T.C.
MİLLÎ EĞİTİM BAKANLIĞI

MEGEP

(MESLEKİ EĞİTİM VE ÖĞRETİM SİSTEMİNİN
GÜÇLENDİRİLMESİ PROJESİ)

BİLİŞİM TEKNOLOJİLERİ

VERİ YAPILARI

ANKARA 2007

Milli Eğitim Bakanlığı tarafından geliştirilen modüller;

- Talim ve Terbiye Kurulu Başkanlığının 02.06.2006 tarih ve 269 sayılı Kararı ile onaylanan, Mesleki ve Teknik Eğitim Okul ve Kurumlarında kademeli olarak yaygınlaştırılan 42 alan ve 192 dala ait çerçeve öğretim programlarında amaçlanan mesleki yeterlikleri kazandırmaya yönelik geliştirilmiş öğretim materyalleridir (Ders Notlarıdır).

- Modüller, bireylere mesleki yeterlik kazandırmak ve bireysel öğrenmeye rehberlik etmek amacıyla öğrenme materyali olarak hazırlanmış, denenmek ve geliştirilmek üzere Mesleki ve Teknik Eğitim Okul ve Kurumlarında uygulanmaya başlanmıştır.

- Modüller teknolojik gelişmelere paralel olarak, amaçlanan yeterliği kazandırmak koşulu ile eğitim öğretim sırasında geliştirilebilir ve yapılması önerilen değişiklikler Bakanlıkta ilgili birime bildirilir.

- Örgün ve yaygın eğitim kurumları, işletmeler ve kendi kendine mesleki yeterlik kazanmak isteyen bireyler modüllere internet üzerinden ulaşılabilirler.

- Basılmış modüller, eğitim kurumlarında öğrencilere ücretsiz olarak dağıtılır.

- Modüller hiçbir şekilde ticari amaçla kullanılamaz ve ücret karşılığında satılamaz.

İÇİNDEKİLER

AÇIKLAMALAR

KOD	**481BB0027**
ALAN	**Bilişim Teknolojileri**
DAL/MESLEK	**Alan Ortak**
MODÜLÜN ADI	**Veri Yapıları**
MODÜLÜN TANIMI	Dizi, işaretçiler ve nesneye yönelik program yazma ile ilgili öğrenme materyalidir.
SÜRE	40/24
ÖN KOŞUL	Yapısal Programlama Temelleri modülünü almış olmak.
YETERLİK	Veri yapılarıyla program yazmaya hazırlık yapmak
MODÜLÜN AMACI	**Genel Amaç** Gerekli ortam sağlandığında, dizi ve işaretçileri tanıyıp nesneye yönelik program yazabileceksiniz. **Amaçlar** 1. Dizi mantığını anlayacak ve dizi değişkenleri kullanabileceksiniz 2. İşaretçiler ile program yazabileceksiniz 3. Nesneye yönelik program yazabileceksiniz
EĞİTİM ÖĞRETİM ORTAMLARI VE DONANIMLARI	Bilgisayar laboratuvarı ve bu ortamda bulunan; bilgisayar, yazıcı, bilgisayar masaları, kâğıt, kalem, lisanslı işletim sistemi programı ve akış diyagramı sembolleri ile ilgili panolar.
ÖLÇME VE DEĞERLENDİRME	Her faaliyet sonrasında o faaliyetle ilgili değerlendirme soruları ile kendi kendinizi değerlendireceksiniz. Modül içinde ve sonunda verilen öğretici sorularla edindiğiniz bilgileri pekiştirecek, uygulama örneklerini ve testleri gerekli süre içinde tamamlayarak etkili öğrenmeyi gerçekleştireceksiniz. Sırasıyla araştırma yaparak, grup çalışmalarına katılarak ve en son aşamada alan öğretmenlerine danışarak ölçme ve değerlendirme uygulamalarını gerçekleştireceksiniz..

GİRİŞ

Sevgili Öğrenci,

Bu modül ile sizlere diziler, işaretçiler ve nesne tabanlı programlama konuları ile ilgili yeterlik kazandıracaktır. Öncelikle bu modülü daha iyi anlamanız için, daha önceki programlama temelleri modüllerinde kavrayamadığınız yerler varsa onları pekiştiriniz. Değişkenler, döngüler ve dosyalama konularını tekrar gözden geçirmeniz tavsiye edilir.

Önceki modülleri anlayarak buraya gelmiş iseniz, daha ileri seviye programlama konularına geçebilirsiniz. "**Veri Yapıları**" modülü ile bir program yazımında gerekli olan bazı eksikliklerimizi gidereceksiniz. Bilgisayarın belleğini ve programınızdaki verileri daha etkin kullanacaksınız.

Programlamada veri yapıları en önemli unsurlardan birisidir. Program yazarken kullanılacak veri yapısının en ideal şekilde belirlenmesi, programcılıkta biraz daha ustalaşman anlamına gelmektedir. Program içerisinde işlenecek veriler diziler ile tanımlanmış bir veri bloğu içerisinden seçilebileceği gibi, işaretçiler kullanılarak daha etkin şekilde hafızada saklanabilir. Veri yapıları, dizi ve işaretçiler ile yapılmasının yanında, nesneler ile de gerçekleştirilebilir.

ÖĞRENME FAALİYETİ–1

AMAÇ

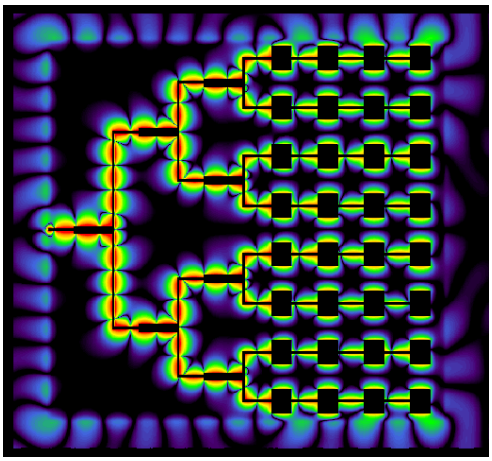
Dizi mantığını anlayacak, dizi ve yapı değişkenleri kullanabileceksiniz.

ARAŞTIRMA

Bu faaliyet öncesinde hazırlık amaçlı aşağıda belirtilen araştırma faaliyetlerini yapmalısınız.

