape	Esc	Vk_Divide	/

Vk_Space	Boşluk	Vk_Decimal	.
Vk_A, Vk_Z	A .. Z harfleri	Vk_0, Vk_9	0 .. 9 rakamları

Shift : Bu parametre ile Shift, Control, Alt tuşları ve farenin hangi tuşlarının basılı olduğunu öğrenebiliriz. Bu parametre aşağıdaki değerleri alır:

Shift	Anlamı
SsShift	Shift tuşu basılı
SsAlt	Alt tuşu basılı
SsCtrl	Ctrl tuşu basılı
Ssleft	Farenin sol tuşu basılı

OnKeyUp (Sender: TObject; var Key: Word; Shift: TShiftState):

Bu olay basılan tuş bırakıldığı anda meydana gelir. Her klavye eventin’de oluşmaz. Şöyle ki A harfine uzun bir süre basarak birden çok A girişi yapılır. Bu giriş esnasında KeyUp olayı sadece son A harfinin yazılmasından sonra meydana gelir.

OnEnter (Sender: TObject):

OnEnter olayı bir kontrol elemanının aktif olmasıyla meydana gelir. Bir nesneye kontrolün geçebilmesi için görülebilir (Visible) ve aktif (Enabled) olması gerekir. Birden fazla form arası geçiş yapılırken ve o anda çalışan diğer uygulamalardan bu uygulamaya geçiş yapılırken OnEnter olayı meydana gelmez.

OnExit (Sender: TObject):

OnExit olayı kontrol elemanının kontrolü kaybetmesiyle meydana gelir. OnEnter olayında olduğu gibi birden fazla form arası geçiş yapılırken ve Windows’un uygulamaları arasında geçiş yapılırken OnExit olayı meydana gelmez .

Örneğin kontrol Edit1 kutusuna geçtiği zaman, zemin rengi kırmızı, kontrolü kaybettiği zaman ise zemin rengi yeşil olmasını sağlayacak bir program yazalım.

// Örnek : Enter,Exit

procedure Tform1.Edit1Enter(Sender: TObject);

begin

```
edit1.color := clgreen;  
end;  
procedure TForm1.Edit1Exit(Sender: TObject);  
begin  
    edit1.color := clred;  
end;
```

- Bir eleman, kontrolü ya Tab tuşuyla ya da fare ile tıklanmasıyla alır. Tab tuşu elemanlar arasında geçişi sağladığından dolayı elemanlara kontrolü verme veya geri alma olaylarında rol oynar.
- OnExit olayının gerçekleşmesi için form üzerinde birden fazla kontrol elemanı yer almalıdır. Form üzerindeki menülerin seçilmesi OnExit olayını meydana getirmez.
- Bu olay daha çok kontrolü kaybeden elemanın içeriğini değerlendirmek için kullanılabilir.

10.1.2. TButton (Komut Düğmesi)

Bir olayın kullanıcı tarafından başlatılması için programlarda çok kullanılan kontrollerden biridir.

Properties

Caption:

Caption özelliği düğmenin üzerinde yazılı olan ifadeyi belirler. Kısa yol tuşu atamak için kısa yol olması istenen tuşun önüne & işareti konur. Önünde bu işaret bulunan harfin altı çizilir. Böylece kullanıcı Alt tuşu ile birlikte bu tuşa basarak komut düğmesini aktif hale getirir.

Cancel:

Bu özelliğin, **true** yapılması, formun herhangi bir yerinde basılan ESC tuşuyla bu düğmenin aktif hale geleceğini belirler.

Default:

Bu özelliğin **true** yapılması, formun herhangi bir yerinde basılan Enter tuşuyla bu düğmenin aktif hale geleceğini belirler.

ModalResult :

Bu özellik, bir form Modal olarak gösterilmişse etkili olur. Formun Modal olması o forma cevap vermeden programdaki diğer formlara geçilememesi demektir. Örneğin Mesaj kutuları, InputBox bilgi giriş kutusu ve bunun gibi kutular modaldir ve bu kutuya cevap vermeden program çalışmaya devam etmez ve bu pencere kapanmadan programdaki diğer formlara ulaşamaz. Bir formun ShowModal özelliğini kullanarak modal olmasını sağlayabilirsiniz. Mesaj kutularını düşünürsek kutuda hangi düğmenin seçildiğini öğrenebiliyorduk. Burada da Modal özelliğin verildiği formlarda komut düğmelerinin ModalResult özelliği aşağıda verilen değerlere göre koda gerek kalmaksızın formu kapatır ve kendisinin seçildiğini formun ModalResult özelliği ile çağrıldığı programa bildirir.

mrNone: Özellik aktif değil.

mrOk, mrCancel, mrAbort, mrRetry, mrIgnore, mrYes, mrNo.

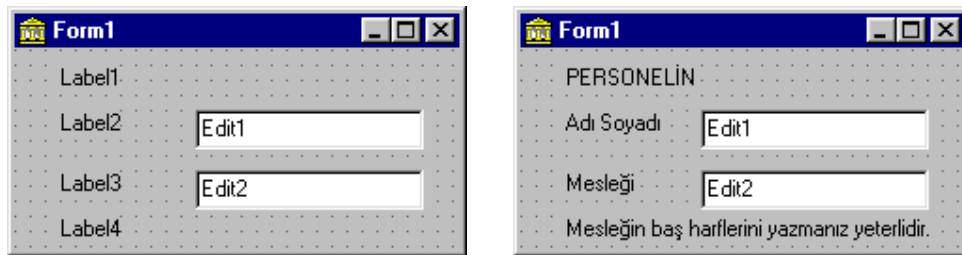
Örnek olarak bir programdan çıkarken ekrana gelen “değişiklikler kaydedilsin mi?” mesaj penceresini verebiliriz. Bu mesaj penceresine cevap vermeden programı çalıştırmayı devam ettiremeyiz.

Events**OnClick (Sender:TObject):**

Komut düğmesinin en önemli olayı budur. Düğme tıklandığında yapılması gereken işler burada tıklanır.

10.1.3. TLabel (Etiket)

Label kontrolü genellikle form üzerinde açıklama yazmak veya başka bir kontrolün ne işe yaradığını belirtmek için kullanılır.



Şekil 10.8. Label kontrolünün kullanılması

Properties

Caption:

Bu özellikle Label'in içinde yazılı olan metin belirlenir.

Label1.Caption := 'Adınızı giriniz';

Komut düğmesinin Caption özelliğinde olduğu gibi kısa yol tuşunu bu özellikle belirlemek için yine kısa yol tuşu olacak harfin önüne & işareti konur. Ancak bunun kullanılıp kullanılmayacağını **ShowAccelChar** ve **FocusControl** özellikleri belirler.

ShowAccelChar:

Bu özellik True yapılırsa Caption özelliğinde kullanılan & karakterinin kısa yol tuşu tanımlamak için kullanılacağı belirlenir.

Label1.Caption := '&Label'; Label1.ShowAccelChar := True;

FocusRect :

Label nesnesine verilen kısa yol tuşu Label'in misyonu gereği Label'in **Click** olayını meydana getirmez. Label genellikle diğer nesnelere açıklama yazmak için kullanıldığından dolayı kısa yol tuşunun açıkladığı kontrole kontrolü vermesi istenir.



Şekil 10.9. Label'a kısa yol tuşu atama

Yukarıdaki formda Alt+A ya basılması durumunda kontrolün Edit1'e ve Alt+M ye basıldığında da Edit2'ye verilmesi istenir. Bu işi yapmak için Label1'in FocusRect özelliğine Edit1 ve Label2'ninkine de Edit2 verilir.

Label1.FocusRect := Edit1;

Label2.FocusRect := Edit2;

Bu özelliğin etkili olması için Label'ın **Caption** özelliğinde kısa yol tuşu tanımlanmış olmalı ve Label'ın **ShowAccelChar** özelliğinin **True** yapılmış olması gerekir.

AutoSize :

Bu özellik **True** yapılırsa Label'ın içindeki yazı değiştiğinde Label'ın boyutları da içindeki yazıya göre yeniden boyutlanır.

WordWrap:

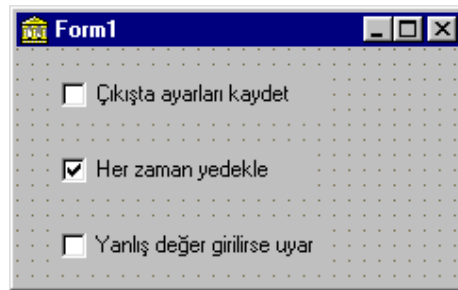
Bu özellik **True** yapılırsa Label kontrolüne girilen ifadenin Label kontrolünün boyutunu aşması durumunda, bir alt satıra geçerek devam etmesini sağlar. Eğer yazı Label'ın boyutlarına ulaştığı halde kelime bitmemişse Label'ın boyutları kelimenin bittiği yere kadar genişler. Bu özelliğin etkili olabilmesi için **AutoSize** özelliğinin de **True** olması gerekir.

Transparent:

Bu özellik nesnenin üzerinde bulunduğu konuma uymasını sağlar. **True** ise üzerinde bulunduğu nesnenin görülmesini sağlar, yani bir cam üzerine yazılmış yazı gibi davranır. **False** ise kendi zemin rengi üzerinde bulunur ve altındaki nesneyi göstermez.

10.1.4. TCheckBox (İşaret Kutusu)

Kullanıcının bir seçeneği aktif veya pasif duruma getirmesi için kullanılan bir kontroldür.



Şekil 10.10. CheckBox kontrolünün kullanılması

Bu kontrol üç durumdan birinde bulunabilir.

Checked: İşaretli

Unchecked: İşaretsiz

Grayed: Bu durum onun temsil ettiği şeyin belirsizliğini gösterir. Örneğin CheckBoxun bir metnin Bold olup olmadığını temsil ettiğini düşünelim. Seçilen yazının tamamı Bold ise bu kutunun işaretli olması, bir kısmı Bold bir kısmı değilse kutunun içeriği belirsiz olması gerekir.

Properties

Checked:

Checked özelliği CheckBox kontrolünün işaretli mi, işaretsiz mi olduğunu öğrenmeye ve değiştirmeye yarar. True ise işaretli, False ise işaretsiz yapar.

```
CheckBox1.Checked := true; // işaretli
```

```
CheckBox1.Checked := false; // işaretsiz
```

State:

Checked özelliği gibi CheckBox'un durumunu öğrenmeye ve değiştirmeye yarar. Checked özelliğinden farklı olarak **Grayed** durumuna da getirebilir. Şu üç değerden birini alabilir.

```
CheckBox1.State := cbChecked;
```

```
CheckBox1.State := cbUnchecked;
```

```
CheckBox1.State := cbGrayed;
```

AllowGrayed:

AllowGrayed özelliği kullanıcının Grayed durumuna getirip getiremeyeceğini belirler. False ise kullanıcı CheckBox'u tıkladığında işaretli ise işaretsiz, işaretsiz ise işaretli duruma geçer. Yani kullanıcı bu iki durumdan birine getirebilir. True yapılırsa kullanıcı Grayed durumuna da getirebilir. CheckBox'un temsil ettiği şeye göre bu değeri belirlemeniz gerekir. Örneğin bir yazının Bold özelliğini temsil eden CheckBox'un bu özelliğinin False olması gerekir.

Alignment:

Bu özellik Caption özelliği ile belirlenen yazının, işaretin sağına mı soluna mı yazılacağını belirler.

Checkbox1.Alignment := taLeftJustify; // soluna

Checkbox1.Alignment := taRightJustify; // sağına

Events

OnClick (Sender: TObject):

CheckBox'un işareti değiştiğinde bu olay meydana gelir. CheckBox için her zaman **Click** olayına kod yazmak gerekmez. Örneğin bir Edit kutusunun font bold özelliğini belirleyen CheckBox'un Click olayına bu işi yapacak kodu yazmak gerekir. Ancak "Çıkışta ayarlar kaydedilsin" gibi bir Check kutusunun **Click** olayına bir kod yazmak gerekmebilir. Çünkü o anda yapılacak bir işi değil çıkarken yapılacak bir işi temsil etmektedir. Bu durumda çıkışta CheckBox'un Checked özelliğine bakılarak durumuna göre kaydedilebilir.

Şimdi bir satış işleminde KDV'nin fiyata dahil edilip edilmeyeceğini bir CheckBox ile belirleyeceğimiz program yazalım. Bunun için Şekil 8.11'deki formu oluşturarak aşağıdaki komutları yazalım.

procedure TForm1.CheckBox1Click (Sender: TObject);

var

f: real;

begin

*f := StrToInt (Edit1.Text) * StrToInt (Edit2.Text);*

// CheckBox işaretli ise

if CheckBox1.Checked Then

// Beyaz eşya ise KDV %25 değilse %15

if RadioButton1.Checked Then

*Edit3.Text := FloatToStr (f + f*25/100)*

else

*Edit3.Text := FloatToStr (f + f*15/100)*

else

Edit3.Text := FloatToStr (f);

end;

a) Hazırlanan form

b) Programın Çıktısı

Şekil 10.11. Örnek program

10.1.5. TGroupBox (Gruplama Kutusu)

Bu kontrol tek başına değil diğer kontrolleri gruplamak için kullanılır. Kontrolleri bu kontrol ile gruplamanın bir çok avantajı vardır.

Bu çerçeveler içine konan kontroller, çerçeveye bağımlıdır ve konumları bu çerçeve dışına taşamaz. Özellikle birkaç kontrolü birden görünür veya görünmez yapmak için hepsinin Visible özelliğini tek tek değiştirmek yerine çerçevenin Visible özelliği değiştirilerek çerçeve içindeki tüm kontroller aynı anda görünmez yapılabilir. Aynı durum taşıma için de geçerlidir her birini tek tek taşımak yerine, çerçeve taşınır. Çerçevelerin buna benzer birçok faydaları vardır. Özellikle Radio Button'larının kullanılmasında çerçeve kullanmak zorunlu hale gelebilir.

GroupBox'larla gruplanan kontrollerin koordinatları artık forma göre değil grup kontrolünün sol üst köşesine göre belirlenir. Form üzerine yerleştirilmiş bir kontrolü taşıyarak bir GroupBox kontrolü üzerine getirmekle o kontrol gruplanmış olmaz. GroupBox kontrolü içerisine bir kontrol yerleştirirken önce GroupBox kutusunu seçin. Eğer zaten form üzerine yerleştirilmiş bulunan kontrolleri GroupBox kontrolü içine

almak istiyorsanız o zaman form üzerindeki kontrolleri kesin (cut) ve GroupBox kontrolünü seçtikten sonra buraya yapıştırın (paste).

GroupBox'lar aşağıda olduğu gibi RadioButton (Seçenek Düğmesi) kontrollerini gruplamak için kullanılırlar.

Şekil 10.12. GroupBox kullanılmış form

Ayrıca GroupBox, kontrolleri sadece bazı şartlarda gösterilmesi gereken kontrolleri de bir arada tutarak bunların kolayca gizlenip/gösterilmesini sağlarlar. Örneğin yukarıdaki form dizaynında Mezun Olduğu Fakülteyi temsil eden RadioButton düğmelerinin sadece Tahsil=Üniversite olması durumunda gösterilmesi gerekir. Mezun Olduğu Fakülteyi belirten altı tane seçenek düğmesini gizleyip, göstermek için sadece onun içinde bulunduğu GroupBox4 kontrolünü gizleyip göstermek yeterlidir. Aşağıdaki kod yazılırsa, sadece tahsili üniversite olanlar için mezun olduğu fakülteyi gösteren seçenek düğmeleri görülecektir.

// Örnek: GroupBox

```
procedure TForm1.FormCreate (Sender: TObject);
begin
    GroupBox4.Visible := false;
end;

procedure TForm1.RadioButton3Click (Sender: TObject);
begin
    GroupBox4.Visible := false;
end;

procedure TForm1.RadioButton4Click (Sender: TObject);
```

```

begin
    GroupBox4.Visible := false;
end;
procedure TForm1.RadioButton5Click(Sender: TObject);
begin
    GroupBox4.Visible := false;
end;
procedure TForm1.RadioButton6Click(Sender: TObject);
begin
    // tahsil üniversite ise fakültesini göster
    GroupBox4.Visible := true;
end;

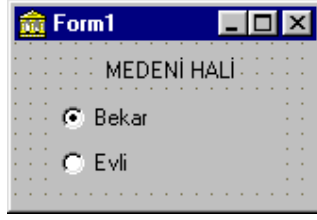
```

Yukarıdaki formda tahsil olarak Lise veya daha düşük bir seçenek seçildiğinde aşağıdaki gibi (Şekil 10.13) Mezun Olduğu Fakülteyi gösteren grup kutusu izlenecektir.

Şekil 10.13. Mezun olunan fakülteyi gösteren grup kutusu gizlenmiş.

10.1.6. TRadioButton (Seçenek Kutusu)

RadioButton kontrolü CheckBox'dan farklı olarak bir kaç seçenekten birini seçme imkanı veren bir kontroldür. Bu kontrolün tek başına kullanılması anlamsızdır. Bir kaç seçenekten birini seçme imkanı veren bir kontrol olduğu için en az iki tanesi birlikte kullanılmalıdır. Gruptaki Radio düğmelerinden biri seçildiğinde diğeri kendiliğinde seçilmiş özelliğini kaldırır. Yani aynı anda bir grupta iki tane işaretli düğme bulunmaz.



Şekil 10.14. Radio düğmeli seçenek kutusu

Yukarıdaki formda düğmelerden biri seçildiğinde diğerinin seçilmişliği kendiliğinden kalkacaktır. Öğrenim durumunu da aynı formda düğmelerle göstermek istersek GroupBox kontrolünü kullanmalıyız. GroupBox kontrolünün kullanımına dair örnek Şekil 10.12’ de verilmiştir.

Properties

Checked:

Checked özelliğinin True olması RadioButton düğmesinin seçili olduğunu gösterir. Şekil 10.14’teki Bekar RadioButton düğmesinin Checked özelliği **True** yapılmıştır.

10.1.7. TRadioGroup

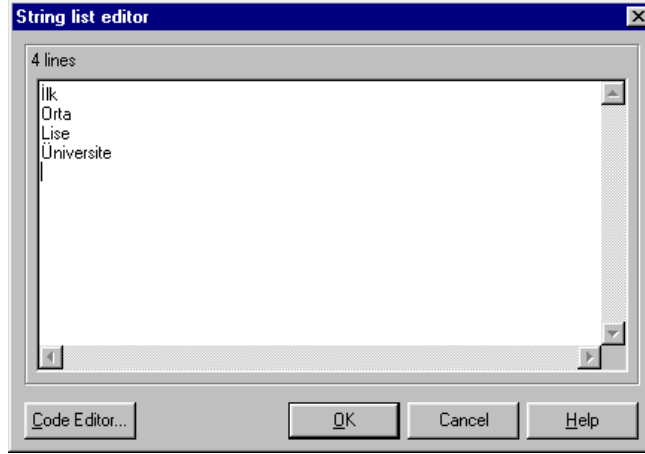
Bu kontrol Radio düğmelerini gruplamak için değil istenen sayıda Radio düğmesini çalışma zamanı oluşturmak için kullanılabilecek bir kontroldür.

Bir Memo kontrolüne satır ekliyormuş gibi bu kontrolle tasarım veya çalışma zamanı kolayca yeni düğmeler oluşturulabilir. Oluşturulan bu düğmelerin ayrı ayrı isimleri yoktur. RadioGroup nesnesinin ismiyle temsil edilir ve ItemIndex özelliği ile hangi düğmenin seçildiği öğrenilebilir.

Properties

Items:

Radio grup kontrolüne bu özellik kullanılarak yeni düğmeler eklenebilir. Bu özellik tasarım zamanında Object Inspector penceresinden çift tıklanırsa aşağıdaki pencere (Şekil 9.15) açılacaktır. Bu pencere kullanılarak RadioGroup içinde bulunacak düğmeler kolayca oluşturulabilir.



Şekil 10.15. String list editor penceresi

ItemIndex:

Seçili olan elemanın numarası bu özellik ile öğrenilebilir.

Columns:

Bu özellik düğmelerin kaç sütunda gösterileceğini belirler.

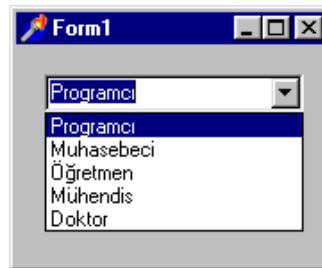
Events

OnClick (Sender: TObject):

RadioGroup nesnesinin **Click** olayı çerçevenin tıklanması ile değil içindeki bir düğmenin tıklanmasıyla meydana gelir. Hangi düğmenin tıklandığı ise ItemIndex özelliği ile öğrenilebilir.

10.1.8. TComboBox (Aşağı Doğru Açılan Liste)

Aşağı doğru açılabilen bir liste kontrolüdür. Genellikle, değerleri daha önceden belli olan elemanların seçimi için kullanılırlar. Listedeki elemanlardan sadece seçileni ekranda görüntüler.