- Banka ve hastane gibi yerlerde kuyrukta bekleyenlerin işini kolaylaştıran "Sıramatik" isimli programı inceleyiniz. Sizce bu program nasıl daha kullanışlı hâle getirilebilir?
- Kargo firmalarının (Aras, Fedex ve DHL gibi) kargo takip programını inceleyiniz. Paketlerin internet üzerinden nasıl takip edildiğini inceleyiniz.

1. DİZİLER



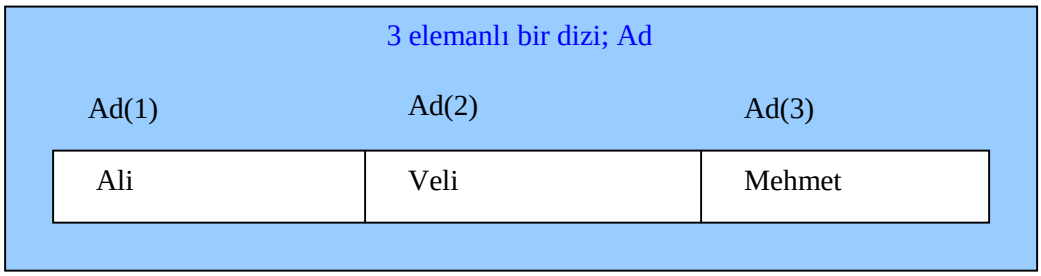
Birkaç tane değişken ile küçük programları idare edebilirsiniz. Programlamada uzun ve benzer bilgilerle dolu değer listelerin oluşturulması "**dizi -** *array*" ile yapılmaktadır. Veri yapısını aşağıdaki gibi ayrı değişken yapabileceğiniz gibi:

```
Ad1 = "Ali"
Ad2 = "Veli"
Ad3 = "Mehmet"
```

Bunun yerine dizi kullanarak, tek değişken ile şu şekilde yapabilirsiniz:

```
Ad(1) = "Ali"
Ad(2) = "Veli"
Ad(3) = "Mehmet"
```

Dikkat ederseniz, önceki örnekte 3 ayrı değişkenimiz varken, sonrakinde ise tek değişken olan "Ad" kullanılmıştır. Dizi sayesinde tek değişken ismi ile birden fazla değer saklanabilmektedir.

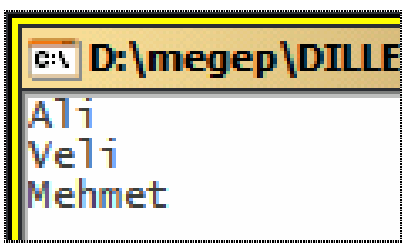


1.1. Dizi Değişkenini Tanımlama

Normalde bir değişkenin bir adı ve bir değeri olabilir. Dizi değişkenlerinin de bir adı vardır, ama içinde aynı türde çok sayıda veri saklanabilir. Tanımlarken dizinin boyutunu belirtmemiz mecburidir. Dizi boyutu tam sayı olarak belirtilmeli, negatif girilmemelidir.

Not: Programlama dillerinde genellikle ilk dizi elemanının indis numarası "0"dır. ***QBasic*** *dilinde "Option Base – Taban Seçeneği" ile varsayılan alt limiti 0 veya 1 olarak değiştirebilirsiniz.*
Dizi elemanlarını tanımlanmadan önce "Option Base 1" olarak seçilmiş ise, 0. elemandan değil 1. elemandan başlanır.

1. Başla
2. Metin Dizi Ad(3)
3. Ad(1) = "Ali"
4. Ad(2) = "Veli"
5. Ad(3) = "Mehmet"
6. Yaz; Ad(1)
7. Yaz; Ad(2)
8. Yaz; Ad(3)
9. Bitir



Resim 1.1: Dizi örneği ve ekran çıktısı

1.2. Dizi Değişkeni ile Veri Saklama

Programdaki dizi değişkenlerinin tanımlandıkları satırlar çalıştırıldığında, ana bellekte dizi boyutunca yer ayrılır. Değişken ile olan işlem bitince ayrılan bellek bölgesi silinir. Belli bir bellek alanı ayrılmasından dolayı dizinin maksimum sınırı dışındaki diğer bellek bölgesine erişemeyiz.

Dizilere değer aktarma veya okuma işlemlerinde döngü komutları kullanılmaktadır. Hangi döngüyü kullanırsanız kullanın, başlangıç ve bitiş değerlerini iyi belirleyiniz, yoksa programınız hata verip kapanır. Sınırın altında veya üstünde indis vermemek gerekir.

1. Başla
2. Metin Dizi Ad(3)
3. Sayısal i, j
4. Döngü i = 1, 3, 1
5. Yaz; i & ". ismi giriniz"
6. Oku; Ad(i)
7. Döngü Bitti
8. Yaz; "Girilen isim listesi"
9. Döngü i = 1, 3, 1
10. Yaz; Ad(i)
11. Döngü Bitti
12. Bitir

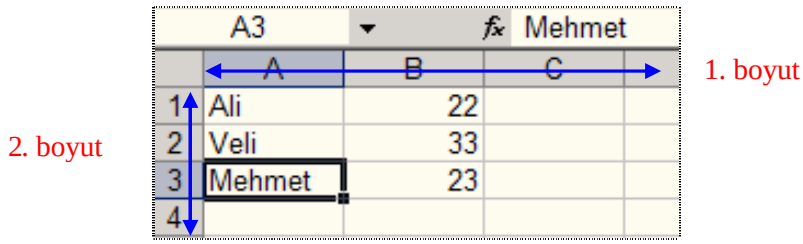
```
 1              . ismi giriniz
Ali
 2              . ismi giriniz
Veli
 3              . ismi giriniz
Mehmet
Girilen İsimlerin listesi:
Ali
Veli
Mehmet
```

Resim 1.2: Döngü ile dizi kullanımı

1.3. Çok Boyutlu Diziler

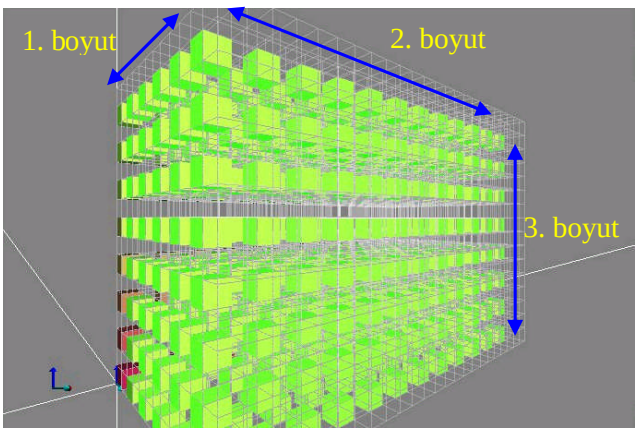
Şimdiye kadar verdiğimiz örnekler tek boyutlu dizi örnekleridir. Bu tip diziler aynen tren vagonları gibi bellekte peş peşe değerlere sahip olan değişkenlerdir.

Tablolama programlarındaki gibi satır ve sütunlu hücrelerden oluşan dizilere; **iki boyutlu** dizi denir. Resim 1.3'te görüldüğü gibi herhangi bir hücrenin değerine satır ve sütun bilgisini kullanarak erişilebilmektedir. Matematikteki **matrisler** de çok boyutlu dizilerdir.

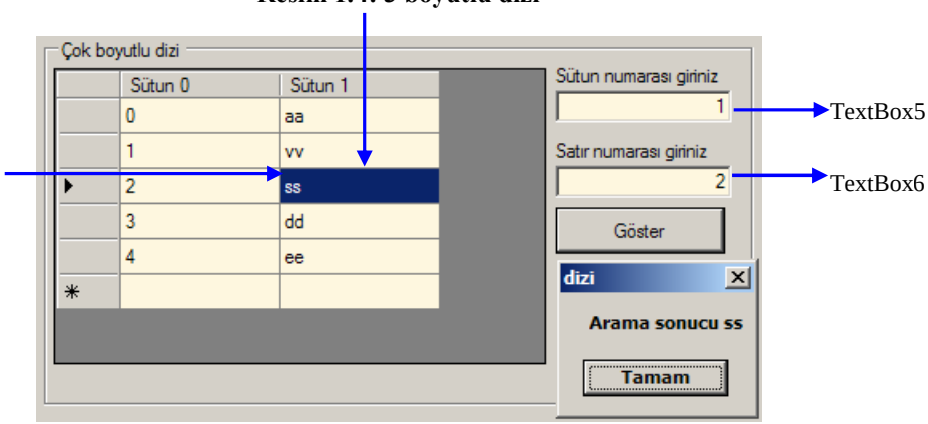


Resim 1.3: Excel'de bir sayfa 2 boyutlu diziye benzetilebilir

Bir boyut daha eklendiğinde küp şekline benzeyen **3 boyutlu** dizi elde edilir. Bu dizilerde satır ve sütun bilgisinin yanında **derinlik** bilgisi de eklenir.



Resim 1.4: 3 boyutlu dizi



Resim 1.5: Çok boyutlu dizi örneği, burada dizi 0 indis numarası ile başlıyor

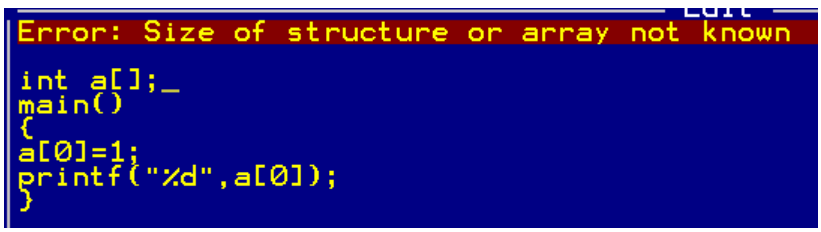
Çok boyutlu dizilerin tanımlanması tek boyutlu diziye benzer şekilde yapılmaktadır:

```
Başla
          Metin Dizi bilgi(2, 6)
          Sayısal i, j, satır, sütun
          Döngü i = 1, 2, 1
                    Döngü j = 1, 6, 1
                              Oku; bilgi(i, j)
                    Döngü Bitti
          Döngü Bitti
          Oku; "Satır ve sütun bilgisini giriniz", satır, sütun
          Yaz; "Aranan bilgi: ", bilgi(satır, sütun)
Bitir
```

Bu örnekte 2 satır ve 6 sütun olduğu için, 2 * 6 yani 12 adet metin bilgisi bulunmaktadır. İki boyut olduğundan iç içe iki döngü hazırladık. İlk döngünün "i" değişkeni satır seçicisi, ikinci iç döngünün "j" değişkeni de sütun seçici olarak çalışıyor. Bilgi(i, j) üzerine veri aktarıp, döngü sonunda hangi satır ve sütun isteniyor ise ekrana değeri getiriliyor.

1.4. Dinamik Diziler

Çoğu dil sadece statik sınırlı dizi imkânı sunar, yani çalışma zamanında dizilerin eleman sayısının sınırını değiştiremezsiniz. Eğer dizi büyük gelmiş ise dizi kısaltılamaz, ya da tam tersi uzatılamaz. Statik dizi içi boş bile olsa hafızada yer kaplamaktadır.

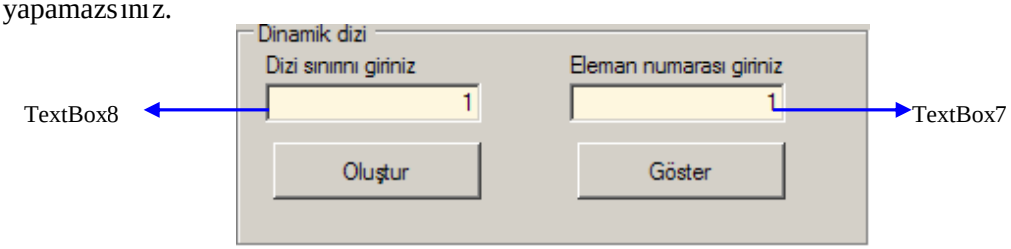


Resim 1.6: C dilinde dinamik dizi yoktur

Hafızayı etkin kullanmak için "**dinamik dizi**" kullanabilirsiniz. İsterseniz diziyi boyutlandırabilirsiniz, ya da silebilirsiniz.

Not: Basic dilinde dinamik diziler için "Redim - Boyutlandır" komutu kullanılır. Bu komut ile eski dizi bilgileri silinir. Dizinin değişken türünü değiştiremezsiniz.

Dizilerde dizi sınırını değiştirebilirsiniz, ama 2 boyutlu bir diziyi 3 boyutlu dizi yapamazsınız.