Şekil 10.16. Örnek ComboBox

ComboBox kontrolüne eleman ekleme ve silme işlemi RadioGroup kontrolünde olduğu gibidir.

Properties

Style:

Aşağıdaki değerlerden birini alarak ComboBox'un stilini belirler.

csDropDown: Aşağıya doğru açılabilen bir ComboBox oluşturulur. Kullanıcı bilgi girişi yapabilir. Kutudaki tüm veriler aynı yüksekliktedir.

csDropDownlist: Aşağıya doğru açılabilen bir combobox oluşturulur. Kullanıcı bilgi girişi yapamaz. Kutudaki tüm veriler aynı yüksekliktedir.

csSimple: Aşağıya doğru açılmayan bir ComboBox oluşturulur. Kullanıcı bilgi girişi yapabilir. Aşağıya doğru açılmadığı için kullanıcı yukarı ve aşağı ok tuşları ile bir seçim yapabilir.

csOwnerDrawFixed: Aşağıya doğru açılabilen bir ComboBox oluşturulur. Kullanıcı bilgi girişi yapamaz. Kutudaki verilerin yükseklikleri **ItemHeight** özelliğiyle değiştirilebilir. Bu tipteki ComboBoxlar grafik nesneler de içerebilir.

csOwnerDrawVariable: Aşağıya doğru açılabilen bir ComboBox oluşturulur. Kullanıcı bilgi girişi yapamaz. Kutudaki verilerin yükseklikleri farklı olabilir. Bu tipteki comboboxlar grafik nesneler de içerebilir.

DropDownCount: Bu özellik Combobox aşağıya doğru açılırken gösterilecek olan eleman sayısını belirtir. Eğer herhangi bir değer belirtilmezse varsayılan değer sekizdir.

MaxLenght: csDropDown, csSimple tipindeki ComboBox'lara kullanıcı bilgi girişi yapabilmekteydi. Bu özellik, istenirse kullanıcının giriş yapabileceği karakter sayısı sınırlanabilir. 0 verilirse sınır kaldırılır.

Text: Combo kutusunda seçilen eleman radiogroup kontrolünde olduğu gibi Items ve ItemIndex özelliği ile öğrenilebilir. Ancak csDropDown, csSimple tipindeki combo kutularında kullanıcı listede olmayan elemanlardan da yazabileceği için kutuda seçilen veya yazılan elemanın bu özellik ile öğrenilmesi gerekir. Ayrıca bu özellik kutuda gösterilmesi istenen eleman da belirlenebilir.

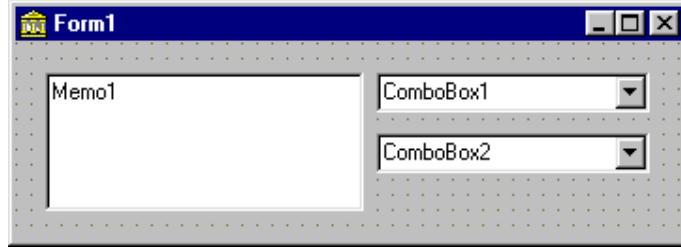
Events

OnClick (Sender: TObject):

Kullanıcın kutudan bir elemanı seçmesi ile bu olay meydana gelir.

OnChange (Sender: TObject):

Kullanıcın kutudan bir elemanı seçmesi veya kutuya bir şey yazması ile bu olay meydana gelir. Örnek olarak kullanıcının bir Memo kontrolündeki yazının font adını ve font büyüklüğünü iki ComboBox kontrolü ile seçmesini sağlayacak bir program yazalım. Örneğimiz için aşağıdaki formu oluşturalım.



Şekil 10.17. Oluşturulacak örnek form

// Örnek: ComboBox

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

```
var
```

```
    i: Integer;
```

```
begin
```

```
    ComboBox1.Items := Screen.Fonts; // 1.kutuya ekran fontlarını ekle
```

```
    {2.kutuya font büyüklüklerini ekle}
```

```
    For i := 8 to 30 do
```

```
        ComboBox2.Items.Add (IntToStr(i));
```

```
        {Memo kontrolünün Font adını kutuda göster}
```

```
        ComboBox1.Text := Memo1.Font.Name;
```

```
        {Memo kontrolünün FontSize'ını kutuda göster}
```

```
        ComboBox2.Text := IntToStr(Memo1.Font.Size);
```

```
        {Font isimlerinin bulunduğu kutuyu sırala}
```

```
        ComboBox1.Sorted := True;
```

```
end;
```

```
procedure TForm1.ComboBox1Change (Sender: TObject);
```

```
begin
```

```
    {Kutuda seçileni veya yazılanı Memo1'in fontu yap}
```

```
    Memo1.Font.Name := ComboBox1.Text;
```

```

end;
procedure TForm1.ComboBox2Change (Sender: TObject);
begin
    {Kutuda seçileni Memo1'in FontSize'ı yap}
    Memo1.Font.Size := StrToInt (ComboBox2.Text);
end;

```

Properties

Caption:

Menünün başlığını belirlemek için bu özellik kullanılır. Diğer Caption özelliklerinde olduğu gibi & işareti ile kıs ayol tuşu da verilebilir. Ayrıca Caption özelliğine - (eksi işareti) verilirse bu menü bir kesme çizgisi olarak oluşturulur.

Checked :

True yapılarak menünün işaretli olması sağlanabilir. Örneğin ‘Çıkışta ayarları kaydet’ isimli bir menü seçeneğimiz olsun. Bu menü seçildiğinde işaretlenmesini ve tekrar seçildiğinde de işaretinin kalkmasını isteyelim. Bunun için menüyü uygun şekilde oluşturup **Click** olayına;

```

A1.Checked := Not A1.Checked;

```

kodunu yazalım. Bu menü seçildiğinde işaretli ise işaretini kaldıracak, işaretli değilse işaretleyecektir.

RadioItem:

Menüler birer Option Button (RadioButton) gibi de kullanılabilir. Eğer bazı menülerin RadioItem özellikleri True verilmişse bu menüler bir seçenek düğmesi gibi hareket eder. Yani gruptaki bir menü seçildiğinde diğer menünün işareti otomatikmen kalkar. Yani aynı grupta sadece bir menü işaretlenebilir.

Hangi menülerin grup oluşturacakları o menülerin GroupIndex özellikleri ile belirlenir. GroupIndex özellikleri aynı olan menüler bir RadioButton gibi davranır. O halde menülerin grup olarak hareketlerini sağlamak için iki şey yapılmalıdır.

- Grupta bulunması istenen menülerin RadioItem özellikleri true yapılmalıdır.
- Grup olması istenen bütün menülerin GroupIndex özelliklerine aynı sayılar verilmelidir.

Bu işlemlerden sonra menülerin seçilebilmesi için Checked özelliğinde anlatılan gibi gerekli kodu **Click** olayına yazmak gerekir.

Hint:

Diğer kontrollerin Hint özelliğine bir metin atayarak kullanıcının fare ile kontrol üzerinde kısa bir süre beklediğinde bir açıklama gösterilmesini sağlayabiliriz. Menü kontrolünün de Hint özelliği ile bu açıklama metni belirlenir. Ancak kullanıcı menünün üzerinde beklemesiyle bu açıklama görüntülenmez. Bu menüdeki açıklamalar özel bir yöntemle bir başka kontrolün içinde gösterilmesi gerekir.

Application nesnesinin OnHint özelliği kullanılarak bir menü seçildiğinde açıklamayı gösterecek kodun nerede olduğu belirlenebilir. Bunun için Formun **Create** olayında,

```
Application.OnHint := AltPrgAdi;
```

şeklinde bir satırla Hint olayında hangi alt programın çalışacağı belirlenir.

Daha sonra bu alt programda, açıklamayı bir başka kontrol içinde göstermek için gerekli kod yazılır.

```
procedure TForm1.AltPrgAdi (Sender: TObject);  
begin  
  label1.Caption := Application.Hint;  
end;
```

ve bu alt programın tanımı formun Private veya Public bölümünde yapılır.

```
private  
procedure AltPrgAdi (Sender: TObject); gibi.
```

Default:

Menülerin alt menülerinden birine default özelliği verilebilir. Bir menüye default özelliği verildiğinde onun üst menüsü çift tıklandığında otomatik olarak default özelliği verilen alt menü aktif hale gelecektir.

10.1.9. TPopupMenu

Bu kontrol elemanı yapı olarak MainMenu elemanı gibidir. Aynı yöntemle PopupMenu tasarımı yapılır. MainMenu elemanından farklı olarak formun başlığında değil herhangi bir yerinde açılabilir.

Bu menü normal bir menü gibi tasarlandıktan sonra hangi kontrolle birlikte açılması isteniyorsa o kontrolün PopupMenu özelliğine tasarlanan bu menünün adı verilir. Böylece ilgili eleman üzerinde iken farenin sağ tuşuna basıldığında PopupMenu'süne ulaşılır.

Örneğin formun herhangi bir yerinde iken farenin sağ tuşuna basıldığında PopupMenu'nün aktif hale gelmesi için Object Inspector penceresinde formun PopUpMenu özelliğine bu menünün adı verilir.

10.2. Additional Kontrol Elemanları



Şekil 10.18. Additional kontrol elemanları paleti



TBitBtn: Üzerinde resim bulunması istenen komut düğmeleri bu kontrolle oluşturulur. Ayrıca standart işlemleri yapan birçok düğme hazır olarak tanımlandığı için bu düğmelerden biri de seçilebilir.



TSpeedButton: Bu komut düğmesi daha çok araç çubuklarında kullanılır. Bu düğmenin en önemli özelliği grup halinde çalışabilmesi ve basılı durumda kalabilmesidir.



TImage: Resim göstermek için kullanılan bir kontrol elemanıdır.



TBevel: Daha çok programa görsel güzellik kazandırmak için kullanılır. Bu kontrol aracılığı ile oluşturulan çerçeveler, kontrolleri görsel olarak gruplandırmak için kullanılır.



TScrollBox: Kaydırma çubukları bulunan bir kutu oluşturur. Bu kontrol içine kaydırma çubuklarını desteklemeyen diğer kontroller yerleştirilerek bunların kaydırılabilmesi sağlanır.



TStringGrid (Izgara kontrol elemanı): Ders programı gibi tablolar oluşturmak için kullanılır.



TMaskEdit: Formatlı bilgi girişi gereken durumlarda Edit kontrolü yerine kullanılır.



TDrawGrid: Mevcut veri yapılarını satır ve sütun şeklinde gösterebilen bir kontroldür. TStringGrid elemanı ile aynı özelliklere sahiptir.



TCheckBox: İçerdiği elemanları birer işaret kutusu ile gösteren bir elemandır. ListBox kontrolüne göre daha kullanışlıdır.



TSplitter: Boyutlandırılabilen elemanları çalışma zamanında boyutlandırma imkanı sağlayan bir kontroldür.



TStaticText: Boyutlandırılabilen bir kontrol olup form üzerinde verileri görüntüler.



TChart: Verileri grafiksel olarak göstermeye yarayan çok amaçlı bir elemandır. Grafikler hem çalışma zamanı hem de tasarım zamanı oluşturulabilir.

10.2.1. TBitBtn (Resimli Komut Düğmesi)

Bu kontrol standart Button kontrolünün resimli halidir. Bu kontrol aracılığı ile komut düğmesinin üzerine resim yerleştirilebileceği gibi Kind özelliği ile hazır düğmelerden biri de seçilebilir.

Properties

Kind:

Bu özellik ile hazır düğmelerden biri seçilebilir. Bu özelliğe verilen değere göre daha önce anlattığımız Default, Cancel ve ModalResult özelliğine de etkisi vardır. Aşağıdaki tabloda özelliğin aldığı değerler ve bu değerlere karşılık gelen düğmeler ve etkilenen diğer özellikler de görülmektedir:

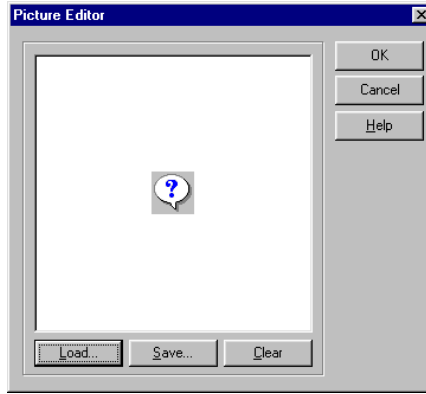
Kind	Düğme Adı	Default/Cancel	ModalResult
BkOk	OK	Default	MrOk
BkCancel	Cancel	Cancel	MrCancel
BkYes	Yes	Default	MrYes
BkNo	No	Cancel	MrNo
BkHelp	Help	-	MrNone
BkClose	Close	Default	MrClose

BkAbort	Abort	Cancel	MrAbort
BkRetry	Retry	-	MrRetry
MkIgnore	Ignore	-	MrIgnore
MkAll	All	Default	MrAll
BkCustom	Programcı tarafından belirlenir.		

Close düğmesi seçildiğinde herhangi bir koda gerek kalmadan form kapatılır, **Help** düğmesi de programın HLP dosyası tanımlanmışsa HelpContextID özelliği ile belirlenen konuyu WinHelp aracılığı ile görüntüler. **Custom** tipindeki düğmede ise resmi ve yazıyı programcı Glyph ve Caption özellikleri ile belirler.

Glyph:

Kind özelliği BkCustom olan düğmelerde, düğme üzerinde gösterilecek resmi belirler. Tasarım zamanı Object Inspector penceresinden çift tıklayarak aşağıdaki açılan pencere ile yükleyebilirsiniz.



Şekil 10.19. Picture Editor penceresi

Çalışma zamanı ise LoadFromFile metodu ile yüklenebilir.

BitBtn1.Glyph.LoadFromFile ('deneme.bmp');

NumGlyph:

Aslında bu düğmeye sadece bir resim değil, düğmenin değişik durumlarında gösterilmesi için dört resim yüklenebilir. Bu işlem, dört ayrı resim dosyasıyla yapılmaz. Tek bir resim dosyası bir kaç parçaymış gibi ele alınarak yapılır. Bu özellik **Glyph** özelliği ile belirlenen resmin kaç parçaya bölüneceğini belirtmek için kullanılır. Her bir resim düğmenin değişik bir durumu için kullanılır.

Resmin birinci parçası düğmenin normal durumu için, ikinci parçası pasif durumu (Enabled = False) için, üçüncü parçası düğme basılı iken ve son parçası da basılı kaldığı sürece gösterilir. (Dördüncü duruma BitBtn düğmesi geçmez. Bu değer SpeedButton kontrolünde etkilidir.)

Etkileşimli düğmeler oluşturmak için kullanılabilecek bir özelliktir.

Layout:

Layout özelliği düğme üzerindeki resmin yerleşim biçimini belirler.

Spacing:

Spacing özelliği düğme üzerindeki resim ile düğme başlığı arasındaki boşluğu belirler.

Margin:

Bu özellik düğme içindeki resmin ve yazının sol köşeye olan uzaklığını belirler. -1 verilirse her zaman ortada gösterilir.

10.2.2. TSpeedButton Kontrol Elemanı

TSpeedButton kontrolü tek başına kullanıldığı zaman TButton kontrolünden farklı bir durum arz etmez. Bu kontrol RadioButton düğmeleri gibidir. Gruptaki bir düğme basılı iken, ikinci bir düğmeye basıldığında, basılı olan düğme kendiliğinden kalkar, son basılan düğme basılı durumuna geçer. Yani bir grupta ancak bir düğme basılı olabilir. Diğer komut düğmelerinden farklı olarak basılan düğme basılı olarak kalır. Bu düğmeleri gruplamak için GroupBox kontrolleri değil bu kontrollerin GroupIndex özellikleri kullanılır. GroupIndex özellikleri aynı olan kontroller birlikte hareket eder. Ayrıca BitBtn düğmesinde olduğu gibi bu düğmeye de resim verilebilir.

Bu tip düğmeleri birçok popüler Windows programında görebilirsiniz. ToolBar' lar üzerinde bulunan düğmeler işte bu tipten düğmelerdir.

Properties

GroupIndex:

Bu kontrolün diğer bir özelliği de grup halinde çalıştırılabilmesiydi. GroupIndex özelliği aynı değerde olan düğmeler birlikte hareket ederler. Yani bu düğmelerden ancak biri basılı kalabilir. Gruptaki bir düğmeye basıldığında gruptaki basılı olan diğer düğme kendiliğinden kalkar. Bu düğmeleri birçok programda görebilirsiniz. Örneğin

kelime işlemcilerde sola, sağa, ortaya ve iki yana yasla düğmelerinden birini seçtiğinizde diğerleri bundan etkilenir.

Örneğin bir yazının Kalın, Eğik, AltÇizgi, Sola, Sağa ve Ortaya özelliklerini temsil edecek 6 tane SpeedButton kontrolümüz olsun. Yazının Kalın, Eğik ve AltÇizgi özellikleri birbirinden bağımsız özellikler oldukları yani bir özellik diğerini etkilemediği için GroupIndex'leri birbirinden farklı olmalıdır (1,2,3 gibi). Sola, sağa ve ortaya özellikleri ise aynı anda bulunamazlar. Bunlardan biri seçildiğinde diğerinin seçilmişliğinin kalkması gerekeceği için bu üçünün GroupIndex özellikleri aynı ve diğerlerinden farklı olmalıdır (4,4,4 gibi).

AllowAllUp:

True ise gruplandırılmış düğmelerin hiç biri basılı konumda bulunmayabilir. Basılı düğme üzerine basılarak düğme kaldırılabilir. False ise gruptaki düğmelerden biri mutlaka basılıdır. Basılı bir düğmeyi kaldırabilmek için gruptaki diğer düğmeye basmak gerekecektir. Eğer grupladığınız düğmelerin temsil ettikleri değerlerden birinin mutlaka aktif olması gerekiyorsa bu özelliği False yapın. Örneğin bir kelime işlemcide; yazılı kısım sağa, sola, ortaya veya iki yana konumlarından mutlaka birinde bulunması gerekir. Yani AllowAllUp özellikleri false olmalıdır.

Down:

Düğmenin basılı olup olmadığı bu özellik ile öğrenilir. Ayrıca AllowAllUp özellikleri false olan düğmelerin bulunduğu grupta düğmelerden birinin mutlaka basılı olması gerektiği için başlangıçta hangi düğmenin basılı olması isteniyorsa o düğmenin Down özelliği **True** yapılır.

10.2.3. TImage (Resim Gösterme Elemanı)

Image elemanı resimleri göstermek için kullanılır. Gösterilecek resim Image kontrolünün Picture özelliğine atanır.

Properties

Picture:

Bu özellik, Image kontrolü içinde gösterilecek olan resmi belirler. Bu özellik aracılığıyla Image nesnesinde icon, metafile ve bitmap resimleri gösterilir. Image kontrolüne şu şekillerde resim eklenir:

- Tasarım zamanında image elemanının Object Inspector penceresindeki Picture özelliğinden; Bu özelliğin yanındaki düğmeyi tıklarsanız açılacak penceredeki (Şekil 9.19) Load düğmesi vasıtası ile göstermek istediğiniz resim dosyasını seçebilirsiniz.

- LoadFromFile metodu ile bir dosyadan;

Image1.Picture.LoadFromFile ('buton.bmp');

- Başka bir kontrolün picture özelliğinden;

Image1.Picture := Image2.Picture;

- Panodan

Uses Clipbrd; // panoyu kullanabilmek için bu satırı ekleyin

Image1.Picture.Assign (Clipboard());

Image kontrolündeki resmi kaldırmak için ise iki yöntem kullanılabilir.

Bunlardan biri resmin için boyamak, diğeri ise **nil** değerini atamak.

Image1.Picture.bitmap.canvas.FillRect (ClipRect);

yada

Image1.Picture := nil;

Stretch:

Stretch özelliği True ise resim Image kontrolünün içine sığdırılır. Yani resim büyükse küçültülür, küçükse büyütülür.

AutoSize:

Bu özellik True ise Image kontrolü resmin boyutlarına ayarlanır. Yani Stretch özelliğinin tersini yapar.

Center:

True verilirse resim Image kontrolünün içinde ortalanır. False ise sol üst köşede gösterilir.

Canvas:

Image kontrolünün Canvas özelliğini kullanarak çizimler yapabilir ve resmi parçalayarak bu parçalar üzerinde işlemler yapabilirsiniz.

10.2.4. TBevel

Bevel elemanı uygulamalarda çizgi, kutu veya çerçeve tipinde görüntüler oluşturmak için kullanılabilir. Bu kontrolün yaptığı özel bir iş yoktur. Bir süsleme elemanı olarak veya ekrandaki kontrollerin bazılarını diğerlerinden görsel olarak ayırmak için kullanılabilir.

Properties

Style:

Style özelliği Bevel elemanının kabartmalı yada oymalı biçimde gözükeceğini belirler. BsRaised (kabartmalı), BsLowered (oymalı) değerlerinden birini alır.

Shape:

Bu özellik Bevel elemanın göstereceği şekilleri belirler. Shape elemanın aldığı değerler; bsBox (kutu), bsFrame (çerçeve), bsBottomLine (alt çizgi), bsTopLine (üst çizgi), bsLeftLine (sol çizgi) ve bsRightLine (sağ çizgi) dir.

10.2.5. TMaskEdit (Formatlı Bilgi Giriş Kutusu)

Bu kontrol standart Edit kontrolüne benzer. Ancak Edit kutusu formatlı girişlerin gerektiği durumlarda fazla kullanışlı değildir. Örneğin kullanıcının doğum tarihini girmesi gereken bir kutuda kullanıcının doğru girip girmediğini Edit kutusu ile anında kontrol etmek daha zordur. MaskEdit kontrolü bilgilerin özel bir düzende girilmesi gerektiği durumlarda hem kullanıcıya hem de programcıya büyük kolaylıklar sağlar.

Properties

Kullanıcının girebileceği format bu özelliklerle belirlenir.

> Sonraki karakterler büyütülür.

< Sonraki karakterler küçültülür.

<> Herhangi bir değişim uygulanmaz. < veya > karakterlerinden sonra normale çevirmek için kullanılır.

\ Özel karakterleri göstermek için kullanılır.

L Sadece harf girilebilir. Girilmesi mecburidir.

I Sadece harf girilebilir. Girilmesi mecburi değildir.

A Sadece harf ve rakam girilebilir. Girilmesi mecburidir.

a Sadece harf ve rakam girilebilir. Girilmesi mecburi değildir.

C Herhangi bir karakter girilebilir. Girilmesi mecburidir.

c Herhangi bir karakter girilebilir. Girilmesi mecburi değildir.

0 Sadece rakam girilebilir. Girilmesi mecburidir.

9 Sadece rakam girilebilir. Girilmesi mecburi değildir.

Rakam veya +,- işaretleri girilebilir.

: Saat, dakika, saniye ayırıcısı olarak kullanılır.

/ Gün, ay, yıl ayırıcısı olarak kullanılır.

; Birden fazla maske için araya konur.

Bunların haricindeki karakterler aynen kutuda görülür. Edit Mask özelliğine boş string atanırsa maske iptal edilir.

Örneğin $f(x)=ax^n + bx + c$ şeklinde bir formül girişi sağlamak için;

MaskEdit1.Edit Mask := 'f(x) = 9x^9 + 9x + 9';

Saat girişini sağlamak için;

MaskEdit1.Edit Mask := '99:99';

Tarih girişini sağlamak için;

MaskEdit1.Edit Mask := '99/99/99';

Yazmak gerekir.

IsMasked:

Bu özellik True ise giriş için maske belirlenmiştir.

EditText, Text:

Maskede belirtilen ancak kullanıcının girmediği karakterler _ karakterleri ile görüntülenir. Örneğin 'aaaaaaaa' olsun. Yani dokuz tane harf veya rakam girilebilsin.

Kullanıcı bu maskeyle belirlenmiş kutuya sadece xz karakterlerini şu şekilde girmişse;

“__xz__”

Text özelliğinin değeri ' xz ',

EditText özelliğinin değeri ise ' __xz__ ' olacaktır.

10.3. System Kontrolleri



Şekil 10.20. System kontrol elemanları paleti



TTimer: Belirli zaman dilimlerinde çalışması gereken işlemleri yapmak için kullanılır.



TpaintBox: Çizim ve boyama kutusu olarak kullanılır.



TmediaPlayer: Multimedia dosyalarını çalmak için kullanılır.



OLE: Programa OLE'yi destekleyen nesneleri eklemek için kullanılır.



Bu kontroller **DDE** (Dynamic Data Exchange) işlemlerinde kullanılır. Windows altında çalışan programlar **DDE** teknoloji sayesinde birbirleriyle haberleşirler.

10.3.1. TTimer

Zamanın belli aralıklarında OnTimer olayını meydana getiren bir kontroldür. OnTimer olayına yazılan program kodu Interval özelliği ile belirlenen periyotlarla çalışır. Timer kontrolü ile belirli sürelerle bazı kontroller yapılabileceği gibi animasyonlarda yapılabilir. Tasarım anında form üzerinde saat ikonuyla görünen Timer kontrolü çalışma zamanında görünmez.

Properties

Interval:

Interval özelliği OnTimer olayını meydana getirecek olan zaman aralığını ayarlar. Bunun birimi milisaniye olup 0 ile 2.147.483.647 arasında bir değer alır. Bir saniyelik sürelerle çalışması için 1000 vermek gerekir. Verilebilecek maksimum değer ise yaklaşık 25 gün kadar uzun bir süre olabilmektedir.

Enabled özelliği False yapılarak Timer'in çalışması durdurulur.

Events

OnTimer (Sender: TObject):

Timer kontrolünün Interval özelliği ile belirtilen süre içerisinde periyodik olarak bu olay meydana gelir. Bu olaya yazılacak kodun hızlı olması gerekir. Eğer bu olay içerisine yazılan program kodu yavaş ise Windows'ta çalışan diğer programların çalışması da yavaşlayacaktır. Eğer bu olaya yazılan program kodunun çalışması Interval özelliğinin meydana gelmesinden daha uzun sürerse, bu süre içerisinde meydana gelmesi gereken timer olayları meydana gelmez.

Formun başlığında saatin devamlı çalışmasını sağlayan bir program aşağıdaki gibi olmalıdır.

// Örnek : Timer

Procedure TForm1.FormCreate (Sender: TObject);

begin

Timer1.Interval := 1000;

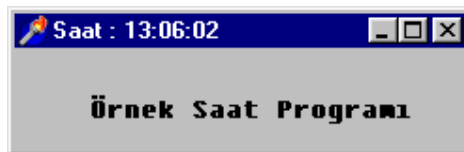
end;

procedure TForm1.Timer1Timer (Sender: TObject);

begin

Form1.Caption := 'Saat: ' + TimeToStr (Time);

end;



Şekil 10.21. Örnek saat programının ekran çıktısı

10.3.2. TPaintBox

PaintBox elemanı form üzerinde dikdörtgensel bir alan belirleyerek grafiksel çizimlerin yapılmasını sağlar. Çizimler sadece bu kontrol içerisinde gerçekleştirilebilir. Bu kontrolün dışındaki bir alanda çizim yapılmasını önler.

PaintBox elemanının Canvas yüzeyi üzerinde küçük bir çizim programı yapalım:

// Örnek : Paintbox

*{ \$R *.DFM }*

var

FareBasilidir: Boolean;

*procedure TForm1.PaintBox1MouseDown (Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);*

var

Alan: TRect;

begin

With PaintBox1 do

Begin

```

    if Button = mbLeft then
    begin
        Canvas.MoveTo (X, Y);
        FareBasilidir := True;
    end;
    {sağ tuşa basılırsa sil}
    if Button = mbRight then
    begin
        alan := Rect ( 0, 0, width, Height);
        Canvas.Brush.Color := clwhite;
        Canvas.FillRect (alan);
    end;
    End;
end;

procedure TForm1.PaintBox1MouseMove(Sender: TObject; Shift: TShiftState; X, Y:
Integer);
begin
    if FareBasilidir then Paintbox1.Canvas.LineTo (x,y);
end;

procedure TForm1.PaintBox1MouseUp (Sender: TObject; Button: TMouseButton; Shift:
TShiftState; X, Y: Integer);
begin
    FareBasilidir := False;
end;

```

Yukarıdaki program bloğuyla farenin sol tuşuna basılması halinde, PaintBox1 kontrolünün Canvas yüzeyi üzerinde, Moveto metodu kullanılarak X, Y aktif noktaları belirlenir. Ve farenin basıldığı bir değişkenle belirlenerek MouseMove olayında basılı olduğu sürece çizim yapılması sağlanır. Program çalıştırıldıktan sonra aşağıdaki gibi bir görüntü oluşturur. Burada farenin sol tuşuyla istenilen çizimler yapılırken, sağ tuşuyla yapılan çizim siliniyor. Buradaki asıl maksat çizim olayının sadece PaintBox elemanının sınırları içerisinde yapılıyor olmasıdır.










10.3.3. TMediaPlayer (Multimedya kontrolü)

MediaPlayer elemanı, Media Control Interface (MCI) sürücülerini kullanmamızı sağlar. Bu eleman aracılığıyla kontrol edilen elemanlar arasında CD-ROM sürücü, ses kartı vb. gibi donanım araçları bulunur. MediaPlayer kontrolü Windows tarafından desteklenen Multimedya işlemlerini kullanıma sokar. Herhangi bir tipteki media aygıtını kullanabilmek için o aygıtın sürücüsünün Windows'a tanıtılmış olması gerekir.

MediaPlayer elemanı bir çok düğmeden oluşmaktadır. MediaPlayer elemanı üzerindeki düğmeler ve bu düğmelerin görevlerini şöyle sıralayabiliriz:



Şekil10.22. MediaPlayer elemanı

Düğme		Değeri	Yaptığı İş
	Play	btPlay	Çalmaya başlar
	Record	btRecord	Kayıt olayını başlatır.
	Pause	btPause	Çalma veya kaydetme olayını durdurur. Tıklandıktan sonra tekrar tıklanırsa çalma yada kaydetme olayını devam ettirir.
	Stop	btStop	Çalma yada kaydetme olayını bitirir.
	Next	btNext	Bir sonraki Track (iz) den çalma olayını devam ettirir. (CD’de sonraki parçaya)
	Prev	btPrev	Bir önceki Track(iz) den çalma olayını devam ettirir. (CD’de önceki parçaya)
	Step	btStep	Okuyucu kafasını bir adım ileri alır.
	Back	btBack	Okuyucu kafasını bir adım geri alır.
	Eject	btEject	Sürücüden CD çıkarılır.

Properties

DeviceType:

DeviceType özelliği, open metoduyla açılacak olan multimedya cihazını veya tipini belirler. DeviceType özelliği için geçerli değerler şunlardır: dtAutoselect, dtAVIVideo, dtCDAudio, dtDAT, dtDigitalVideo, dtMMMovie, dtOther, dtOverlay, dtScanner, dtSequencer, dtVCR, dtVideodisc, dtWaveAudio. Varsayılan değeri dtAutoSelect'tir. Bu durumda medya tipinin ne olduğuna FileName özelliğinin uzantısına bakarak kendisi karar verir. Dosya uzantısı farklı ise veya çalınan birim bir dosyada değilse (CD Audio gibi) bu özellik ile çalınacak birim belirlenmelidir.

Shareable :

Cihaz açılmadan önce birçok uygulama tarafından kullanılıp kullanılmayacağı belirlenmelidir. Normalde False'dır ve diğer uygulamalar MCI aygıtını siz kullandığınız sürece kullanamazlar. True yaparsanız kullanabilirler. Bazı media tipleri shareable olarak çalışmazlar.

Wait:

True verilirse MCI verilen komutu yerine getirinceye kadar kontrolü programa bırakmaz. Bu da üst üste iki MCI komutunun aynı anda gelmemesini sağlar.

AutoOpen:

Medyanın tipi ve dosya adı verildikten sonra Open metodu ile aktif hale getirilmelidir. Bu özelliğe True verilirse Open metodunu kullanmaya gerek kalmadan birim aktif hale getirilir.

AutoRewind:

True ise sona ulaşılmıca otomatik olarak başa alınır. Bu özellik False ise sona ulaşıldıktan sonra Rewind metodu ile tekrar başa alınması gerekir.

EnabledButtons:

Bu özellik, mediaplayer üzerindeki hangi düğmelerin aktif olduğunu öğrenmeye ve değiştirmeye yarar.

VisibleButtons:

VisibleButtons özelliği mediaplayer üzerinde görülebilecek düğmeleri belirler. Normalde bütün düğmeler gözüktür durumdadır. Örneğin sadece Wav dosyalarını çalmak için kullanacaksanız Eject düğmesinin görülmemesini isteyebilirsiniz.

MediaPlayer1.VisibleButtons := MediaPlayer1.VisibleButtons - [btEject];

Display:

Display özelliği, multimedia kontrolünün görüntüleri için kullanacağı çıkış penceresini belirler. Herhangi bir form yada panel kontrolü bir çıkış penceresi olarak kullanılabilir. Display özelliğinin varsayılan değeri "0" dır. Bu değerin sıfır olması demek media kontrolü çıkışlar için kendisine ait bir çıkış penceresi oluşturur.

Bir çıkış penceresine ihtiyaç duyan multimedia cihazları arasında Animation, AVI Video, Digital Video, Overlay ve VCR sayılabilir.

Aşağıda verilen program koduyla 'deneme.avi' video görüntüsü için çıkış penceresi olarak form3 kullanılmaktadır:

```
with MediaPlayer1 do
begin
    FileName := 'deneme.avi'
    Open;
    Display := Form3;
    Form3.Show;
    Play;
end;
```

DisplayRect:

Display özelliği ile çıkış penceresi belirleniyordu. Bu özellik ile de çıkışın (yani bir film ise görüntünün) pencere içindeki koordinatları belirlenebilir. Aşağıdaki gibi bir kodla formun tamamının gösterim için kullanılmasını sağlarsınız. Eğer formunuz tam ekransa aşağıdaki kodla tam ekran AVI oynatabilirsiniz.

```
MediaPlayer1.Display := Form1;
MediaPlayer1.DisplayRect := Rect (0,0,ClientWidth, ClientHeight);
```

Kullanıcının seçtiği bir AVI dosyasını tam ekran oynatacak bir kod yazalım. Örneğimiz için form üzerine MediaPlayer ve OpenFileDialog kontrollerini yerleştirelim.

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

```
    with MediaPlayer1 do
    begin
        DeviceType := dtAutoSelect;
        if not OpenFileDialog1.Execute then exit;
        FileName := OpenFileDialog1.FileName;
```

```

    open;
    display := Form1;
    DisplayRect := rect (0, 0, Form1.ClientWidth, Form1.ClientHeight);
    play;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
    form1.WindowState := wsMaximized;
    Show;
    OpenFileDialog.Filter := 'Avi Dosyaları | *.AVI' ;
    MediaPlayer1.Top := ClientHeight-30;
    MediaPlayer1.Left := (Form1.ClientWidth-MediaPlayer1.Width) div 2;
end;

```

Programı çalıştırdıktan sonra formu tıklarsanız bir dosya seçmeniz istenecektir. Bu işlemten sonra seçtiğiniz AVI dosyası tam ekran oynatılacaktır.

Error:

Sadece çalışma zamanı etkisi kullanılabilen bu özellik Back, Close, Eject, Next, Open, Pause, PauseOnly, Play, Previous, StartRecording, Resume, Rewind, Step veya Stop olaylarından sonra bir hata oluşup oluşmadığını belirtir. Bu özelliğin değeri 0 ise hata oluşmamıştır. Değilse hatanın numarasını verir bu hatanın ne olduğu ise ErrorMessage özelliği ile öğrenilebilir.

ErrorMessage:

Kullanılan metotlar sonucu bir hata oluştu ise hatanın ne olduğunu bu özellik verir. Bu özellikteki metin, kullanılan Windows'un diline göre standart bir mesaj verir.

Aşağıdaki kodu MediaPlayer kontrolüne gönderdiğiniz komutlardan sonra kullanırsanız, bir hata oluşması durumunda bu hata kullanıcıya Türkçe olarak bildirilecektir.

If MediaPlayer1.Error<>0 Then

```
ShowMessage(MediaPlayer1.ErrorMessage);
```

Methods

Open:

Open metodu multimedia cihazını açar. Bu metot kullanılmadan önce çalıştırılacak cihazı belirlemek için, DeviceType özelliğine bir değer atanması gerekir.

```
MediaPlayer1.DeviceType := dtCDAudio;
```

```
MediaPlayer1.Open;
```

Open komutundan sonra çalışmaya hazır hale gelir.

Close:

Close metodu multimedia cihazını kapatır. Bir sonraki dosyayı çalmadan önce öncekinin kapatılması ve sonrakinin açılması gerekir.

Play, Pause, Stop, Next, Previous, Step, Back, Eject, StartRecording:

MCI aygıtı üzerindeki düğmelere kullanıcı basarak ilgili işlemi başlatılabilir. Düğmeler bu metotlarla programdan da aktif hale getirilebilir. Kullanıcının Play düğmesine basmasıyla, aşağıdaki kod aynı işi yapar.

```
MediaPlayer1.Play;
```

Events

OnClick (Sender: TObject; Button: TMPBtnType; var DoDefault: Boolean)

Düğmelerden biri tıklandığında bu olay meydana gelir. Hangi düğmenin tıklandığı Button özelliği ile öğrenilir. Eğer basılan düğmenin üzerine düşen görevi yerine getirmesi istenmiyorsa yani basılan düğmenin göz ardı edilmesi isteniyorsa DoDefault parametresine False değeri atanır.

Aşağıdaki kod basılan düğmenin ne olduğuna bakarak yapılan işlemi Label içerisinde gösterecektir.

```
procedure TForm1.MediaPlayer1Click (Sender: TObject; Button: TMPBtnType; var  
DoDefault: Boolean);
```

```
begin
```

```
case Button of
```

```
btPlay:Label1.Caption := 'Çalıyor';
```

```
btPause:Label1.Caption := 'Bekliyor';
```

```
btStop: Label1.Caption := 'Durduruldu';
```

```
btNext:Label1.Caption := 'Sonraki';
```

```
btPrev:Label1.Caption := 'Önceki';
```

```
btEject:Label1.Caption := 'Çıkarıldı';
```

```
end;
```

end;

OnPostClick (Sender: TObject: Button: TMPBtnType):

OnPostClick olayı, OnClick olayı içindeki kodun işletilmesinden sonra meydana gelir.

OnClick olayı bitene kadar düğme çalışmaya başlamaz. OnClick olayındaki kod bittikten sonra düğme görevini yerine getirir ve OnPostClick olayı meydana gelir. Eğer OnClick olayında düğmenin görevi **DoDefault := False** verilerek iptal edilirse OnPostClick olayı da meydana gelmez.

OnNotify (Sender: TObject):

OnNotify olayı, media kontrol elemanının metotlarının (Back, Close, Eject, Next, Open, Pause, PauseOnly, Play, Previous, Resume, Rewind, StartRecording, Step, Stop) işletilmesinden sonra, Notify özelliği True ise meydana gelir. Bu olayda verilen komutun başarılıp başarısız olduğu **NotifyValue** özelliği ile değerlendirilir.

procedure TForm1.MediaPlayer1Notify (Sender: TObject);

var

s: string;

begin

case MediaPlayer1.NotifyValue of

nvSuccessful : s:= 'Komut yerine başarı ile yerine getirildi';

nvSuperseded : s:= 'Diğer komut işlenmeden yeni bir komut verildi';

nvAborted : s:= 'Komut iptal edildi';

nvFailure : s:= 'Komut hatalı';

end;

ShowMessage (s);

end;

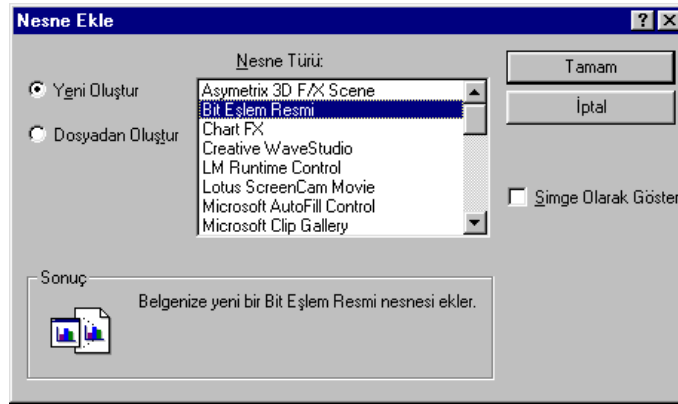
10.3.4. OLE (Nesne Bağlama ve Gömme Kontrolü)

OLE (Object Linking and Embedding) uygulamalar arasında veri paylaşımını sağlayan bir metottur. Bu metotla veri transferi kolay bir şekilde yapılabilir. Veri

transferinin yapılabilmesi için veriye kaynaklık edecek bir uygulamaya ve o kaynağa bağlantıyı kuracak olan **OleContainer** nesnesini içeren diğer bir uygulamaya ihtiyaç vardır. Veri kaynağını oluşturan program **OLE Server** adını alırken, ole nesnesini içeren program **OLE Container** adını alır. OLE metodu ile bağlantı kurulan uygulama üzerinde bilgi girişi ve bilgi düzenlemesi yapmak mümkün olmaktadır. Ole nesnesini içeren program ile Oleye kaynaklık edecek program arasında bir kere bağlantı kurulduktan sonra uygulamalar arası geçişler oldukça hızlı bir şekilde gerçekleştirilmektedir. OLE metodu ile ses, resim, doküman, grafik, elektronik tablolar vb. dosyalar kolay bir şekilde paylaşılır.