Resim 1.7: Dinamik dizi örneği

1. Başla
2. Metin Dizi dinamikDizi()
3. Sayısal i, sınır, eleman
4. Oku; "Dizi sınırını giriniz", sınır
5. Boyutlandır dinamikDizi(sınır)
6. Döngü i = 1, sınır, 1
7. Oku; dinamikDizi(i)
8. Döngü Bitti
9. Oku; "Eleman numarasını giriniz", eleman
10. Yaz; dinamikDizi(eleman)
11. Bitir

Dizilerde farklı veri türlerini saklama imkânı yoktur. Yani öğrenci listesinde öğrenci isimleri metin, öğrenci numaraları sayı türünde olsun diyemezsiniz. Tüm elemanların türü aynıdır. Bu eksiklik "**yapı**" kullanımı ile giderilmektedir. Bir yapıda istediğiniz türde değişkenleri beraber tanımlayarak tek isim altında kullanabilirsiniz.

1.5. Yapılar

"Yapısal Programlama Temelleri" modüldeki dosyalama kısmında, yapılar ve kayıtlar hakkında az da olsa giriş bilgisi verilmişti. Sayı ve metin değişkenleri karışık olarak bir yapı (*structure*) içine kaydedebiliyoruz.

```
Ad1   = "Ali"
Notu1 = 45
Ad2   = "Veli"
Notu2 = 55
Ad3   = "Mehmet"
Notu3 = 75
```

Bunun yerine artık şu şekilde yapabiliriz:

```
Ogrenci1.Ad     = "Ali"
Ogrenci1.Notu   = 45
Ogrenci2.Ad     = "Veli"
Ogrenci2.Notu   = 55
Ogrenci3.Ad     = "Mehmet"
Ogrenci3.Notu   = 75
```

Görüldüğü gibi farklı türlerde birbiri ile ilişkili yapı elemanları "Ad ve Notu" değişkenleri beraber, tek bir isim olan "Ogrenci" yapısında toplanıyor.

```
Yapı Kisi
     Metin Ad
     Sayısal Yas
Yapı Bitti
birKisi Kisi
```

Dosyalama konusunda "Kisi" değişken isminde yapı hazırlamıştık. Programcının kendi yaptığı değişkenlere "**kullanıcı tanımlı veri türü – *user defined variable***" de denir.

Yapının Genel Kullanımı

```
KayitDegiskeni.Degisken = "Veri"

Degisken = KayitDegiskeni.Degisken
```

1.6. Yapıları Dizi Olarak Kullanmak

Dizi olarak kullanımda sadece yapıyı kullanacak değişkenin yazımı değişmektedir. Yani yapı kısmı aynı şekilde tanımlanırken değişken kısmı değişir.

```
Başla
            Yapı birÖğrenci
                        Metin Ad
                        Sayısal Notu
            Yapı Bitti

            birÖğrenci Dizi Öğrenci(3)
            Sayısal i

            Döngü i = 1, 3, 1
                        Oku; "Öğrenci adını giriniz ", Öğrenci(i).Ad
                        Oku; "Öğrenci notunu giriniz ", Öğrenci(i).Notu
            Döngü Bitti
            Döngü i = 1, 3, 1
                        Yaz; "Öğrenci adı ", Öğrenci(i).Ad
                        Yaz; "Öğrenci notu ", Öğrenci(i).Notu
            Döngü Bitti
Bitir
```

UYGULAMA FAALİYETİ

İşlem Basamakları	Öneriler
1. Bir dizi oluşturunuz.	İsimlendirmede değişken isimlendirme kurallarına uyunuz.
2. Dizi elemanına değer aktarınız.	Dizilerin ilk indis numarası genellikle 0 veya 1 ile başlar. Değer aktardıktan sonra ekranda değerlerini listeleyiniz.
3. Döngü içinde diziye veri giriniz ve gösteriniz.	Genellikle "Döngü – For" döngüsü ile veri giriş çıkışı yapılabilir. Döngünün değişkeni dizinin ilerlemesi için kullanılır.
4. Bir yapı oluşturunuz.	Çeşitli veri türlerine sahip değişkenleri yapı içine tanımlayınız. Program içinde tanımladığınız yapıyı kullanınız.
5. Yapı değişkeni ile verileri kontrol ediniz.	Yapı dizi olarak tanımlanıp, döngü içinde veri girişi ve ekrana listeleme yapılabilir.

ÖLÇME VE DEĞERLENDİRME

A- OBJEKTİF TESTLER (ÖLÇME SORULARI)

Aşağıdaki sorulardan; sonunda parantez olanlar doğru / yanlış sorularıdır. Verilen ifadeye göre parantez içine doğru ise "D", yanlış ise "Y" yazınız. Şıklı sorularda uygun şıkkı işaretleyiniz.

1. ( ) Ayrı ayrı değişken tanımlamak, **diziden** daha az bellek alanı işgal eder.

2. ( ) Dizi değişkenlerinin indis numaraları sadece tam sayı olabilir.

3. ( ) Dizi içinde ondalıklı değerler saklayabiliriz.

4. ( ) Dizi içindeki veriler otomatik olarak sıralanır.

5. ( ) Hangisini dizi olarak tanımlayamayız?
A) Yapı
B) Metin
C) Karakter
D) Döngü

6. ( ) Genellikle dizi indisi hangi karakterler arasına yazılır?
A) ( )
B) [ ]
C) { }
D) { ( ) }

ÖĞRENME FAALİYETİ–2

AMAÇ

Bu öğrenme faaliyetinde işaretçiler ve işaretçi yapıları ile program yazabileceksiniz.

ARAŞTIRMA

Bu faaliyet öncesinde hazırlık amaçlı aşağıda belirtilen araştırma faaliyetlerini yapmalısınız.

- Hangi dillerde bağlı liste veya işaretçi (*pointer*) kullanımı vardır? İşaretçi kullanımının avantajları ve dezavantajlarını araştırınız.
- Posta görevlisi mektupları adresine nasıl ulaştırıyor? Bir mektup gönderenden alıcıya giderken hangi aşamalardan geçer, araştırınız.