OLE kaynağı tarafından oluşturulan bir nesne, OLE nesnesini içeren program vasıtasıyla aktif hale getirildikten sonra ole kaynağı ile ole nesnesi arasında bağlantı kurulur. Bu bağlantı esnasında kaynağın menü çubuğu hedefin menü çubuğunda yer alır. Aynı zamanda kaynağın speed düğmeleri hedefin speed çubuğunda yer alır. Tasarım zamanında OLE nesnesi aracılığıyla veri paylaşımını şöyle yapabiliriz:

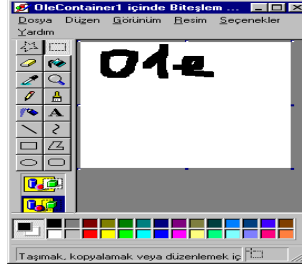
- OleContainer nesnesi form üzerine alınır.
- OleContainer1 çift tıklanarak OLE'yi destekleyen programların bulunduğu pencereye gidilir:



Şekil 10.23. Nene ekleme penceresi

- Burada istenilen nesne seçilir. Daha sonra **Tamam** düğmesi tıklanır.

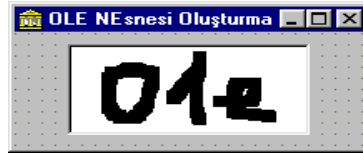
Böylece OLE'ye kaynaklık edecek olan program aktif hale gelir. Yukarıda görüldüğü gibi "Paintbrush Resmi" seçildiği için Paint programı aşağıda görüldüğü gibi aktif hale gelecektir.



Şekil 10.24. Açılan Paint programı

Buraya istenilen resim çizilir.

- Kaynak programdan bilgi girişi tamamlandıktan sonra Forma dönmek için kaynak programın Dosya menüsünden Çık ve xxx uygulamasına geri dön seçeneği kullanılır. Böylece ole nesnesi oluşturulmuş olur.



Şekil 9.25. Oluşturulan OLE nesneli form

Properties

AutoActivate:

AutoActivate özelliği bir OLE containerin ne şekilde aktif hale geleceğini belirler. AutoActivate özelliğinin aldığı değerler aşağıda gösterilmiştir.

aaManual: Kullanıcı Ole nesnesini direk aktif hale getiremez. OLE nesnesi DoVerb(ovShow) koduyla aktif hale getirilebilir.

aaGetFocus: OLE nesnesine focus kontrolü geçmesi halinde aktif hale gelir. Eğer form üzerinde ole nesnesinden başka focus kontrolünü alacak eleman yoksa focus kontrolü sürekli ole nesnesinde olacağından dolayı OLE nesnesi focus alamayacağı için aktif hale gelemes.

aaDoubleClick: OLE nesnesinin çift tıklanması halinde aktif hale gelir.

AllowInPlace:

AllowInPlace özelliği bir Ole nesnesinin bulunduğu yerden aktif olup olmayacağını belirler. Eğer true değerini alırsa aktif olur. false değeri verilirse Ole nesnesi kendi penceresinde açılır, yani Ole 1.0 gibi davranır. Ole 2.0'da ise Ole kaynağı kendi penceresinde değil çağırılan uygulamanın penceresi içinde çalışabilir.

Modified:

Modified özelliği ile Ole nesnesinin, OleContainer'ın başlatıldığı andan itibaren değişip değişmediği belirlenir. Eğer bu özellik true değerini alırsa nesne değişmiştir. False değerini almışsa nesne değişmemiştir.

Methods**InsertObjectDialog:**

Bu metod ole ekleme diyalog kutusunu görüntüler. Bu kutu görüntülendikten sonra kullanıcının herhangi bir nesneyi seçerek OK düğmesini tıklaması halinde InsertObjectDialog metodu true değeri gönderir. Cancel düğmesinin seçilmesi halinde ise false değeri gönderir.

Copy:

Copy metodu OLE Container içindeki OLE nesnesini clipboard'a kopyalar. Eğer Ole kontrolü içersinde bir OLE nesnesi yoksa hata oluşur.

OleContainer1.Copy;

Paste:

Panodaki bilgiyi bir OLE nesnesi olarak kontrolün içeriğine yapıştırır. Bu işlem için panodaki bilgiye kaynaklık eden uygulamanın OLE'yi destekliyor olması gerekir. Örneğin Word'de yazılıp panoya kopyalanmış bir metin;

OleContainer1.Paste;

Satırı ile Ole kontrolünün içersine alınırsa panodaki bilgi Word'de nasıl biçimlenmişse aynı şekilde Ole kontrolü içinde görüntülenecek ve çift tıklandığında Word menüleri programınızın penceresinde yerini alarak yerinde düzenlemeye müsaade edecektir.

Form üzerine bir OleContainer ve bir komut düğmesi yerleştirerek komut düğmesine aşağıdaki kodu yazalım.

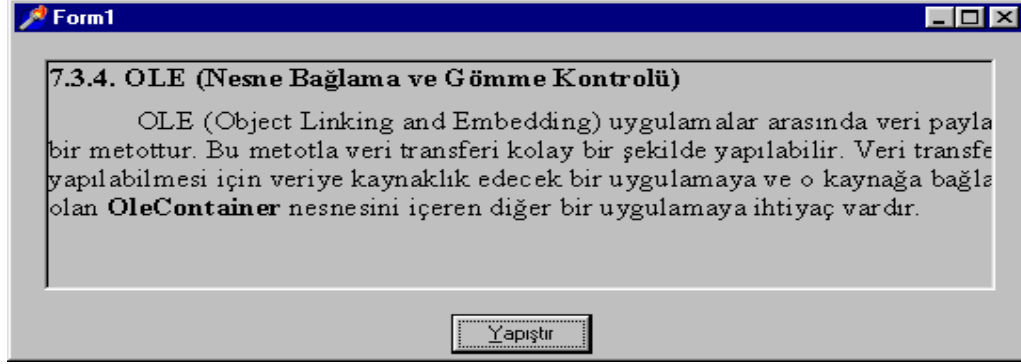
Procedure TForm1.Button1Click (Sender: TObject);

Begin

OleContainer1.Paste;

End;

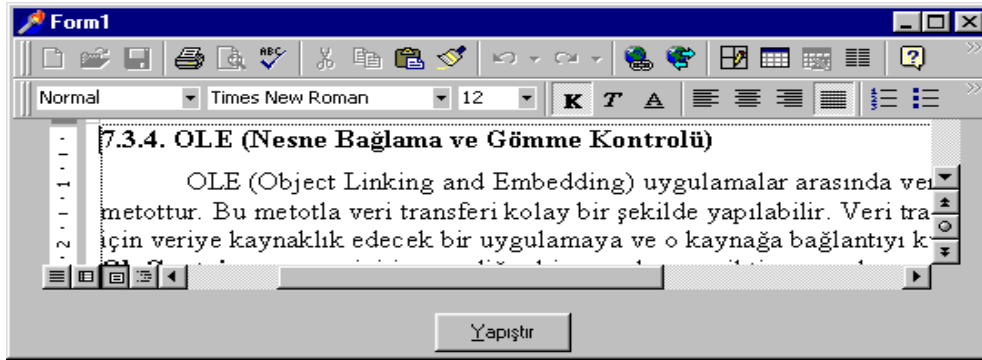
Word'den veya OLE'yi destekleyen başka bir programdan bir miktar bilgiyi panoya kopyaladıktan sonra programımızdaki komut düğmesine basalım. Panoya kopyaladığımız bilgi aynı formatta formumuzdaki Ole kontrolü içerisinde görülecektir.



Şekil 10.26. OLE kontrolüne yapıştırılan belge

Yukarıdaki formda Ole nesnesini düzenlemek için çift tıklarsanız Word'ün menüleri ve araç çubukları da formunuzda yerini alır (Şekil 10.32).

Şekil 10.32'de görüldüğü gibi Word'ün araç çubukları forma gelmesine rağmen menüleri gelmemiştir. Bunun nedeni hazırladığımız formda bir menünün bulunmamasıdır. Menülerin de formumuza gelebilmesi için tasarladığımız formun menü çubuğunun bulunması gerekir. Bunun için formumuza bir MainMenu kontrolü yerleştirerek programı çalıştırmamız yeterli olacaktır.



Şekil 10.27. Word'ün araç çubukları yerleştirilmiş form penceresi

10.4. Diyalog Kutuları



Şekil 10.28. Diyalog Kutuları Paleti



TOpenDialog: Windows'un sağladığı standart **Aç** diyalog penceresi.



TsaveDialog: Windows'un sağladığı standart **Kaydet** diyalog penceresi.



TOpenPictureDialog: Aslında OpenDialog gibidir ancak özel olarak bmp, ico, emf, wmf resimleri kullanıma açmadan önce ön izleme sağlar.



TsavePictureDialog: Resimleri kaydetmek için kullanılan pencere.



TfindDialog: Bul diyalog penceresi.



TReplaceDialog: Değiştir diyalog penceresi.



TfontDialog: Windows'ta yüklü fontları gösteren ve font seçimini sağlayan Font diyalog penceresi.



TColorDialog: Renk seçimini sağlayan diyalog penceresi.



TPrintDialog: Yazdır diyalog penceresi.



TPrinterSetupDialog: Yazıcı ayarları diyalog penceresi.

Diyalog kutuları Windows tarafından sağlanan standart arabirimler oluşturmak için kullanılan kutulardır. Bu kutuların tamamı **Execute** metoduyla aktif hale gelirler. Bu kutular tasarım zamanında form üzerinde görülmesine rağmen çalışma zamanında görülmezler.

Windows bir çok işlem için standart arabirimler sunar. Zaten Windows'un kullanımının kolay olmasının en büyük nedenlerinden biri de bu özelliğidir. Her

program aynı tip diyalog kutularını kullandığı için kullanıcı bu pencereleri hiçbir zorluk çekmeden kullanabilmektedir. Ayrıca standart diyalog pencerelerinin kullanımı programın uluslar arası kullanımını da kolaylaştırmaktadır. Çünkü bu diyalog kutuları Türkçe Windows kullanan bir yerde Türkçe olarak, İngilizce kullanan bir yerde İngilizce olarak diğer dilleri kullanan bir yerde ise o dilde gösterilecektir. Bu diyalog pencereleri sadece o işi yapmak için bir arabirim sağlarlar o işi yapacak kodu yazmak programcının işidir. Örneğin Bul diyalog kutusu aranan şeyi bulmaz onu bulduracak kodu yazmak programcının işidir.

10.4.1. TOpenDialog (Dosya Aç Penceresi)

Bu kontrolün temel gayesi herhangi bir dosyayı **Aç** diyalog kutusuyla kullanıma sokmaktır. **Open** diyalog kutusu Execute metoduyla aktif hale getirilir. Kullanıcının bu diyalog penceresinde seçtiği dosya ismi, OpenFileDialog kutusunun FileName özelliğiyle öğrenilir. Bu kutunun **Filter** özelliği kullanılarak kutuda görülecek dosya tipleri seçilebilir.

Properties

FileName:

Bu özellik openFileDialog kutusundaki dosya adının girildiği bölümde görülecek olan dosya adını belirler. Aynı zamanda kullanıcının seçtiği dosya adı da bu özellik ile öğrenilir. Örnek olarak kullanıcının seçtiği dosyayı bir **Memo** içinde gösterecek bir program yapalım.

// Örnek: Filename

procedure TForm1.Button1Click (Sender: TObject);

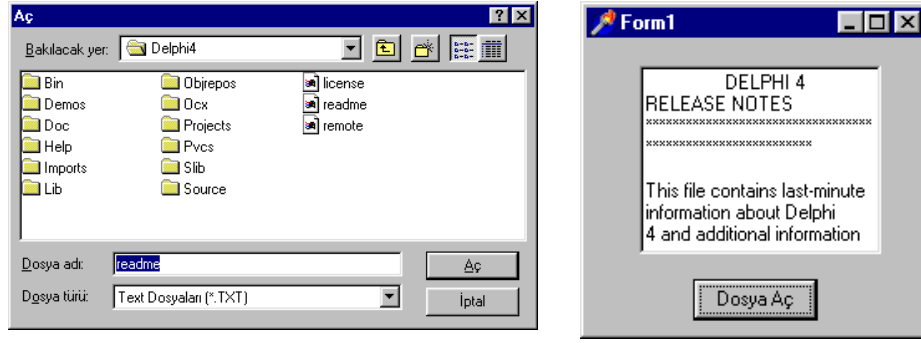
begin

OpenDialog1.Filter := 'Text Dosyaları (.TXT) | *.TXT| Bütün Dosyalar (*.*)
| *.*';*

if OpenDialog1.Execute Then

Memo1.Lines.LoadFromFile (OpenDialog1.FileName)

end;



Şekil 10.29. Örnek programın ekran çıktısı

Files:

Options özelliğinin bir alt özelliği olan ofAllowMultiSelect **True** yapılmışsa kullanıcı birden fazla dosyayı seçebilir. Bu özellik seçili olan dosyaları öğrenmek için kullanılır. Files özelliği daha önce gördüğümüz Items, Lines özellikleri gibi Tstrings tipinden tanımlanmıştır ve onlara uygulanan işlemler bu özelliğe de uygulanabilir.

Örneğin seçilen dosyaları bir liste içinde göstermek için aşağıdaki gibi bir program yazmamız gerekir.

With OpenFileDialog do

Begin

Options := [ofAllowMultiSelect];

*Filter := 'Pas Dosyaları (PAS) | *.Pas';*

Execute;

End;

ListBox1.Items := OpenFileDialog.Files;

Filter:

Filter özelliği OpenFileDialog penceresinde gösterilecek dosya türlerini belirtir. Bir kaç dosya türü belirtilirken “|” karakteri kullanılır.

*OpenDialog1.Filter := 'Metin Dosyaları | *.TXT';*

ifadesindeki “Metin Dosyaları” bölümü **dosya tipi** penceresinde, “| *.TXT” bölümü ise **dosya adı** penceresinde gözüktür.

OpenDialog1.Filter := 'Text Dosyaları | *.TXT | Pascal Dosyaları | *.PAS | Tüm Dosyalar | *.*';

FilterIndex:

FilterIndex özelliği Filter özelliği ile belirlenen türdeki dosyaların hangisinin ilk etapta gözükeceğini belirler. Varsayılan değeri “ 1 ” dir.

Filter özelliği,

OpenDialog1.Filter := ‘Text Dosyaları (*.TXT) |*.TXT | Pascal Dosyaları (*.PAS) | *.PAS | Tüm Dosyalar (*.*) | *.* ’;

olan dosyaların FilterIndex özelliği 2 verilirse, Pascal dosyalarının varsayılan değer olarak kutuda gözükeceğini belirtir.

DefaultExt:

Kullanıcı, diyalog kutusunda bir dosya ismi yazdıktan sonra uzantı yazmazsa bu özellik ile belirlenen uzantı otomatik olarak o dosyanın adına eklenir. Örneğin yaptığınız program BMP uzantılı dosyalarla çalışıyorsa bu özelliğe “BMP” değerini vererek kullanıcının uzantı belirtmediği durumlarda otomatik olarak BMP uzantısının eklenmesini sağlayabilirsiniz.

FileEditStyle:

Bu özellik, OpenDialog kutusu açıldığı zaman, dosya adının girileceği kısmın biçimini belirler. fsEdit, fsComboBox değerlerinden biri verilerek giriş kutusu şeklinde mi, combobox şeklinde mi olacağı belirlenir.

Methods

Execute:

Execute metodu ile diyalog kutusu aktif hale gelir. Diyalog kutularındaki Tamam düğmesi seçilirse True döner İptal düğmesi seçilirse False değeri döner. Bu yüzden diyalog kutuları aşağıdaki şekilde kullanılır.

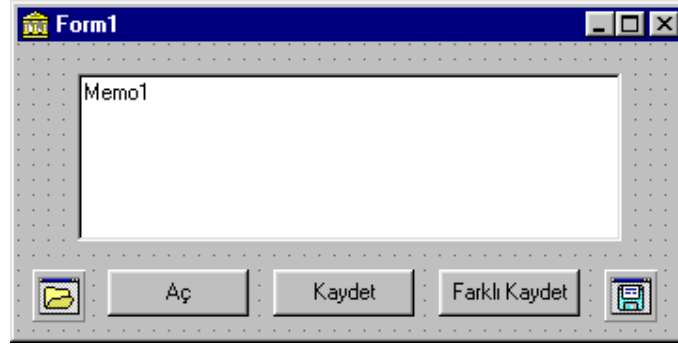
```
If OpenDialog1.Execute Then
// Tamam düğmesi seçildi. Gerekli kod
Else
// İptal düğmesi seçildi. Gerekli kod
```

10.4.2. TSaveDialog (Dosya Kaydet Penceresi)

Bu kontrol elemanı bir dosyanın kaydetme işleminde yardımcı olacak standart pencereyi sunar. Ayrıca OpenDialog penceresinin taşıdığı tüm özelliklere sahiptir. Bu da

Execute metoduyla aktif hale gelir. Herhangi bir dosya ismine otomatik uzantı eklenmek isteniyorsa DefaultExt özelliği kullanılmalıdır.

Örnek olarak bir Memo kontrolü içerisine dosyadan yükleme yapacak ve yapılan değişiklikleri aynı veya farklı bir isimle kaydedecek bir program yapalım. Örneğimiz için aşağıdaki formu oluşturalım.



Şekil 10.30. Hazırlanacak form

```
procedure TForm1.FormCreate (Sender: TObject);
begin
    Caption := 'Adsız.TXT'; // Dosya adı Caption özelliğinde tutuluyor
    // Uzantı belirlenmezse TXT kabul edilecek
    OpenFileDialog.DefaultExt := 'TXT';
    SaveDialog1.DefaultExt := 'TXT';
end;

procedure TForm1.Button1Click (Sender: TObject);
begin
    if OpenFileDialog.Execute Then
    begin
        Memo1.Lines.LoadFromFile (OpenDialog1.FileName);
        Caption := OpenFileDialog.FileName;
    end;
end;

procedure TForm1.Button2Click (Sender: TObject);
begin
    // Eğer bir isim verilmemişse farklı kaydet düğmesini aktif hale getir
    if Caption = 'Adsız.TXT' Then
```

```

    Button3Click (Self)
    Else
    Memo1.Lines.SaveToFile(Caption);
end;
procedure TForm1.Button3Click (Sender: TObject);
begin
    SaveDialog1.FileName := Caption;
    if SaveDialog1.Execute Then
    Begin
        Memo1.Lines.SaveToFile (SaveDialog1.FileName);
        Caption := SaveDialog1.FileName;
    End;
end;

```

10.5. Win32 Kontrollerleri



Şekil 10.31. Win32 Kontrolleri paleti

Delphi 4, Windows95'in getirdiği yeni kontrolleri programcılarının kullanımına sunmaktadır. Bu kontroller şunlardır:



TTabControl: Aynı kontrolleri barındıran tablalar oluşturmak için kullanılır.



TPageControl: Farklı kontrolleri barındıran tablalar oluşturmak için kullanılır.



TTreeView: ağaç yapısında listeler oluşturmak için kullanılır.



TListView: Resimli listeler oluşturmak ve bunları değişik şekillerde gösterebilmek için kullanılan bir liste kontrolüdür.



TImageList: birden fazla resmi bir arada tutmak için kullanılan bir kontrolüdür. TTreeView ve TListView kontrollerindeki resimlerde ancak bu kontrolle belirlenebilmektedir.



THeaderControl: Daha çok birden fazla listeye başlık oluşturmak için kullanılan kontrolüdür.



TRichEdit: İçersine aynı anda değişik font ve tipte yazı yazılabilen ve RTF

formatını destekleyen gelişmiş bir metin kutusudur.



TUpDown: Bir değeri arttırıp azaltmak için kullanılan bir kontroldür.



TStatusBar (Durum Çubuğu): Daha çok programların en alt kısmında bulunan ve o anki durumla ilgili bilgilerin yazıldığı panelleri oluşturmak için kullanılır. Tek kontrolle birden fazla panel oluşturulabilir.



TprogresBar: İşlemlerin ilerleme durumunu göstermek için kullanılır.



TTrackBar: bu kontrol de bir değeri arttırıp azaltmak için kullanılabileceği gibi kaydırma çubuğu gibi de işlev görür.



THotKey: Kısa yol tuşlarını kullanıcının girebilmesini sağlayan kontroldür.



TAnimate: Win95 ile birlikte gelen işlem animasyonlarının tamamını göstermekle birlikte ayrıca avi formatındaki diğer dosyaları da form üzerinde oynatmak mümkündür.



TDateTime: Saati ve tarihi seçebilmek için güzel bir arabirim sunar. Bir edit kutusu görünümündedir ancak kullanıcı bir tarih seçmek istediğinde otomatik olarak açılan takvimi kullanabilmektedir.



TToolBar: Araç çubuklarını ve üzerindeki düğmeleri kolayca oluşturup yönetebileceğiniz bir kontroldür.



TCoolBar: Araç çubuklarının yer aldığı bantları oluşturur. Bununla araç çubuklarının konumlarını değiştirmek, boyutlandırmak, yan yana yada alt alta görüntülemek mümkün olmaktadır.

10.5.1. TPageControl

Birden fazla sayfaya sahip diyalog kutusu oluşturmak için düşünülmüş bir elemandır. Bu kontrolün en önemli özelliği her sayfaya farklı kontrollerin yerleştirilebilmesidir.

Daha çok diyalog pencerelerinde kullanılan bu kontrol çok sayıda kontrolün form üzerine rahatça yerleştirilebilmesini sağlar. Windows altında bu kontrolün kullanıldığı örnekleri bir çok programda görebilirsiniz. Örneğin Word'ün aşağıdaki penceresi (Şekil 10.37) bu kontrolü kullanmaktadır.

Tasarım zamanında elemana yeni bir sayfa ekleme işi sağ fare tuşu aracılığıyla olmaktadır. Eleman form üzerine alındıktan sonra, eleman seçili iken sağ fare tuşuna basıldığı zaman popup olarak açılan menüde New Page, Next Page, Previous Page menü seçenekleri gözükür. Burada New Page ile PageControl elemanına istenilen sayıda tab eklenir. Next Page ile sonraki taba geçiş yapılır. Previous Page ile de önceki taba geçiş yapılır. Eklediğiniz her sayfaya farklı kontroller yerleştirebilirsiniz.



Şekil 10.32. Word'ün sayfa yapısı penceresi

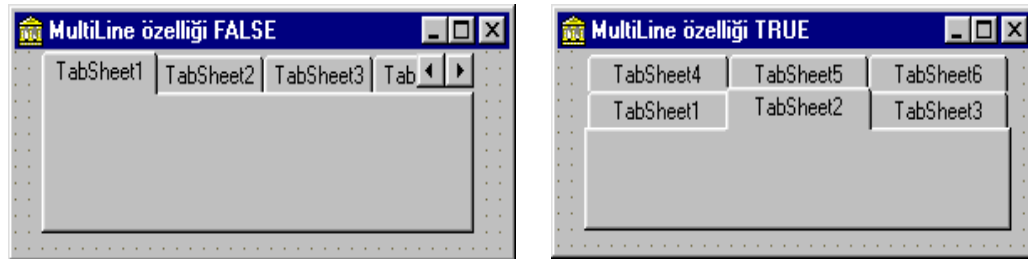
Properties

ActivePage:

PageControl üzerindeki aktif tabı öğrenmek ve değiştirmek için bu özellik kullanılır.

MultiLine:

Bu özelliğin değeri false ise ve tablar kontrolün içersine sığmıyorsa tab kontrolüne kaydırma çubukları eklenir ve tabların tek sıra halinde bulunması sağlanır. Bu özelliğin değeri true ise ve tablar kontrolün içersine sığmıyorsa, tablar alt alta gelecek şekilde yerleştirilir.



Şekil 10.33. MultiLine özelliği False ve True olan tab kontrolleri

Events

OnChange (Sender: TObject; var AllowChange: Boolean):

Bu olay tab değişken yeni taba geçmeden önce meydana gelir. OnChanging olayı henüz aktif tab kontrolü kaybetmediği için diğer taba geçilmeden yapılacak işlemler burada yapılır. Yeni taba geçtikten sonra yapılacak ilk işlemler ise **Change** olayında yapılır. Belirli şartlar gerçekleşmedi ise, örneğim yanlış bir değer girilmişse geçiş işlemini iptal etmek için **AllowChange := false;** yapılır.

Properties**Multiline:**

Bu özelliğin true olması halinde birden fazla satıra sahip tab oluşturmak mümkün olmaktadır.

TabIndex:

Seçili olan tabın sıra numarasını belirler. İlk tabın sıra numarası 0 iken bir sonraki tabın sıra numarası 1 dir. Bu sıralama böylece devam eder.

Events**OnChange (Sender: TObject):**

Bu olay yeni bir tabın seçilmesi esnasında meydana gelir. Aşağıdaki program satırlarını yazarak bu tabın çalışmasını tespit edebiliriz.

Procedure TForm1.TabControl1Change (Sender: TObject);

Begin

ShowMessage(inttostr(TabControl1.tabindex) + 'nolu tab seçildi.');

End;

OnChange(Sender: TObject; var AllowChange: Boolean):

Bu olay tab değişken yeni taba geçmeden önce meydana gelir. Change olayı değiştikten sonra meydana gelir. OnChanging olayı, henüz aktif tab kontrolü kaybetmediği için diğer taba geçilmeden yapılacak işlemler burada yapılır. Yeni taba geçtikten sonra yapılacak ilk işlemler ise Change olayında yapılır. Belirli şartlar gerçekleşmedi ise örneğin yanlış bir değer girilmişse geçiş işlemini iptal etmek için **AllowChange:=False** yapılır.

10.5.2. TUpDown

Daha çok bir değeri arttırıp azaltmak için kullanılan bir kontroldür.

Properties

Min, Max:

Bu özellikler updown elemanın arttırma ve azaltma işlemlerinde varabileceği alt ve üst değeri belirlerler.

Orientation:

Bu özellik updown elemanını yatay yada dikey olarak ayarlar. UdVertical dikey, udHorizontal yatay hizalamayı sağlar.

Position:

Bu özellik updown elemanının değerini temsil eder. Bu değer min ve max değerleri arasında olabilir. Aşağıdaki örnekte updown kontrolünün iki farklı kullanım alanı görülmektedir. Updown1 kontrolü ile Edit1 kontrolünün font büyüklüğü değiştirilirken, Updown2 kontrolü ile de Edit2 kontrolünün içindeki sayı değiştirilmektedir.

// Örnek: UpDown

```
procedure TForm1.UpDown1Changing (Sender: TObject; var AllowChange: Boolean);  
begin
```

```
    Edit1.Font.Size := UpDown1.Position;
```

```
end;
```

```
procedure TForm1.UpDown2Changing (Sender: TObject; var AllowChange: Boolean);  
begin
```

```
    Edit2.Text := IntToStr (UpDown2.Position);
```

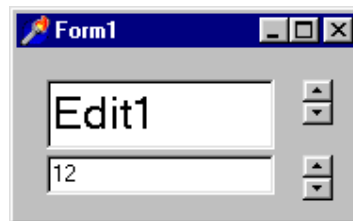
```
End;
```

```
procedure TForm1.Edit2Change (Sender: TObject);
```

```
begin
```

```
    UpDown2.Position := StrToInt (Edit2.Text);
```

```
end;
```



Şekil 10.34. Programın çıktısı

10.5.3. TdateTimePicker:

DateTimePicker kontrolü tarih ve saatin bulunduğu hazır bir kutu sunar.

Properties

Kind:

Kontrolün tarihimi yoksa saatimi göstereceğini belirler. Bu özelliğe dtkDate değeri verilirse tarihi, dtkTime verilirse saati gösterir.

DateMode:

Eğer kind özelliğine dtkDate değeri verilerek kontrolün tarihi göstermesi sağlanmışsa bu özellik ile kontrolün görünümü belirlenebilir. Bu özelliğe dmComboBox değeri verilirse kontrol bir combobox olarak görülür ve liste aşağı doğru açıldığında aylık takvim gösterilir.

MinDate, MaxDate:

Kullanıcının girebileceği tarih aralığı bu iki özellik ile belirlenebilir. Eğer kullanıcı belirlenen bu aralık dışında bir tarih seçerse bir hata mesajıyla bildirilir.



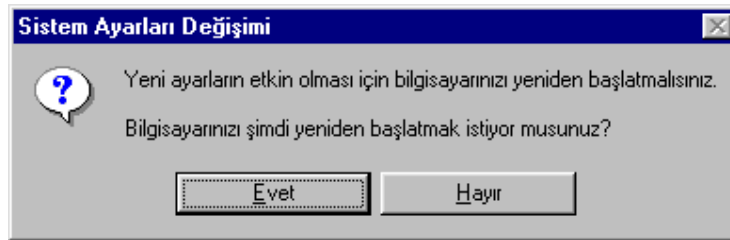
Şekil 10.35. Tarih ve Zaman kontrolleri yerleştirilmiş form

BÖLÜM 11

API FONKSİYONLARI

11.1. API'nin Tanımı

API'ler hakkında bilgi vermeye örnek bir olayı anlatarak başlayacağım. Windows'un çalışması üzerinde etkili olan bir işlem veya ayarlama yapıldığı zaman yapılan ayarlamanın etkili olması için Windows'un yeniden başlatılması gerekir. Örneğin bilgisayara yüklü olan fontlarla işlem yapıldığı zaman Windows ekrana **Sistem Ayarları Değişimi** diyalog kutusunu getirir ve yapılan ayarlamanın etkili olabilmesi için Windows'u (95 veya 98) yeniden başlatmanız konusunda sizden onay alır.



Şekil 11.1. Sistem Ayarları Değişimi adlı diyalog kutusu

Sistem Ayarları Değişimi başlıklı bu diyalog kutusundaki **Evet** düğmesine tıklama yaparsanız Windows yeniden başlatılır. **Hayır** düğmesine tıklama yaparsanız yapılan ayarlama daha sonra etkili olur. Eğer Delphi projesi dahilinde Windows'u yeniden başlatmak istiyorsanız bu amaçla kullanabileceğiniz herhangi bir Delphi fonksiyonu veya deyimi yoktur. Bu gibi durumlarda API olarak adlandırılan Windows'un dahili fonksiyonlarına başvurabilirsiniz.

Windows'u yeniden başlatmak için **ExitWindowsEx()** fonksiyonundan yararlanılmaktadır. Bu fonksiyondan yararlanılarak Windows'un nasıl tekrar başlatıldığını size anlatmak için önce yeni bir proje hazırladım ve projenin formuna "Windows Oturumunu Kapat" başlıklı bir düğme yerleştirerek Clik yordamına aşağıdaki kodları yazdım.

Procedure TForm1.KapatClick (Sender: TObject);

Var

Tus : Integer;

Begin

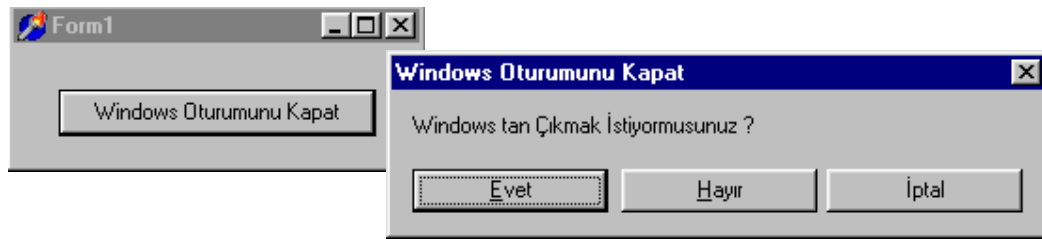
*Tus := Application.MessageBox ('Windows tan Çıkmak İstiyor musunuz ?',
'Windows Oturumunu Kapat', mb_YesNoCancel + mb_DefButton1);*

If Tus = 6 Then

ExitWindowsEx (42,1);

End;

Forma yerleřtirmiş olduđum düğmenin Click yordamında kullandığım ExitWindowsEx() fonksiyonu dışardan iki parametre almaktadır. Eğer ikinci parametre olarak “1” değeri yerine “2” değeri verilirse ekrana “Bilgisayarınızı Kapatabilirsiniz” mesajı getirilmez ve bilgisayar yeniden başlatılır. Aşağıda verdiđim ekran görüntüsünü hazırladıđım bu projeyi çalıştırıp “Windows Oturumunu Kapat” başlıklı düğmeye tıklama yaptıktan sonra aldım.



Şekil 11.2. Programın çalıştırılması

Bu sırada evet başlıklı düğmede tıklama yapılırsa MessageBox() fonksiyonu geriye 6 değerini gönderir ve ExitWindowsEx() fonksiyonu işletilerek o sırada çalışan bütün programlarla birlikte Windows’un çalışması sona erdirilir.

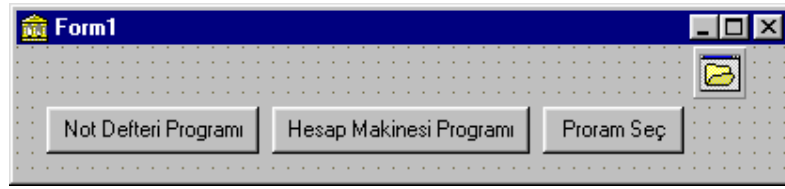
Anlatılan şekilde Windows’un API olarak adlandırılan fonksiyonların özelliklerini öğrenip Delphi projeleri dahilinde kullanabilirsiniz. Windows’un API fonksiyonları hakkında bilgi edinmek için Delphi ile birlikte verilen yardım metinlerinden yararlanabilirsiniz. API fonksiyonları hakkında bilgi içeren **Win32.hlp** adlı dosya Delphi’nin kurulu olduđu dizinin altında yer alan Help dizininde yer bulunmaktadır. Bu bölümde size yalnızca birkaç API fonksiyonu hakkında bilgi vereceğim.

11.2. Delphi Projesi Dahilinde Başka Programları Çalıştırmak

Bilgisayara kurulu olan herhangi bir programı çalıştırmak için Windows’un WinExec() adlı API fonksiyonunda yararlanabilirsiniz. WinExec() fonksiyonu dışarıdan parametre olarak iki bilgi almaktadır. İlk parametrede çalıştırılacak program dosyasının

adı verilmektedir. İkinci parametrede ise çalıştırılacak program dosyasına ait pencerenin ilk şekli belirlenmektedir.

Çalıştırılan program dosyasına ait ilk pencere ekrana getirildiği zaman ekranı kaplaması isteniyorsa WinExec() fonksiyonuna ikinci parametre değeri olarak 3 değeri verilmelidir. İkinci parametre olarak 6 değeri verilirse programa ait pencere simge durumuna küçültülür., 9 değeri verilirse pencere orijinal boyutları ile ekrana getirilir. WinExec() fonksiyonunun işlevini anlatmak için üzerinde çalıştığım projenin formuna üç düğme ve bir OpenFileDialog nesnesi yerleştirdim.



Şekil 11.3. Hazırlanan form

Çalışma anında Not Defteri Programı başlıklı düğmede tıklama yapıldığı zaman WinExec() fonksiyonu sayesinde Windows ile birlikte verilen not defteri programının çalıştırılmasını sağlamak için bu düğmeye ait Click yordamını aşağıdaki gibi düzenledim.

Procedure TForm1.Button1Click (Sender: TObject);

Begin

WinExec ('C:\Windows\notepad.exe', 9);

End;

WinExec() fonksiyonunun deklare edildiği satıra göre WinExec() fonksiyonu işletildiği zaman kendisine parametre olarak verilen program dosyası ile ilgili olarak geriye sayısal bir değer gönderilmektedir. Eğer çalıştırılmak istenen program dosyası hard diskte bulunamazsa geriye 2 değeri gönderilir. Eğer belirtilen sürücü veya klasör bulunamazsa WinExec() fonksiyonu bu kez geriye 3 değerini gönderir.

Bilgisayarda belirtilen programı çalıştırmak için yeterli bellek yoksa geriye 8 değeri döndürülür. Delphi projesi dahilinde çalıştırılacak program dosyasını çalışma anında belirleyebilmek için projenin formuna bir OpenFileDialog nesnesi yerleştirdim ve Program Seç başlıklı düğmenin click yordamına aşağıdaki kodları yazdım.

Procedure TForm1.Button3Click(Sender: TObject);

Var

```

Sonuc: Integer;
Begin
    OpenFileDialog.Execute;
    Sonuc := WinExec (PChar (OpenDialog1.FileName), 9);
End;

```

11.3. Windows'un Kurulu Olduğu Klasörü Öğrenmek

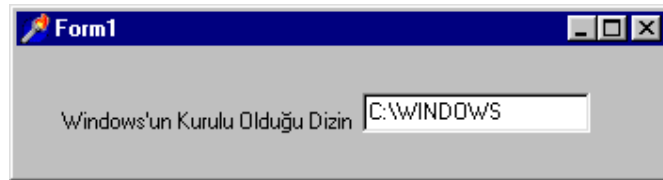
Windows'un kurulu olduğu klasörü öğrenmek istiyorsanız Windows'un **GetWindowsDirectory()** fonksiyonunu kullanabilirsiniz. Bu fonksiyon dışardan iki parametre almaktadır. Bu fonksiyon işletildiği zaman Windows'un kurulu dizinin adı, fonksiyona ilk parametre olarak verilen karaktersel değişkene parametre olarak aktarılmaktadır.

GetWindowsDirectory() fonksiyonu ilk parametre olarak verilen karaktersel tipteki değişkenin daha önce tanımlanması ve Windows'un kurulu olduğu dizinin uzunluğu kadar bilgi aktarılması gerekir. GetWindowsDirectory() fonksiyonuna ilk parametre olarak verilen karaktersel bilgi içeren değişkenin içerdiği bilginin uzunluğu fonksiyona ikinci parametre olarak verilir. Hazırladığım formun Create yordamına aşağıdaki kodları yazarak programı çalıştırdım.

```

Procedure TForm1.FormCreate(Sender: TObject);
var
    yol: Pchar;
begin
    yol := '-----';
    GetWindowsDirectory (Yol,Length(Yol));
    Edit1.Text :=Yol;
end;

```



Şekil 11.4. Programın çıktısı

Diğer yandan Windows ortamında çalışan programların gerek duyduğu DLL veya diğer dosyalarının bulunduğu, Windows'un kurulu dizini altında yer alan System

dizinin yolu öğrenilmek istenebilir. System dizinin yerini öğrenilmek için **GetSystemDirectory()** fonksiyonu kullanılmaktadır. Bu **GetWindowsDirectory()** fonksiyonu ile aynı özelliklere sahiptir. Yukarıdaki örnekte **GetWindowsDirectory()** fonksiyonu yerine **GetSystemDirectory()** fonksiyonu yazıp programı çalıştırsak sistem dizinin yolunu öğrenebiliriz.

BÖLÜM 12

DLL DOSYASI YAZMA

12.1. DLL Nedir?

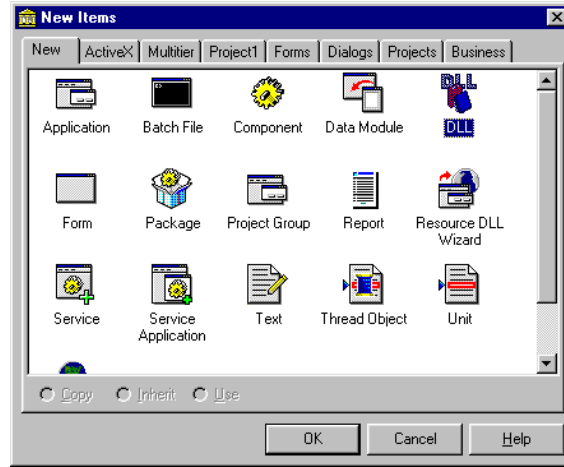
DLL dosyaları gerçekte birer program olmalarına rağmen kendi başlarına çalıştırılmaları bir anlam ifade etmez. Bunu denemek için bir DLL dosyasının uzantısını EXE olarak değiştirip çalıştırın. Dosyanın herhangi bir hata vermeden çalıştığını ancak sonuçta hiçbir şey olmadığını göreceksiniz. Evet DLL dosyaları gerçekte çalıştırılabilir dosyalardır ancak tek başlarına tek başlarına çalıştırılmak için değil, diğer programlar tarafından çalıştırılmaları için oluşturulurlar. Bu dosyalar içlerinde değişik işlemler yapan fonksiyon ve prosedürler içerirler. Bunun bir çok avantajı vardır.

Örneğin GIF formatındaki resimleri göstermek için bir fonksiyon yazdığınızı düşünelim. GIF formatında resimleri gösterecek bütün programlarımıza bu kodu yazmak yerine bunu bir Unit olarak kaydedip diğer programlarımızda da kullanabiliriz. Ancak başka bir programlama dili ile yazdığımız bir programda bu Unit'i doğal olarak kullanamayız. Bunu o dil için tekrar yazmamız gerekir.

Bu gibi durumlar için bu fonksiyonunuzu Windows tarafından standartlaştırılan formatta derlerseniz Windows altında çalışan diğer bütün programlar bu fonksiyonu kullanabilir. Başkalarının yazdığı fonksiyonları da nasıl yazıldığını bilmenize gerek kalmadan programlarınızda kullanabilirsiniz.

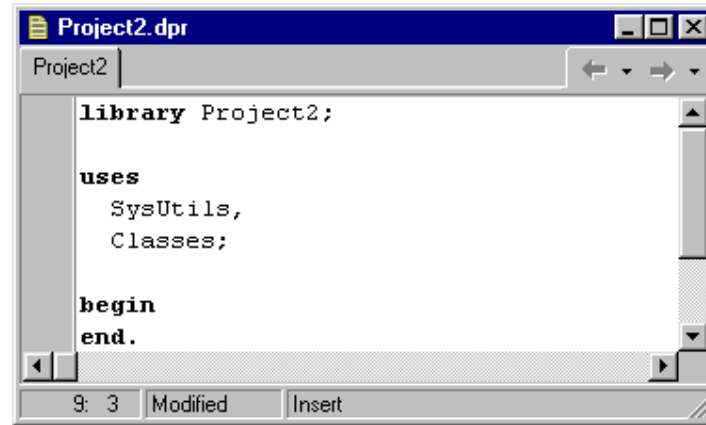
12.2. DLL Nasıl Yazılır?