2. İŞARETÇİLER

Bir dizi yaptığınızda boyutunu belirtmek zorunda kalırsınız. Eğer dizi çok küçük ise, bilgilerinizi saklamak için yeterli olmaz, ya da diziniz çok büyükse bilgisayarın değerli ana belleğini boşuna meşgul edersiniz. Dizilerde başka bir problem de içeriğin kolayca tekrar düzenlenememesidir. Yani dizi içeriğini alfabetik olarak sıralamak isterseniz, tüm bilgileri dışarı alıp, düzenleyip, içine tekrar atmak zorunda kalırsınız.

İşaretçiler ve bağlı listeler sayesinde dizinizin uzunluğunu esnek olarak kullanabilirsiniz.

2.1. İşaretçi Nedir?

Dizilerdeki bilgiler peş peşe kutulara benzetilebilir. Bağlı listeler ise birbirine bağlanmış ayrı kutular olarak düşünebiliriz. İşaretçiler birbiri ile ilgili olan, düğüm halindeki bilgileri birbirine bağlar.

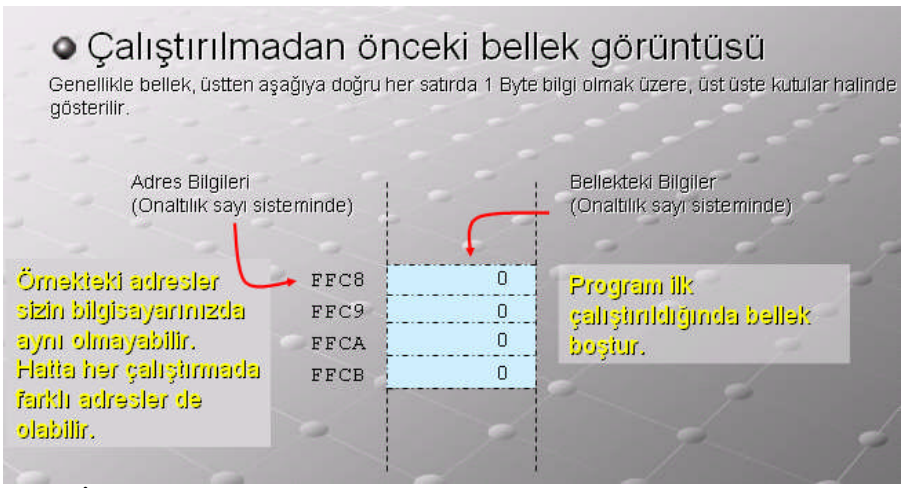
? Tupol dilinde işaretçi, yapı ve dizi örneklerini inceleyiniz.

İşaretçi değişkenler sadece "**bellek adresi**" bilgisini saklayan değişkenlerdir. Diğer değişkenler ise metin veya sayı olabilirler. İstenilen değeri hafızadan çağırmak için adresini bilmeniz yeterlidir.

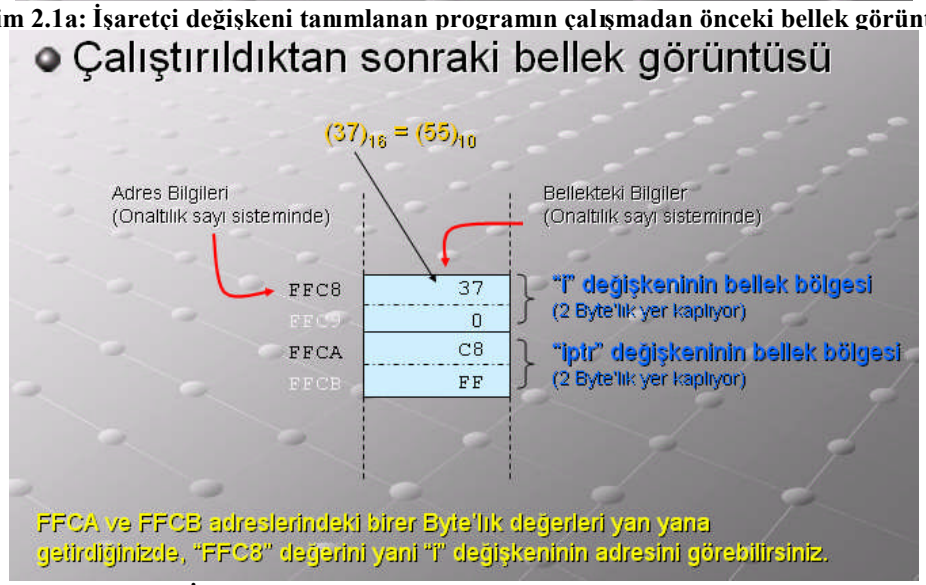
İşaretçiler C ve C++ programlama dillerinin temel bölümüdür. Yanlış bellek bölgesine ulaşmak, kötü bir şekilde programın çökmesi ile sonuçlanabilir. Bunu beyninize rastgele batırılan sivri bir iğne gibi düşünebilirsiniz. İşaretçileri hatalı kullanmak bilgisayarın belleğini rastgele kurcalamak anlamına gelir.

C dilinde işaretçi örneği

```
#include <stdio.h>
main(){
int i;      /* "i" tamsayı değişkenlerdir */
int *iptr;  /* "*" işaretçi olduğunu belirtir */
iptr  = &i;
/* işaretçiye muhakkak bir değişken adresi atanır, "&" adres simgesidir */
*iptr = 55;
/* aslında işaretçinin işaret ettiği değişkene 55 değeri atandı */

clrscr();
printf("%d\n",i);       /* ekrana 55 yazar */
printf("%d\n",*iptr);   /* 55 "i" değişkeninin değeri*/
printf("%X\n",iptr);    /* FFC8 "i" değişkeninin hafızadaki adresi*/
printf("%X\n",&iptr);   /* FFCA işaretçinin adresi*/
printf("%X\n",&i);      /* FFC8 "i" değişkeninin hafızadaki adresi*/
getch();
}
```



Resim 2.1a: İşaretçi değişkeni tanımlanan programın çalışmadan önceki bellek görüntüsü