DLL dosyaları içlerinde fonksiyon barındıran dosyalar olduğu için normal bir program gibi yazılmazlar. Delphi'de bir DLL yazmak için kolay bir yolunuz var. File menüsünden New seçeneğini seçerseniz aşağıdaki pencere gelecektir.



Şekil 12.1. New Items penceresi

Bu pencereden **DLL** (Eski versiyonlarda bu seçenek yerine **Library**) seçeneğini kullanarak Delphi'nin bir kütüphane için gerekli olan satırları oluşturmasını sağlayabiliriz. Bu işlem sonunda penceremizdeki kod aşağıdaki gibi olacaktır.



Şekil 12.2. DLL hazırlamak için oluşturulan projenin kod penceresi

İlk satırdaki **Library** kelimesi bunun bir kütüphane – dll programı olacağını gösterir.

- Yazacağımız fonksiyon, prosedür veya değişken tanımlarını **Begin** satırından önce yazmamız ve eğer bu fonksiyonlar dışarıdan çağırılabilirse fonksiyon tanımının sonunda **Export** deyimi ile bunu bildirmemiz gerekir.
- Yazdığımız fonksiyonların sonunda, Begin satırından önce dışarıdan çağırılabilir fonksiyonları **Exports** altında bildirmemiz gerekir.

- DLL dosyası F9 tuşu ile çalıştırılmaz. Yalnız derlemek için **Ctrl – F9** tuşunu veya Project – Compile seçeneğini kullanmamız gerekir. Bu işlem sonucunda program EXE olarak değil DLL uzantılı olarak derlenecektir.
- DLL dosyası tasarlanırken çalıştırıp sonucu test etmek aynı anda zor olduğu için fonksiyonları bir projede hazırlayıp en son haliyle bir library projesine kopyalamak daha kolay olacaktır.
- Herhangi bir DLL dosyasından bir fonksiyon çağırmadan önce onu aşağıdaki gibi tanımlamamız gerekir.

Function Ad(Parametreleri: Tipi): Tipi; far; external 'DLLADI' index no;

Bu tanımlı yaptıktan sonra normal fonksiyon çağırısı yaparak fonksiyonu kullanabiliriz.

- Yazdığımız DLL içindeki fonksiyonlardan biri geriye string tipinde bir değişken gönderiyorsa **GetMem** fonksiyonu ile o string için bellek ayırmanız gerekir.

Örnek DLL

Örnek olarak bir DLL dosyası yazalım. Bu DLL dosyası girilen rakamı yazıya çevirip geri göndersin. Yani rakam olarak “5” girerse yazı olarak “Beş” dönsün. Önce File – New menüsünden DLL seçeneğini kullanarak yeni bir DLL dosyası açalım ve bunu **dllSayiCevir** olarak kaydettikten sonra aşağıdaki kodları yazalım.

library dllsayicevir;

uses

SysUtils,

Classes;

function SayiCevir (X: integer): PChar; export;

var

Sayi: PChar;

begin

GetMem (Sayi,100); // Bellekte 100 bayt ayır

case x of

1:Sayi := 'Bir';

2:Sayi := 'İki';

3:Sayi := 'Üç';

4:Sayi := 'Dört';

```

5:Sayi := 'Beş';
6:Sayi := 'Altı';
7:Sayi := 'Yedi';
8:Sayi := 'Sekiz';
9:Sayi := 'Dokuz';
0:Sayi := 'Sıfır';
Else Sayi := 'Basamak Sayısı Fazla';
end;
SayiCevir := Sayi;
end;
Exports
    SayiCevir;

```

Bu kodu yazdıktan sonra Ctrl + F9 tuşlarına basarak projeyi derleyelim.

Şimdi bu dosyadaki DLL fonksiyonunu çağırarak programı yapalım. Yeni bir proje başlatarak form üzerine bir metin kutusu, bir label kontrolü ve bir de buton yerleştirelim. Yerleştirdiğimiz butonun Click yordamına aşağıdaki kodları yazalım. DLL dosyasında tanımlı olan fonksiyonları kullanabilmek **implementation** satırından sonraki komut cümlesini de yordamımıza ilave etmeliyiz.

implementation

```

Function SayiCevir (x: integer): PChar; far; external 'dllSayiCevir.dll';
procedure TForm1.Button1Click (Sender: TObject);
var
    sayi: integer;
begin

```

```

    sayi := StrToIntDef (Edit1.Text,0);

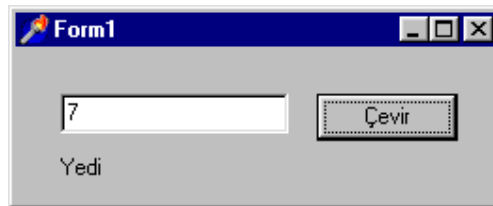
```

```

    Label1.Caption := String (SayiCevir(sayi));

```

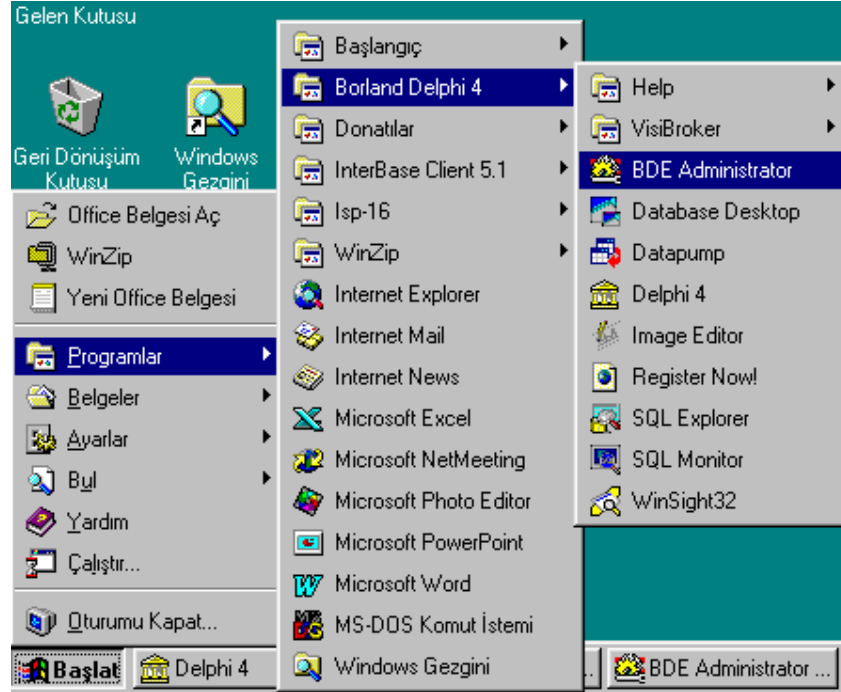
End; Yukarıdaki kodları yazdıktan sonra programımızı çalıştırsak aşağıdaki gibi bir sonuç elde ederiz.



BÖLÜM 13

ALIAS HAZIRLAMAK

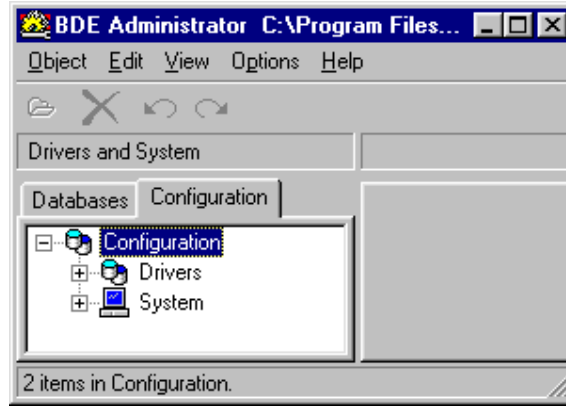
Delphi’ de projelerde kullanılan veri tabanı dosyaları gruplamak için kullanılan **Alias** adında yeni kavram veya özellikler bulunmaktadır. Delphi projeleri dahilinde çok sayıda veri tabanı dosyaları üzerinde işlem yaparken Delphi projeleri ile birlikte verilen **Borland Database Engine** programından yararlanılmaktadır. Delphi ile birlikte ayrıca Borland Database Engine programının çalışma şeklini ayarlama da kullanılan BDE Administrator adında yardımcı bir program verilmektedir.



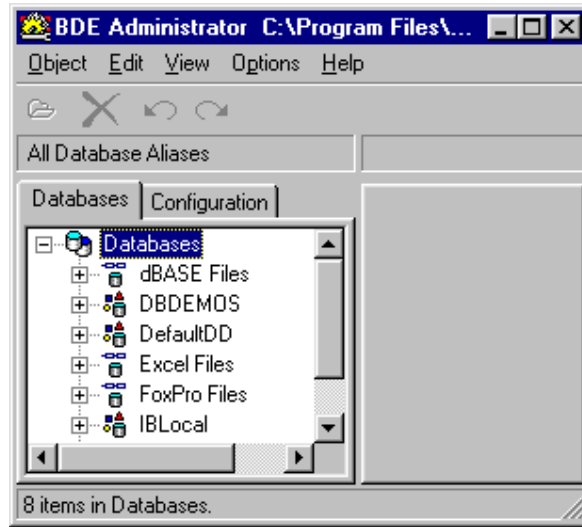
BDE Administrator programı ile yapılan seçim ve ayarlamalar “CFG” uzantılı bir dosyaya yazıldığı için BDE Administrator programı penceresinin başlığına ayrıca üzerinde işlem yapılan CFG uzantılı dosyanın adı eklenmektedir.

BDE Administrator penceresinde menu çubuğundaki menülerden başka Database ve Configuration adında iki sekme bulunmaktadır. Configuration sekmesinde

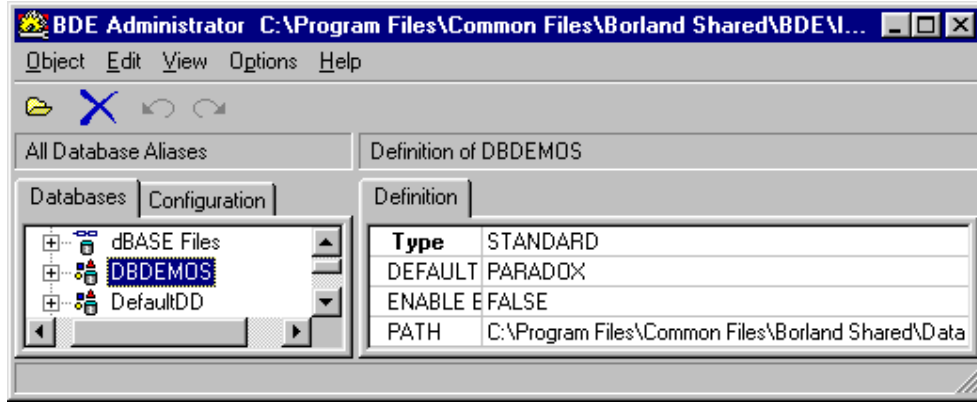
Drivers ve System adında iki seçenek bulunmaktadır. Drivers seçeneği ile sürücüsü yüklü olan veri tabanlarını görebilir ve sürücü dosyalarının özelliklerinde değişiklik yapabilirsiniz.



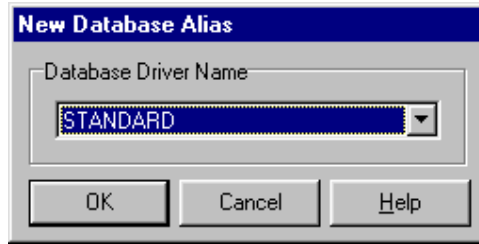
Yeni bir Alias hazırlamak veya daha önce hazırladığınız Aliasın özelliklerinde değişiklik yapmak istiyorsanız Database sekmesine gelmeniz gerekir. Seçildiğinde birçok Alias seçilmektedir.



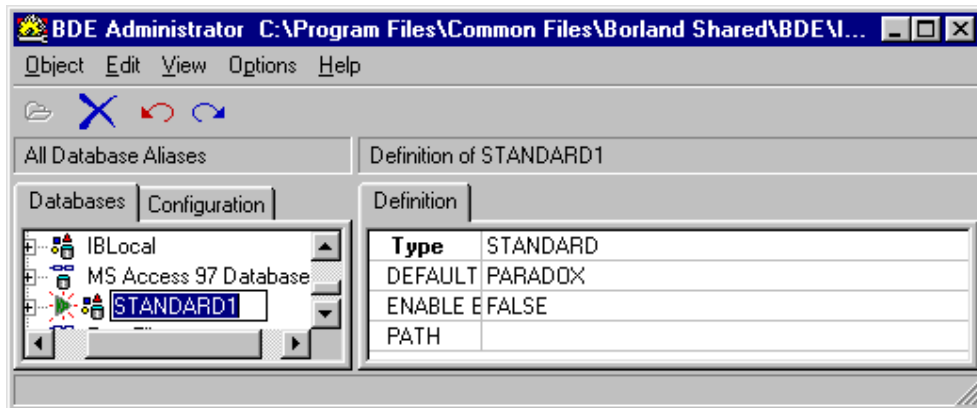
Mevcut Aliaslardan hangisi hakkında bilgi edinmek veya özelliklerinde değişiklik yapmak istiyorsanız o Aliası seçmeniz gerekir. Aşağıda verilen ekran görüntüsünü DBDEMOS adlı Aliası seçtikten sonra aldım.



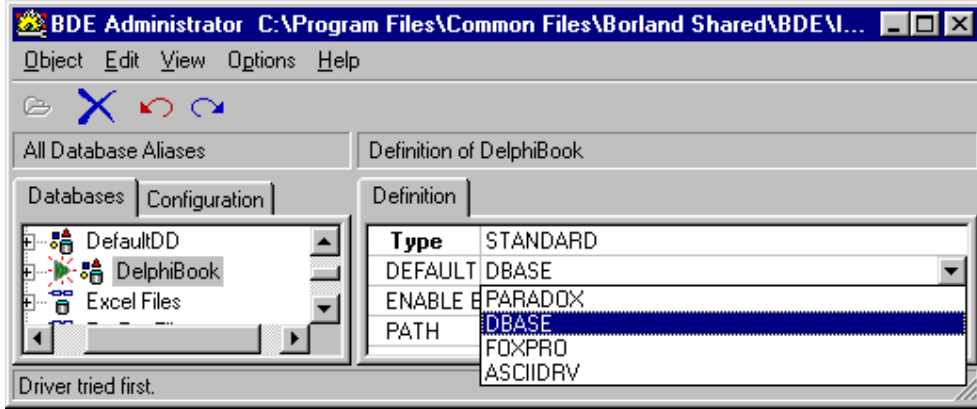
Yeni bir alias hazırlamak için Databases sekmesi seçili durumda iken Object menüsünden New komutu verilmelidir.



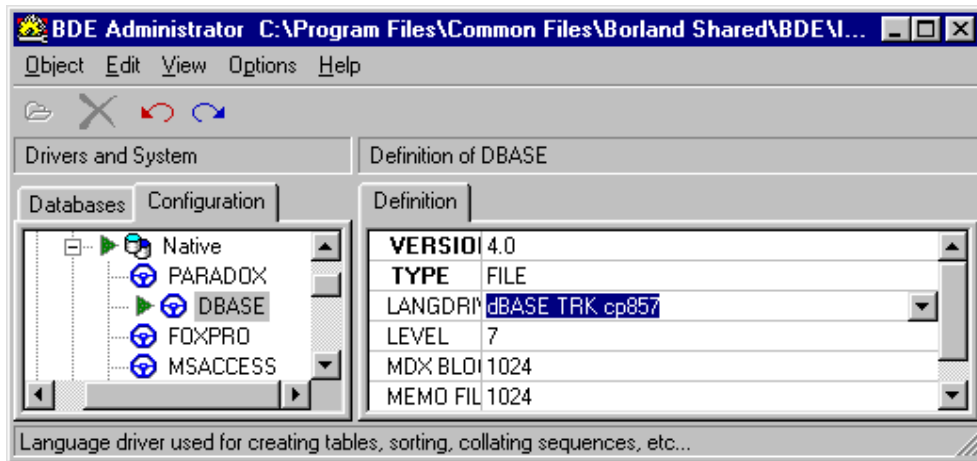
Bu liste kutusunu açıp Delphi projeleri dahilinde kullanılabilecek veri tabanı sürücülerini görebilirsiniz. Başlangıçta STANDART adlı veri tabanı sürücüsünü kullanmak istediğim için New Database Alias diyalog kutusunda herhangi bir işlem yapmadan OK düğmesine tıklama yaparak standart özelliklere sahip bir Aliasın hazırlanmasını sağladım.



Bu Aliasın adını DelphiBook olarak değiştirdim. Ancak bu yeni hazırladığımız Alias yardımı ile açılacak ve üzerinde işlem yapılacak veri tabanı dosyalarının hangi sürücü ve klasörde yer alacağı henüz belli değildir. Default Driver parametresinin içeriğini dBASE olarak değiştirelim.



Örnek olması için hazırlayacağım veri tabanı dosyalarını "C:\Ornekler" dizini içinde saklamayı düşündüm. BDE Administrator penceresinde hazırladığım yeni Aliasın Path parametresine "C:\Ornekler" yazdım. Bu Alias yardımı ile dBASE formatında hazırlayacağım veri tabanı dosyalarında kayıtların Türkçe'nin alfabetik sırasına göre sıralanmasını istediğim için hazırladığım Alias için dil seçimi yaptım. Bu amaçla Configuration sekmesine geçtim. Dbase sürücüsünü seçip LANGDRIVER seçeneği ile açıp dBASE sürücüsü için Türkçeyi seçtim.

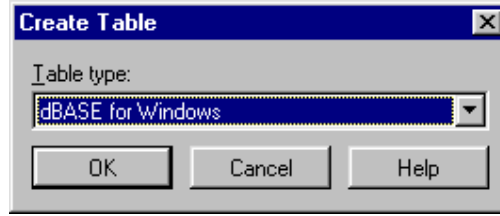


Bu ayarlamalardan sonra CFG uzantılı Configuration dosyasını hard diske kaydettim. Object menüsünden Configuration komutu verilir. Idapi32.cfg olarak kaydedilir.

BÖLÜM 14

VERİ TABANI DOSYASI HAZIRLAMAK

Delphi dahilinde dBASE, Paradox, InterBase formatında veri tabanı dosyası hazırlayıp üzerinde işlem yapabilirsiniz. Yeni bir veri tabanı dosyası hazırlamak için Delphi ile verilen DataBase Desktop programından yararlanılmaktadır. Tools menüsündeki Database Desktop komutu verilerek uygulama başlatılır. Yeni bir veri tabanı dosyası için Database Desktop penceresinden File menüsünden New komutu verilir. Alt menüden Table komutunu verecek olursanız dosyanın formatının seçildiği Table Type diyalog kutusu ekrana gelir. Bu diyalog kutusundan “dBASE for Windows” formatını seçelim.



OK tuşuna basıldığında veri tabanı dosyasının kayıt yapısını belirten diyalog kutusu ekrana gelir. Veri tabanı dosyalarının kayıt yapısı veya dosyada yer alacak alanlar bu şekilde belirlendikten sonra sıra kayıt arama işlemlerinde kullanılacak olan Indexlerin seçimine gelir. Hazırlanmak istene veritabanı dosyası için Index hazırlamak için bu diyalog kutusundaki Table Properties liste kutusunda Index seçeneği varken Define düğmesine tıklama yapılır. Ekrana gelen Define Index diyalog kutusunda bütün alanlar listelenir. Alan seçili duruma getirilir. Dosyada aynı soyada sahip olan birden fazla kaydın olmasına izin vermek istemiyorsanız Unique onay kutusunu, eğer NDX dosyalarını kullanmak istiyorsanız Maintained onay kutusunu seçili olma durumunu iptal etmelisiniz.

Başlangıçta NDX uzantılı index dosyasının nasıl hazırlandığını göstermek için Maintained onay kutusunu pasif duruma getirdim. Index veri tabanı dosyasında bulunan bütün kayıtları dahil etmek istemiyorsanız Subset Condition (Filter) Expression metin kutusuna filtre şartlarını yazabilirsiniz. Gerekli ayarlamalar yapıp OK düğmesine basıldığında Save Index As diyalog kutusu ekrana gelir. Index File Name kutusunda dosya adı girilip Ok yapılırsa NDX uzantılı index dosyası hazırlanmış olur.

Define Index

Field list:

AD
SOYAD
ADRES
TEL
SEHIR

Indexed field:

SOYAD

Expression Index

Options:

☒ Unique

☐ Maintained

☐ Descending

Expression index:

Subset condition (filter) expression:

OK Cancel Help

Create dBASE for Windows Table: [Untitled]

Field roster:

	Field Name	Type	Size	Dec
1	AD	C	10	
2	SOYAD	C	15	
3	ADRES	C	30	
4	TEL	C	15	
5	SEHIR	C	10	

Table properties:

Indexes

Define... Modify...

SOYAD.NDX

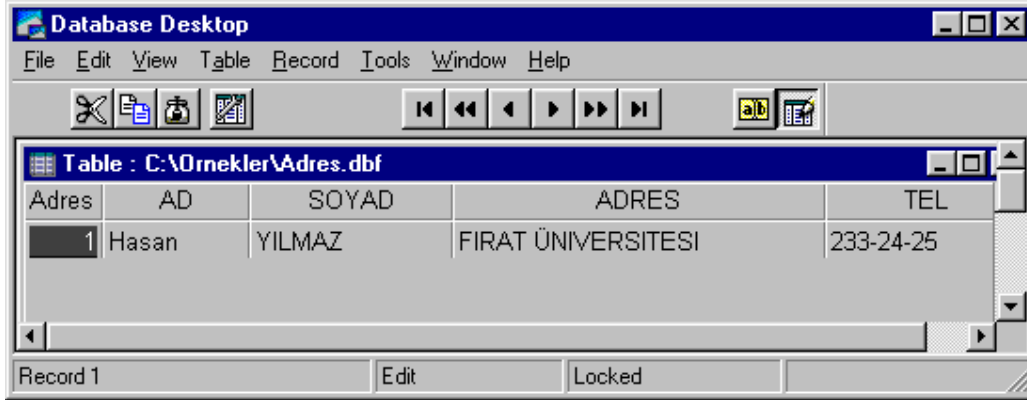
Erase

Record lock

☐ Info size

Borrow... Save As... Cancel Help

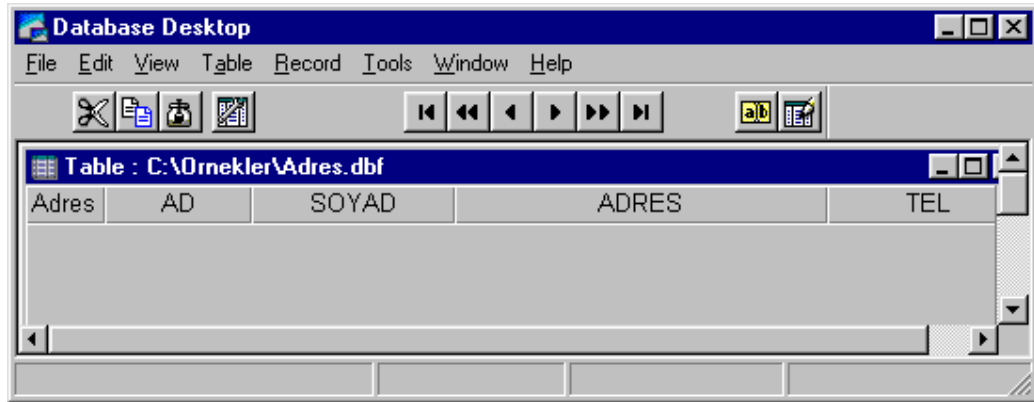
Daha önce hazırlamış olduğunuz Indexte değişiklik yapmak istiyorsanız Modify düğmesine, silmek istiyorsanız Erase düğmesine tıklama yapmanız gerekir. SOYAD.NDX uzantılı dosyayı silip MDX uzantılı dosya oluşturmak için bu dosya Erase ile silinir. Define düğmesine tıklayarak define index kutusunu ekrana getirilir.



MDX uzantılı dosya hazırlamak için Maintained onay kutusu seçilir. Index Tag Name metin kutusunda dosya adı yazılır.

Veri tabanı dosyası için anlatılan şekilde Index hazırlandıktan sonra, dosya için dil seçimi yapmak gerekir. Dil seçimi için ilk olarak Table Properties liste kutusunu açıp Table Language seçeneğini seçmek gerekir. Modify düğmesine tıklayıp dBASE TRK cp857 seçilir.

Bu işlem de tamamlandıktan sonra şimdi de kaydedelim. Save As düğmesine tıklayınca Save Table As diyalog kutusu ekrana gelir. Burada Alias DelphiBook olarak seçilir. Dosya Ornekler dizinine kaydedilir. Dosya açıldıktan sonra şu ekran gelir:



Veri tabanı dosyasına kayıt girişi yapmak istiyorsanız Table menüsünden Edit Data komutu veya F9 kısa yol tuşu kullanılabilir.

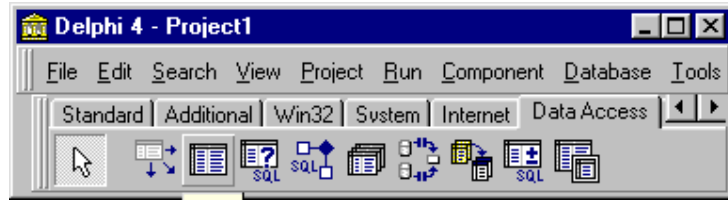
Eğer veri tabanı dosyasının yapısında değişiklik yapmak istiyorsanız Tools menüsünden Utilities alt menüsünden Restructure komutu verilir

BÖLÜM 15

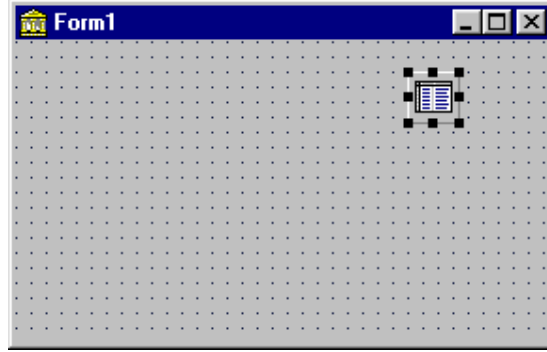
VERİ TABANI DOSYALARI ÜZERİNDE İŞLEM YAPMAK

Bu bölümde DataBase Desktop veya dBASE, Paradox gibi veri tabanı programları ile hazırlanmış olunan veri tabanı dosyaları üzerinde Delphi projeleri dahilinde nasıl işlem yapıldığını anlatacağım. Bu işlemi bir önceki bölümde hazırladım ve ADRES.DBF adıyla kaydettiğim veri tabanı dosyası üzerinde anlatacağım.

Bu amaçla yapılacak işlem, Component Palette araç çubuğundaki Data Access sekmesine tıklamaktır.

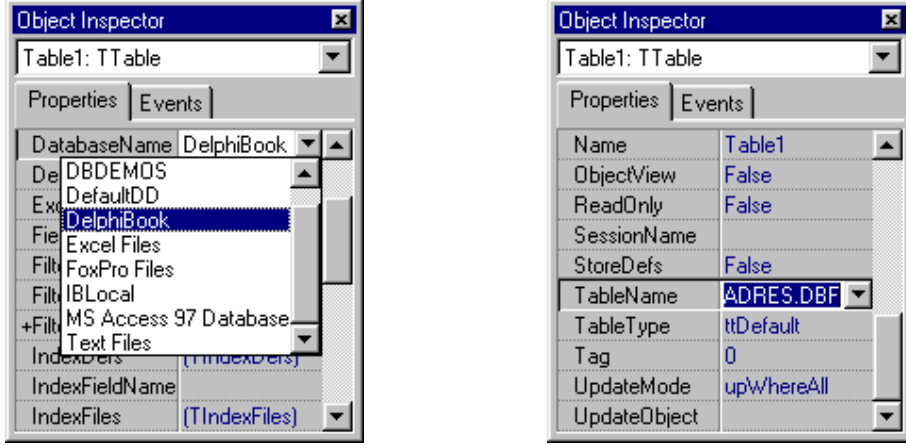


Data Access sekmesinde daha önce hazırlanıp kaydedilen bir veri tabanı dosyasının içeriğini görüntülemek veya dosyaya yeni kayıtlar eklemek için ilk gerek duyulan kayıttın adı **Table**' dir. Component Palette araç çubuğunda Data Access sekmesinde yer alan Table düğmesi seçili durumda iken projenin formu üzerinde tıklama yaparak Table nesnesi forma eklenir.



Object Inspector penceresinde Table nesnesi ile ilgili olarak en başta Database Name özelliğinden yararlanarak Alias seçimi yapılır. Üzerinde işlem yapmak istediğiniz dosyayı hangi Alias'a dayandırarak hazırladıysanız o Aliası seçmeniz gerekir. Bir önceki bölümde DelphiBook adında Bir Alias hazırlanmış ve ADRES.DBF dosyasını bu Alias'a dayandırarak hazırlamıştım. Table nesnesinin Database Name özelliği ile

Alias seçiminden sonra sıra veri tabanı dosyasını seçmeye gelir. Veri tabanı dosyası Table nesnesine ait TableName özelliğinden faydalanılarak Adres.DBF dosyası eklenir.



DatabaseName özelliği ile Alias , TableName özelliği ile veri tabanı dosyası seçimi yapıldıktan sonra sıra Table nesnesine ait **Active** özelliğne True değerini aktarmaya gelir. Böylece Table nesnesi ile ilgili olarak mutlaka yapılaması gereken işlemler yapıldı. Ancak Delphi’de veri tabanı dosyaları üzerinde pratik bir şekilde işlem yapmak için Table kontrolü tek başına yeterli gelmiyor. Table kontrolüne yardımcı olması için Forma ayrıca **DataSource** kontrolünün de dahil edilmesi gerekir.

Ancak bu bölümde önce DataSource kontrolünden yararlanmadan veri tabanı dosyası üzerinde işlem yapacağım. Bu amaçla forma eklediğimiz Table nesnesinin üzerine sağ fare ile tıklayıp table nesnesi aracılığı açılan dosyanın üzerinde işlem yapmak istenen alanları için önceden bazı tanımlamalar yapılmak için **Fields Editor** komutunu verdim. Table nesnesinin adını içeren boş bir diyalog kutusu ekrana gelir.

Açılan bu diyalog kutusunun içinde iken sağ fare tuşuna basılıp Add fields komutu verilirse veri tabanı dosyasında hazırlanan alanlar eklenir.



Çalışma anında üzerinde bulunan kaydın SOYAD alanının içeriği Table1SOYAD nesnesinin Text özelliğinde, AD alanının içeriği ise Table1AD nesnesinin Text özelliğinde saklanır. Table nesnesini içeren forma sahip olan proje çalıştırılır çalıştırılmaz kayıt okuma kafası daha önce seçilen veri tabanı dosyasındaki ilk kaydın üzerine konumlanır. Şimdi örnek olması için çalışma üzerinde tıklama yapıldığı zaman işletilecek bir program satırını projedeki tek formun FormClick yordamına dahil edeceğim.

ShowMessage (Table1AD.Text);

Table nesnesi yardımı ile açılan ve her alanı için Fields Editor penceresinde tanımlama yapılan bu dosyadaki aktif kaydın bütün içeriğini ekrana getirmek için üzerinde çalıştığım projenin tek formuna 5 Label ve 5 Edit kutusu ilave ettim. Table1AD nesnesine ait text değişkeninin içeriği, Edit1 kutusuna Table1SOYAD nesnesine ait text değişkeninin içeriği Edit2 metin kutusuna ilave edilir.

Bu işlemler çalıştırılır çalıştırılmaz otomatik olarak yapılacağı için, bu işlemleri yapacak program satırlarını forma ait FormCreate yordamına yazdım.

begin

Edit1Text := Table1AD.Text;

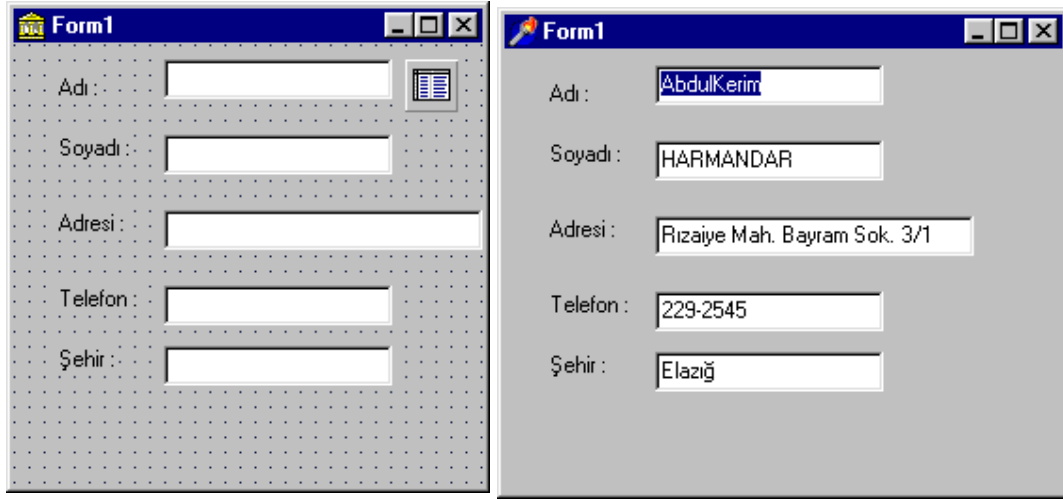
Edit2Text := Table1SOYAD.Text;

Edit3Text := Table1ADRES.Text;

Edit4Text := Table1TEL.Text;

Edit5Text := Table1SEHIR.Text;

Proje çalıştırılır çalıştırılmaz aşağıdaki ekran görüntüsü elde edildi.



Form1

Adı :

Soyadı :

Adresi :

Telefon :

Şehir :

Form1

Adı :

Soyadı :

Adresi :

Telefon :

Şehir :

15.1 Dosyadaki Kayıtlar Arasında Dolaşmak

Daha önce hazırladığım ADRES.DBF adındaki veri tabanı dosyasında yalnızca bir kayıt vardı. Database Desktop programını çalıştırdım ve dosyadaki kayıt sayısını üçe çıkardım.



Database Desktop

File Edit View Table Record Tools Window Help

Table : C:\Ornekler\ADRES.DBF

ADRES	AD	SOYAD	ADRES	TEL
2	Fatih	ÖZSARI	Ankara/ Keçiören	0312-356-854
3	AbdulKerim	HARMANDAR	Rızaiye Mah. Bayram Sok. 3/1	229-2545
4	Mehmet	Yılmaz	Rızaiye Mah. Bayram Sok. 3/1	233 96 63

Record 2

Çalışma anında tablodaki bir sonraki kayıt üzerine gitmek için projenin formuna Sonraki Kayıt başlığına sahip bir düğme dahil ettim. Table kontrolü yardımı ile açılan veri tabanı dosyasındaki kaydın üzerine gitmek için **Next** metodundan yararlanılmaktadır. Bu amaçla çalışma anında formdaki düğmenin üzerinde tıklama yapıldığı zaman işletilmek üzere düğmeye ait Click yordamını şöyle düzenledim:

```
Begin;  
Table1.Next;  
Edit1.Text := Table1AD.Text;  
Edit2.Text := Table1SOYAD.Text;  
Edit3.Text := Table1ADRES.Text;  
Edit4.Text := Table1TEL.Text;  
Edit5.Text := Table1SEHIR.Text;
```

Eğer çalışma anında bir önceki kayda dönmek istiyorsanız eklenen düğmeye **Prior** metodu uygulanmalıdır. Yani;

```
Begin;  
Table1.Prior;  
Edit1.Text := Table1AD.Text;  
Edit2.Text := Table1SOYAD.Text;  
Edit3.Text := Table1ADRES.Text;  
Edit4.Text := Table1TEL.Text;  
Edit5.Text := Table1SEHIR.Text;
```

15.2 Dosyalarda Yeni Kayıtlar Eklemek ve Kayıtlarda Değişiklik Yapmak

Table nesnesi ile açılıp erişim sağlanan veri tabanı dosyasının sonuna yeni bir kayıt eklemek için **Append** metodundan yararlanmaktır. Yeni Kayıt adlı ikona aşağıdaki program satırları yazılır;

```
Begin;  
Table1.Append;  
Edit1.Text := ' ';  
Edit2.Text := ' ';  
Edit3.Text := ' ';  
Edit4.Text := ' ';  
Edit5.Text := ' ';  
Edit1.SetFocus;
```

Önce dosyanın sonuna içeriği boş bir kayıt eklenir. Ardından aktif kaydın içeriğini görüntülemeye kullandığımız metin kutularının içeriğini silmektir. SetFocus metodu

yardımı ile ekleme noktası bilgi girişi için formdaki ilk metin kutusunda hazır olarak bekler. Çalışma anında bu metin kutularına girilecek bilgileri dosyanın sonuna yazılan boş kayda aktarmak için Forma “Değiştir” başlığına sahip bir düğme ekledim. Üzerinde bulunan kaydın içeriğın formdaki metin kutularına girilmiş veya değiştirilmiş bilgiler ile değiştirmek için **UpdateRecord** metodundan yararlanılmaktadır. Bu nedenle ortama ait Click yordamını aşağıdaki gibi düzenledim;

```
Begin;  
Table1.Edit;  
Table1AD.Text := Edit1.Text;  
Table1SOYAD.Text := Edit2.Text;  
Table1ADRES.Text := Edit3.Text;  
Table1TEL.Text := Edit4.Text;  
Table1SEHIR.Text := Edit5.Text;  
Table1.UpdateRecord;
```

15.3 Dosyadan Kayıt Silmek

Veri tabanı dosyasından hangi kaydı silmek istiyorsanız , o kaydı aktif hale getirip **Delete** metodunu kullanmanız yeterlidir. Bunun için form üzerine Kayıt Sil düğmesi ekledim. Ardından bu yordamın Click ortamına aşağıdaki program satırını yazdım.

```
Begin ;  
Table1.Delete;
```

```

Table1.Prior;
Edit1.Text := Table1AD.Text;
Edit2.Text := Table1SOYAD.Text;
Edit3.Text := Table1ADRES.Text;
Edit4.Text := Table1TEL.Text;
Edit5.Text := Table1SEHIR.Text;
end;

```

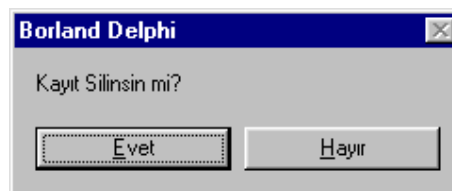
Bütün gelişmiş programlarda olduğu gibi kayıt silme işlemi öncesi kullanıcıdan onay alabilirsiniz. Kayıt silme işlemi öncesinde kullanıcıdan onay almak için “Kayıt Sil” başlıklı düğme için hazırladığım program kodlarını aşağıdaki gibi düzenledim.

```

Var;
Tus : Integer;
Begin;
Tus := Application.MessageBox('Kayıt Silinsin mi?',
    'Borland Delphi', mb_YesNo);
if Tus = IDYES Then
Begin;
Table1.Delete;
Table1.Prior;
Edit1.Text := Table1AD.Text;
Edit2.Text := Table1SOYAD.Text;
Edit3.Text := Table1ADRES.Text;
Edit4.Text := Table1TEL.Text;
Edit5.Text := Table1SEHIR.Text;

```

Bu düzenlemelerden sonra program çalıştırılacak olunursa aşağıdaki mesaj kutusu ile karşılaşılır.



15.4 Veri Tabanı Dosyalarını Açmak ve Kapatmak

Bu bölümün daha önceki sayfalarında belirtildiği gibi Delphi projeleri içerisinde Table nesnesi ile temsil edilen veri tabanı dosyasını açmak için tasarım anında Object Inspector penceresinde Table nesnesine ait Action değişkenine True değerini aktarmak yeterli oluyordu. Bunun dışında çalışma anında program yazarak Table nesnesine ait Action değişkenine True değeri aktararak dosya açılabilirdi. Bu anlatılanların dışında, Open metodu ile veri tabanı dosyası açılabilir ve Close metodu ile kapatabilirsiniz.

```
Begin
Table1.Open;
Edit1.Text := Table1AD.Text;
Edit2.Text := Table1SOYAD.Text;
Edit3.Text := Table1ADRES.Text;
Edit4.Text := Table1TEL.Text;
Edit5.Text := Table1SEHIR.Text;
```

15.5 Index Dosyalarını Kullanmak

Daha önce ADRES.DBF dosyasını Database Desktop ile hazırlarken Soyad alanına göre indexleme yapmıştık. Şimdiye kadar hazırlandığını bu indexten yararlanmadım. Bu nedenler kayıtlar ekrana kayıt edildikleri sıra ile geldi. Soyada göre kayıtları ekrana getirmek için Table nesnesi seçili iken Object Inspector penceresinde **IndexName** özelliğe ait liste kutusu açılır. Soyad adlı Index seçilir.