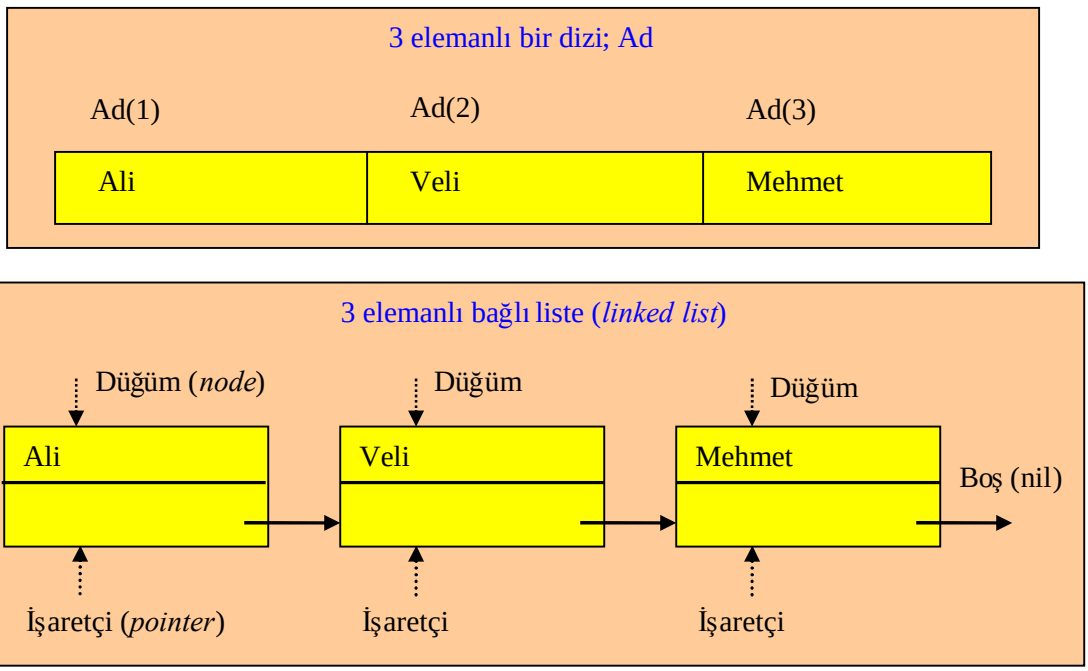


Resim 2.1b: İşaretçi değişkeni ve normal bir değişkenin bellek görüntüsü

2.2. Bağlı Liste Tanımlama

Bağlı listelerde (*linked list*) Şekil 2.1'de görüldüğü gibi her eleman birbirine işaretçiler ile bağlıdır. İşaretçinin en son gösterdiği ise "nil veya null" adı verilen boş bir değerdir. "Nil" liste sonunu belirtir.

Asıl verileri yer değiştirerek düzenlemek yerine, işaretçiler tekrar düzenlenerek yer değiştirme işlemi hızlı bir şekilde yapılır.



Şekil 2.1: Diziler ve bağlı listeler

Pascal dilinde bağlı listeleri aşağıdaki şekilde kullanabiliriz.

- Yeni bir işaretçi tanımlayalım:

```
TYPE
    isaretciAdi= ^KayitTuru;
```

- Hemen tanımlamanın altında da yapımızı "KayitTuru" tanımlayalım:

```
KayitTuru=  RECORD
Adi:        String[15];
Notu:       integer;
Sonraki:    isaretciAdi;
END;
```

Alt alta yazılan bu satırlar ile yeni bir yapı ve onun adresini saklayacak bir işaretçimiz oldu. “Sonraki” değişkeni diğer düğümün adresini saklayacaktır.

- Son olarak bu kayıt yapısını belirten bir değişken tanımlayalım:

```
VAR
   Dugum: isaretciAdi;
```

- Kodları bir araya getirelim:

Pascal dilinde bağlı liste örneği

```
PROGRAM bagliListeler;

TYPE
    isaretciAdi= ^KayitTuru;    {"KayitTuru" yapısının işaretcisi}

    KayitTuru= RECORD           {yapı veya veri kümesi}

    Adi:       String [15];
    Notu:      integer;
    Sonraki:   isaretciAdi;     {sonraki kayıt}

    END;

VAR
   Dugum: isaretciAdi;
```

Artık programımızın “kurulum” kısmı tamamlanmıştır.

- Yaptığımız değişkenlerden sonra, ana programın kod yazımında ise:

  - Bir düğüm oluşturalım.
  - Düğüme veri aktaralım
  - Düğümün işaretçisini düzenleyelim (başlangıç, orta, bitiş olarak)

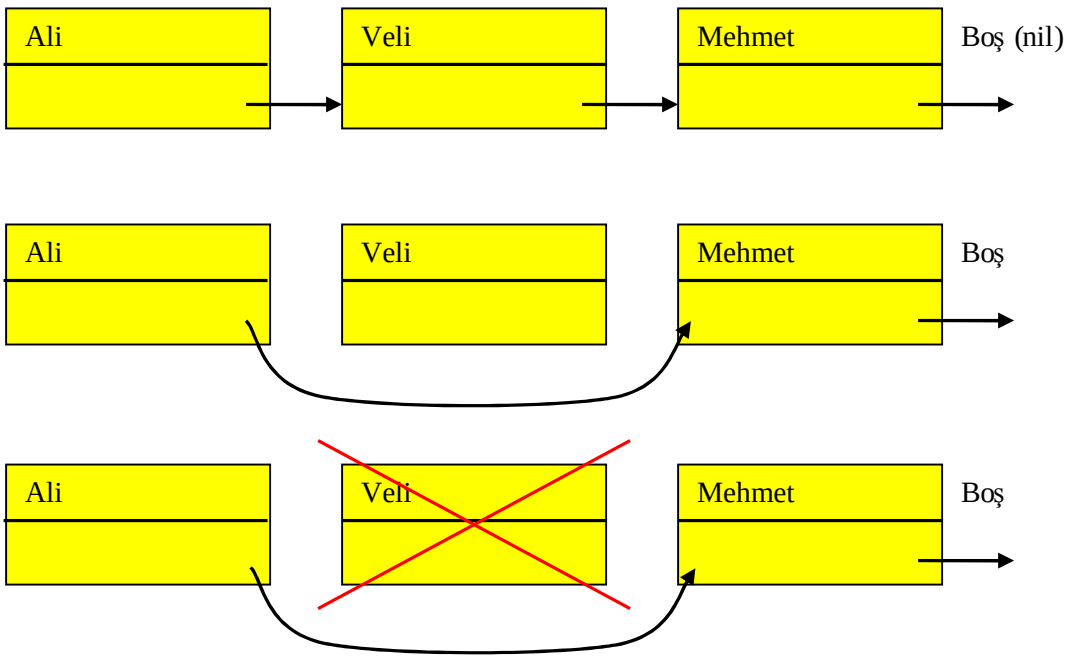
```
BEGIN
    New(Dugum);                     {Yeni düğüm oluşturduk}
    Dugum^.Adi := 'Ali Can';        {Veri aktardık}
    Dugum^.Notu:= 45;
    Dugum^.Sonraki:= nil;           {Bağlı Listeyi sonlandırıyoruz}

    writeln('Adı: ',      Dugum^.Adi);
    writeln('Notu: ',     Dugum^.Notu);
    readln;
    Release(Dugum);                 {düğümü sildik}
END.
```

2.3. Bağlı Listeyi Kullanma

Bir dizi içindeki bir elemanı sildiğinizde, hala bellekte yer kaplayan bir boş alan oluşur. Ayrıca işe yaramayan bir “boşluk”, programda istenmeyen hatalara neden olabilir.

Bağlı listelerde ise düğüm silmek çok kolaydır:

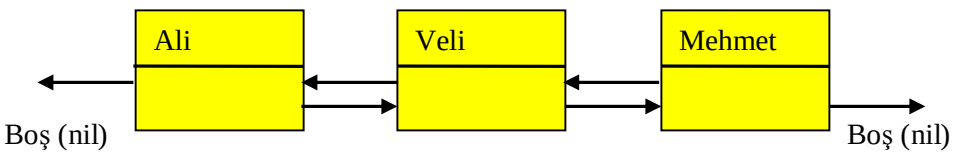
- Düğümlerdeki işaretçileri düzenleyin
- Düğümü silin



Tek bağlı listelerin dezavantajı ilk kaydı bulmanın mümkün olmamasıdır. Yani siz geri yönde gidemezsiniz. Hep sonraki kaydın bağını sakladığımız için bu mümkün değildir. "Ali" düğümü, "Veli" düğümünü gösteriyor, ama "Veli" düğümünün önceki düğüm ile ilgili hiçbir ipucu yoktur.

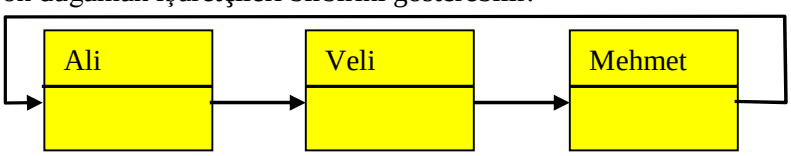
2.4. Çift Bağlı Listeler

İki işaretçi kullanarak önceki ve sonraki düğümlerin adres bilgilerini tutabilirsiniz.



2.5. Dairesel Bağlı Listeler

İlk ve son düğümün işaretçileri birbirini gösterebilir.



2.6. Yığın

Yığını (*stack*), özel tek yönlü bağlı listelere benzetebiliriz. Ekleme ve silme işlemlerini sadece listenin en başındakiler üzerinde yapabilirsiniz. Üst üste tabaklar gibi düşünebilirsiniz. Alttaki tabaklara ulaşmak için mecburen üstteki tabakları kaldırmak zorundasınızdır. Yeni bir tabak gelince de yığının üstüne koyarsınız.

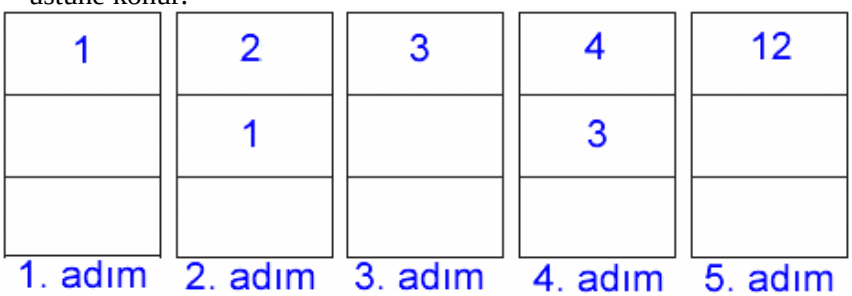
"Ters Polonyalı Yazımı – *Reverse Polish Notation* RPN" yöntemi bunu kullanır:

Formül:
```
(1 + 2) * 4
```

Aynı formülün RPN hali:
```
1 2 + 4 *
```

Normalde 1 ve 2 toplar, sonucu 4 ile çarparız. Sonuç 12 olur. RPN yönteminde ise adım olarak şöyle yapılır:

- 1 rakamı yığının en üstüne konur.
- 2 rakamı yığının üstüne konur, 1 alta iner.
- 2 ve 1 rakamı yığından çekilerek toplama işlemi yapılır, sonuç olan 3 yığının en üstüne konur.
- 4 rakamı yığının en üstüne konur, 3 alta iner.
- 4 ve 3 rakamı yığından çekilerek çarpma işlemi yapılır, sonuç olan 12 yığının en üstüne konur.



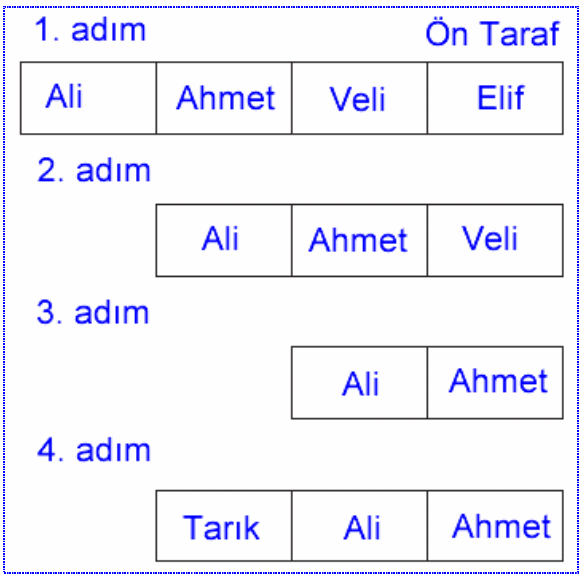
Resim 2.2: RPN yöntemi

Yığınlar genellikle **LIFO** (*Last in First out* – Son giren İlk çıkar) yöntemini temel alır.

2.7. Kuyruk

Kuyruk (*queue*) listelerinin iki kuralı vardır. Birincisi yeni eklenen bilgiler sadece sona eklenebilir, ikincisi de silinecek bilgi sadece baştan silinebilir. **FIFO** (*First in First out* – İlk giren ilk çıkar), yöntemi de denilmektedir.