Bunun dışında Index tanımlanması yapılan diyalog kutusunda Unique adında bir onay kutusu vardı. Eğer bu aktif hale getirilirse çalışma anında yalnızca aynı soyada sahip ilk kayda erişim sağlanabilir. Bunun için Form Click ortamına şu satır yazılır:

```
Begin
Table1.Index := ' ' ;
```

Eğer tasarım anında Object Inspector penceresi aracılığı ile bir indexi seçmediyseniz, çalışma anında geçerli olmak üzere program kodu yazarak Index name

değişkeninin içeriğini değiştirip daha önce hazırlamış olduğunuz indexi devreye sokabilirsiniz.

```
Table1.IndexName := 'SOYAD';  
Table1.Open;  
Edit1.Text := Table1AD.Text;  
Edit2.Text := Table1SOYAD.Text;  
Edit3.Text := Table1ADRES.Text;  
Edit4.Text := Table1TEL.Text;  
Edit5.Text := Table1SEHIR.Text;
```

15.6 Dosyadaki İlk ve Son Kaydın Üzerine Gitmek

Delphi projesi içerisinde Table nesnesi ile temsil edilen veri tabanı dosyasındaki ilk kaydın üzerine bir seferde gitmek için **First** değiminden faydalanılır.

```
Table1.First;  
Edit1.Text := Table1AD.Text;  
Edit2.Text := Table1SOYAD.Text;  
Edit3.Text := Table1ADRES.Text;  
Edit4.Text := Table1TEL.Text;  
Edit5.Text := Table1SEHIR.Text;
```

Eğer istenilen kayda belirli aralıklarla gidilmek isteniyorsa o zaman program satırına şunları yazmak gerekir;

```
Table1.MoveBy(Adım değeri);  
Edit1.Text := Table1AD.Text;  
Edit2.Text := Table1SOYAD.Text;  
Edit3.Text := Table1ADRES.Text;  
Edit4.Text := Table1TEL.Text;  
Edit5.Text := Table1SEHIR.Text;
```

15.7 Dosya Sonu Kontrolü – EOF ve BOF Özellikleri

Kayıt okuma kafasını dosya sonunda olup olmadığını Table nesnesine ait **EOF** özelliğinden yararlanılarak öğrenilir. Eğer kayıt okuma kafası dosyanın sonunda ise EOF özelliği True değerine, değilse False değerine sahip olur. EOF özelliğinin işlevini görmek için çalıştığım projenin bir sonraki kayda gitmede kullandığımız yordamı aşağıdaki gibi değiştirdim.

```
Table1.Next;  
Edit1.Text := Table1AD.Text;  
Edit2.Text := Table1SOYAD.Text;  
Edit3.Text := Table1ADRES.Text;  
Edit4.Text := Table1TEL.Text;  
Edit5.Text := Table1SEHIR.Text;  
If Table1.Eof = True then  
    ShowMessage ('Dosyanın Sonundasınız!!!');
```

15.8 Veri Tabanı Dosyasını Yalnız Okunabilir Yapmak – (Read Only)

Eğer table nesnesi aracılığı ile açtığınız dosyada herhangi bir değişiklik yapmak istemiyorsanız Object Inspector penceresinde Table nesnesine ait ReadOnly değişkenine True değerini aktarabilirsiniz. Ancak açık veya Active değişkeni True değerini içeren bir Table nesnesinin ReadOnly özelliğinde değişiklik yapmak mümkün değildir. Bunun için Table nesnesine ait Active değişkenine false değerini aktarıp ondan sonra ReadOnly değişkenine True değerini aktarmanız gerekir. ReadOnly değişkenine true değerini aktardıktan sonra tekrar active değişkenine True değerini aktarıp dosyanın içeriğine erişim sağlayabilirsiniz.

BÖLÜM 16

STANDART FONKSİYONLAR

16.1. Matematiksel Fonksiyonlar

Matematik işlemleri yapmak için gerekli komutların tamamı Delphi’de mevcut değildir. Ancak mevcut komutlarla olmayan komutları türetmek mümkündür.

Matematiksel işlemleri yapmak için gerekli bir çok fonksiyon **math** ve **system** unitlerinde tanımlanmıştır. Math unitinde tanımlanmış fonksiyonları kullanabilmeniz için formunuzdaki **uses** satırına math unitini eklemeniz gerekir.

Uses

*Windows, Message, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, **math**;*

Uses satırı formunuzun en başında, **interface** kısmı altındadır. Yukarıdaki satırdaki unitlerin çoğu form oluşturulduğunda eklenir. Ancak **Math** uniti eklenmez. Bu satırı yukarıdaki gibi sizin eklemeniz gerekir.

Sık kullanılan matematiksel fonksiyonların bazıları aşağıda açıklanmıştır.

16.1.1. ABS() Fonksiyonu

Bir sayının mutlak değerini bulmak için kullanılır.

Genel Yazılışı:

ABS (Sayı)

Örnek:

Sonuc:=ABS(-15) // Sonuc=15 olur.

16.1.2. INT() Fonksiyonu

Ondalık nokta içeren bir sayıyı tamsayıya çevirmek için kullanılır.

Genel Yazılışı:

Int (Ondalık sayı)

Örnek:

Sonuc:= int(10,25) // Sonuc=10 olur.

Sonuc:= int(-10,25) // Sonuc= -10 olur.

Verilen örneklerden tespit edilebileceği gibi `int()` fonksiyonu kendisine parametre olarak verilen, ondalık nokta içeren sayılarda yuvarlama yapmayıp noktadan sonrasını atmaktadır.

16.1.3. Yuvarlatma Fonksiyonları

Genel Yazılışları:

Floor (Ondalık Sayı)

Ceil (Ondalık Sayı)

Frac (Ondalık Sayı)

Floor fonksiyonu sayıyı aşağıya doğru yuvarlatırken **ceil** fonksiyonu yukarıya doğru yuvarlatır. **Frac** fonksiyonu sayının ondalık kısmını verir.

Örnek:

Var

X: Real;

Begin

X:= Frac (10.46); // X=0.46

X:= Floor (5.9); // X=5

X:= Ceil (5.9); // X=6

X:= Ceil (-3.2); // X = -3

End;

16.1.4. SQRT() Fonksiyonu

Bu fonksiyon, pozitif bir sayının karekökünü bulmak amacıyla kullanılır.

Genel Yazılışı:

SQRT (Karekökü Alınacak Sayı)

16.1.5. Üst Alma Fonksiyonları

Genel Yazılışları:

Power (taban,üst)

Sqr (Sayı)

Exp (Sayı)

x^y işlemini yapabilmek için **Power** fonksiyonu kullanılır. e^x işlemini yapabilmek için **exp** fonksiyonu kullanılır. **Sqr** fonksiyonu ise bir sayının karesini almak için kullanılır.

Örnek:

Var

X: Real;

Begin

X := Power (3,2.5); // 3 ün 2.5 kuvveti

X := exp (3); // e sayısının 3. kuvveti

X := sqr (5); // $5^2=25$

End;

16.1.6. Mod Operatörü

Mod operatörü x sayısının y sayısına bölümünde kalanı verir.

Genel Yazılışı:

X mod Y

16.2. Rasgele Sayı Üretim Fonksiyonları

16.2.1. Random Fonksiyonu

Genel Yazılışı:

Random [(ÜstSınır)];

Fonksiyon 0 ile ÜstSınır arasında rasgele sayı üretir. ÜstSınır parametresi verilmezse 0 ile 1 arasında sayı üretilir.

Rasgele sayı üretme fonksiyonu hep aynı şekilde çalışacağı için belirli anlarda üretilen sayılar da hep aynı olacaktır. Yani program her çalıştığında aynı sayıları tutacaktır. Bunu önlemek için rasgele sayı üretme işleminin başlangıç değeri için değişen bir referans almak gerekir. Bilgisayarda sürekli olarak değişen şey sistem saati

olduğu için başlangıç referansı saatin o anki değerinden alınırsa program her başlatıldığında aynı sayıları üretmeyecektir. Bunun için Randomize fonksiyonu girişte kullanılmalıdır.

16.2.2. Randomize Fonksiyonu

Bu fonksiyonun yaptığı tek iş rasgele sayı üretim fonksiyonunun referansını belirlemektir. Fonksiyondan her hangi bir değer geri dönmez. Yalnız Random fonksiyonunun ürettiği sayıları etkiler.

16.3. Karakterel Fonksiyonlar

16.3.1. Length() Fonksiyonu

Length() fonksiyonu, kendisine parametre olarak verilen karakterel veya başka bir tip bilginin uzunluğunu bulur. Karakterel bilgiler Length() fonksiyonuna doğrudan veya bir değişken aracılığı ile verilebilir.

Örnek:

Ad= “Ümit”

Uzunluk := Length (ad) // Uzunluk 4 olur.

Uzunluk:=Length (“Bilgisayar”) // Uzunluk 10 olur.

16.3.2. Copy() Fonksiyonu

Bu fonksiyon kendisine parametre olarak verilen karakterel (string) değişken veya sabit bilginin soldan itibaren istenilen kadarını ayırmak, çıkarmak amacıyla kullanılmaktadır.

Genel Yazılışı:

Copy (karakterel ifade, başlama yeri, uzunluk)

Copy() fonksiyonu parametre olarak aldığı karakterel bilgiden, ikinci parametrede belirtilen yerden üçüncü parametrede belirtilen kadarını soldan itibaren çıkarıp geri döndürür. Uzunluk değerinin 0 olarak verilmesi halinde geriye boşluk (null)

değeri döndürülür. Uzunluk değeri karakter sel bilginin uzunluğundan fazla ise geriye karakter sel bilginin kendisi döndürülür.

Örnek:

Sehir := ‘Antalya’;

Sonuc := Copy (sehir,3,3); // Sonuc “Ant” olur.

16.3.3. LowerCase() Fonksiyonu

Bu fonksiyon parametre olarak aldığı karakter sel bilginin içinde bulunan alfabetik karakterlerin büyük olanlarını küçük harfe çevirir.

Genel Yazılışı:

LowerCase (Karakter sel Bilgi)

16.3.4. UpperCase() Fonksiyonu

Bu fonksiyon parametre olarak aldığı karakter sel bilginin içinde bulunan alfabetik karakterlerin küçük olanlarını büyük harfe çevirir.

Genel Yazılışı:

UpperCase (Karakter sel Bilgi)

16.3.5. Pos() Fonksiyonu

Bu fonksiyon karakter sel bir bilgide başka bir karakter sel bilgiyi aramak amacıyla kullanılır. Bu fonksiyon birinci bilginin ikinci bilgideki ilk yerini sayısal bilgi olarak geriye döndürür.

Genel Yazılışı:

Pos (Aranacak Bilgi, Karakter sel Bilgi)

Örnek:

Ornek := Pos (‘a’, ‘antalya’); // ornek 1 olur.

Ornek := Pos (‘n’, ‘antalya’); // ornek 6 olur.

16.3.6. Insert() Deyimi

Bu deyim karakterse bilginin istenilen yerine başka bir karakterse bilgiyi dahil etmek için kullanılır.

Genel Yazılışı:

Insert (Eklenecek Bilgi, Hedef Bilgi, Başlama Yeri)

Eklenecek karakterse sabit bilgi veya karakterse bilgi içeren değişken, Insert() deyimine birinci parametre olarak verilir. Ekleme işleminin yapılacağı karakterse bilgi ise ikinci parametre ile belirtilir. Üçüncü parametre ise ekleme işleminin başlayacağı yeri belirtir.

Örnek:

Var

Kitap1 : String [30];

Kitap2 : String [20];

Begin

Kitap1 := 'Delphi ile Programcılık';

Kitap2 := 'Görsel';

Insert (Kitap2, Kitap1, 12);

Edit1.Text := Kitap1;

End;



Şekil 16.1. Insert Deyiminin etkisi

16.3.7. Delete() Deyimi

Bu deyim ile karakterse bilginin istenen yerinden istenen kadar silinebilir. Eğer karakterse sabit bir bilgi veya karakterse bilgi içeren bir değişkenin içeriğinin bir kısmını silmek istiyorsanız bu deyimden yararlanabilirsiniz.

Genel Yazılışı:

Delete (karakterse bilgi, başlama yeri, silinecek karakter sayısı)

Örnek:

Var

Kitap1: String [30];

Begin

Kitap1:= 'Delphi ile Görsel Programcılık';

Edit1.Text := Kitap1;

Delete (kitap1,11,7);

Edit2.Text := Kitap1;

End;



Şekil 16.2. Delete Deyiminin etkisi

16.4. Tip Değiştirme Fonksiyonları

Bazen bir sayıyı bir string içinde veya bir reel sayıyı tam sayı gibi kullanmak gerekebilir. Örneğin bir sayıyı Text kutusunda normal olarak gösteremezsiniz. Bu gibi durumlarda tip dönüştürme fonksiyonlarını kullanmanız gerekir. Aynı durum text kutusuna girilen bir sayının işleme tabi tutulması içinde gereklidir.

16.4.1. IntToStr() Fonksiyonu

Tamsayı tipindeki bir değişken, string bir değişkene atanmadan önce bu fonksiyonla stringe çevrilmelidir.

Genel Yazılışı:

IntToStr (Sayı)

Örnek:

Procedure TForm1.FormClick (Sender: TObject);

Var

Yil: Integer;

Begin

Yil := 1999;

Edit1.Text := IntToStr (yil);

End;

Yukarıdaki örneğin sonucunda edit kutusunda ‘1999’ yazar. Eğer tam sayı olan yıl değişkenini stringe çevirmeseydik derleyici hata verirdi.

16.4.2. StrToInt() Fonksiyonu

String değişken içindeki bir sayıyı örneğin Edit kutusuna yazılmış bir sayıyı aritmetik bir işlemde kullanabilmek veya sayısal değişkene aktarabilmek için sayıya çevirmek gerekir. String ifadeleri tam sayıya çevirebilmek için StrToInt fonksiyonu kullanılır.

Genel Yazılışı:

StrToInt (String ifade)

Örneğin edit kutusuna girilen bir sayıyı 5 ile çarpıp sonucu edit kutusunda göstermek için aşağıdaki kodu yazmamız gerekir.

```
Edit1.Text := IntToStr(StrToInt(Edit1.Text) * 5);
```

Burada önce Edit1 içindeki sayı üzerinde işlem yapabilmek için integer tipine çevirdik, sayıyı 5 ile çarptıktan sonra sonucu edit kutusunda göstermek için tekrar stringe çevirdik.

16.4.3. Chr() Fonksiyonu

Bu fonksiyon 0 ile 255 arasındaki sayıya karşılık gelen ASCII karakterini elde etmek amacıyla kullanılır.

Genel Yazılışı:

CHR (ASCII Kodu)

Örnek:

```
Label1.Caption := 'ASCII 65 sayısının karakter kodu : ' + CHR (65);
```

```
// Sonuç: ASCII 65 sayısının karakter kodu : A
```

16.4.4. Ord () Fonksiyonu

Verilen karakterin ASCII kodunu verir.

Genel Yazılışı:

Ord (Tek karakterlik bilgi)

Örnek:

```
Label1.Caption := 'A harfinin ASCII kodu : ' + IntToStr (Ord('A'));  
// Sonuç : A harfinin ASCII kodu: 65
```

16.4.5. Val() Fonksiyonu

Bu fonksiyon stringi herhangi tipteki bir sayıya çevirir. Eğer çevirme işlemi sonucunda bir hata oluşursa hatanın olduğu karakter numarası **Durum** parametresi ile geri döner. Durum parametresi 0 ise string başarılı bir şekilde çevrilmiştir.

Genel Yazılışı:

Val (String, Sayı, Durum)

Örnek:

```
Var  
I, Durum : Integer;  
Begin  
    Val (Edit1.Text, I, Durum);  
    If Durum <> 0 Then {Hata oluştu ise}  
        ShowMessage (IntToStr (Durum) + ' . karakter hatalı')  
    Else  
        ShowMessage ('Sonuç = ' + IntToStr (I));  
End;
```

16.5. Tarih ve Zaman Fonksiyonları

Delphi’de tarih formatı string olarak değil Float tipindeki bir sayıda tutulur. Bu sayının tam kısmında tarih, ondalık kısmında ise saat tutulur.

Tarih.Saat

Delphi’deki bu yapı dolayısı ile gerçek hayatta kullandığımız string formatındaki “04/05/1999 19:12.15” tarih ve saati Delphi’de kullanmadan önce Delphi tarih formatına, Delphi formatındaki tarihi de kullanmadan önce string formatına çevirmemiz

gerekir. Delphi’de tarih ve saatin sayı formatında olması üzerinde işlem yapılmasını kolaylaştırır. Örneğin;

```
Label1.Caption := DateToStr(Date-5);
```

satırı ile bugünden 5 gün önceki tarihi bulabiliriz.

16.5.1. Date (Tarih) Fonksiyonu

Date fonksiyonu şu anki tarihi yukarıda bahsedilen sayı tipinde verir. Kullanmadan önce DateToStr fonksiyonu ile stringe çevrilmelidir.

Örnek:

```
Label1.Caption := 'Bugün: ' + DateToStr(date);
```

```
Label2.Caption := '100 gün önce: ' + DateToStr(date-100);
```

16.5.2. Time (Saat) Fonksiyonu

Bu fonksiyon şu anki saati verir. Yine kullanmadan önce TimeToStr fonksiyonu ile stringe çevrilmelidir.

Örnek:

```
Form1.Caption := 'Saat : ' + TimeToStr(Time);
```

16.5.3. Now (Tarih ve Saat) Fonksiyonu

Now fonksiyonu tarih ve saati birlikte verir. Now ile Date+Time aynı işi yapar. Kullanmadan önce tarih kısmını yazdırmak için DatetoStr, saat kısmını yazdırmak için TimeToStr, her ikisini birden yazdırmak için DateToStr fonksiyonu kullanılmalıdır. Now fonksiyonu tarih ve saati şu formatta verir:

4/5/1999 19:30:45

Örnek:

```
Label1.Caption := DateToStr (now);
```

```
Label2.Caption := TimeToStr (now);
```

```
Label3.Caption := DateTimeToStr (now);
```

16.5.4. DayOfWeek (Haftanın Günü) Fonksiyonu

Bu fonksiyon verilen tarihin haftanın hangi günü olduğunu 1 ile 7 arasında bir sayı ile gösterir. Haftanın 1. günü pazar iken 7. günü ise cumartesidir.

Örnek olarak kullanıcıya doğum tarihini soran ve hangi gün olduğunu bulan bir program yazalım. Programımız için form üzerine bir label ve bir komut düğmesi yerleştirerek aşağıdaki kodları yazalım.

Örnek:

Procedure TForm1.Button1Click (Sender: TObject);

Begin

Case DayOfWeek (StrToDate(InputBox('DT', 'Doğum Tarihiniz ?', '')));

1: Label1.Caption := 'Pazar';

2: Label1.Caption := 'Pazartesi';

3: Label1.Caption := 'Salı',

4: Label1.Caption := 'Çarşamba';

5: Label1.Caption := 'Perşembe';

6: Label1.Caption := 'Cuma';

7: Label1.Caption := 'Cumartesi';

End;

Label1. Caption := Label1.Caption + 'günü doğmuştunuz';

End;

16.6. Dizin ve Dosya Yönetim Fonksiyonları

16.6.1. GetDir() Fonksiyonu

Genel Yazılışı:

GetDir (SürücüNo, Dizin)

SürücüNo parametresi ile verilen sürücüdeki (0: Aktif, 1: A, 2: B, 3: C ...) aktif dizini Dizin parametresi ile geri gönderir.

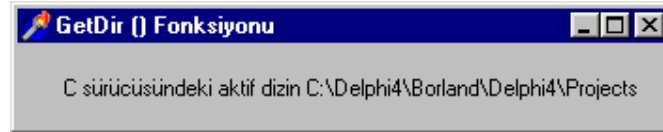
Örnek:

Var

```

D: String;
Begin
    GetDir (3,d);
    Label1.Caption := 'C sürücüsündeki aktif dizin' + d;
End;

```



Şekil 16.3. Örnek programın sonucu

16.6.2. ChDir() Fonksiyonu

Aktif dizini değiştirmek için kullanılır.

Genel Yazılışı:

ChDir(Dizin)

Örnek:

```

ChDir ('C:\Windows');
If IOResult <> 0 Then
    ShowMessage ('Windows dizinine geçilemedi');

```

16.6.3. Mkdir() Fonksiyonu

Genel Yazılışı:

Mkdir(dizin)

Dizin parametresi ile verilen dizini oluşturur.

Örnek:

```

Mkdir ('Gecici');
If IOResult <> 0 Then ShowMessage ('Dizin oluşturulamadı');

```

16.6.4. Rmdir() Fonksiyonu

Genel Yazılışı:

Rmdir (Dizin)

Dizin parametresi ile verilen dizini siler.

Örnek:

```
Rmdir ('Gecici');  
If IOResult <> 0 Then ShowMessage ('Silinemedi');
```

16.6.5. IOResult (Son Disk İşleminin Durumu)

Bu fonksiyon son yapılan işlemden sonra herhangi bir hata oluşup oluşmadığını belirtir. Geriye dönen değeri 0 ise işlem başarılmıştır.

16.6.6. DeleteFile Deyimi

Verilen dosyayı diskten siler. Eğer dosya yoksa veya silinemediyse geriye False değeri döner.

Genel Yazılışı:

DeleteFile (Dosya)

Örnek:

```
DeleteFile ('c:\gecici.doc');
```

16.6.7. RenameFile Deyimi

Genel Yazılışı:

RenameFile (EskiAd, YeniAd)

EskiAd parametresi ile verilen dosyanın ismini YeniAd parametresi ile verilen isme çevirir. EskiAd ile YeniAd'ın kullandığı sürücü veya dizinler farklı ise dosya taşınır. Geri dönen değer false ise işlem başırlanamamıştır. (Aynı isimli bir dosya olabilir.)

Örnek:

```
If RenameFile ('Gereksiz.Doc', 'Gerekli.Doc') = False Then  
    ShowMessage ('Değiştirilemedi')  
Else  
    ShowMessage ('Değiştirildi');
```