Sinema kuyruğundaki insanları düşünün, ilk giren ilk kuyruktan çıkar. Son gelen ise son çıkar. Kuyruğa girmek isterseniz en sona gitmeniz gerekir, biletini alan müşteri ise en başta olduğundan kuyruktan ayrılır.



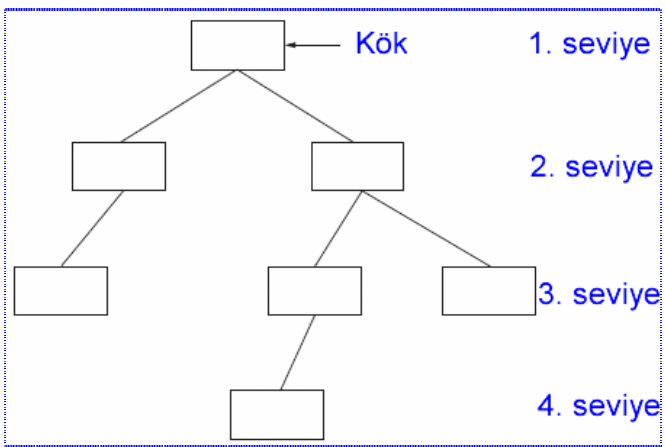
Resim 2.3: Kuyruk yöntemi

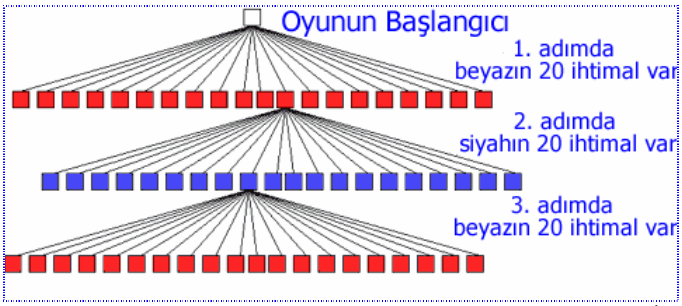
2.8. Ağaç

"Ağaç – *Tree*" listeleri doğrusal veya dairesel şekilde gitmezler. Kök düğüm üzerinde dalların bulunduğu bir listedir.

Bazı programcılar bu yönteme "**ikilik ağaç** – *binary tree*" demektedir. Bir düğüm; boşluğa, başka bir düğüme veya iki düğüme işaret edebilir.

Genellikle yapay zekâ programlarında kullanılır. Mesela bir satranç oyunu olabilir. İlk hamle en üstteki köktür. Yapılabilecek her hamlenin ihtimalleri dallara ayrılır. Karşı oyuncunun hareketine göre de ihtimaller değişerek dallar oluşur. En son dal "şah – mat" ile biter.

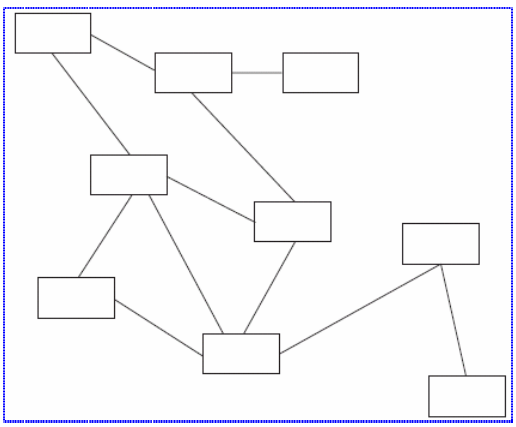




Resim 2.4: Ağaç listesi ve satranç oyunundaki dallanma*

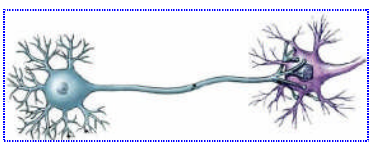
2.9. Grafik

Belli bir şekli olmayan bağlı listelerdir. Bir düğüm bir veya daha fazla düğüme işaret edebilir.



Resim 2.5: Grafik yöntemi

Genellikle "sinir ağları – *neural network*" kurulumunda kullanılır. Beyin de bu yöntemle işlem yapmaktadır. Her düğüm (*neuron*), siniri (*synapses*) temsil eder. Programınız karmaşıklaştıkça bu gelişmiş yöntemleri keşfederek algoritmalarınızı yeniden şekillendireceksiniz.



* Toplam 10 ^ 120 ihtimal vardır. Evrendeki tahmini atom sayısı 10 ^ 75 olduğuna göre bu rakam ile satranç ihtimallerini karşılaştırınız.

Filmlerdeki bilgisayarlarda...

- Uzun cümle ve komutlar yazarken boşluk tuşunu kullanmaya hiç gerek olmaz...
- Editörlerde imleç hiç görünmez...
- Tüm karakterler en az 3 cm büyüklüğündedir...
- Karakterler ekrana en fazla 110 baud hızında listelenir...
- NASA, CIA, KGB gibi örgütlerin bilgisayarlarında son derece gelişmiş kullanıcı arabirimleri vardır. Örneğin *"Bana tüm gizli dosyaları göster"* demeniz, aradığınız dosyayı ekranda görmeniz için yeterlidir...
- Virüs bulaştırmak çok kolaydır: "UPLOAD VIRUS" gibi bir komut yazmanız yeterlidir...
- Herhangi bir bilgisayardan dünyadaki herhangi bir bilgisayara erişebilirsiniz; kapalı olsa bile...
- Ekranlarındaki görüntü o kadar parlaktır ki, ekrandaki yazılar kullanıcının yüzüne yansır; seyirci her şeyi oyuncunun yüzünden okuyabilir...
- Şifreler en fazla üç denemede bulunabilir...
- Her türlü erişim koruması "OVERRIDE PROTECTION" falan gibi bir komutla aşılabilir...
- Tüm bilgisayarlar her türlü disketi okur; ters takılsa bile...
- Herhangi bir program, herhangi bir bilgisayarda çalışır...
- Ne kadar küçük olursa olsun; amacı ne olursa olsun bütün bilgisayarlar ekranlarında üç boyutlu nesneler görüntüleme ve bunları döndürme yeteneğine sahiptir...
- Portatif bilgisayarların tümü CRAY kadar hızlıdır ve gerçek zaman video/ses yetenekleriyle donatılmıştır.
- Tüm bilgisayar arızaları kendini bol dumanlı patlamalarla belli eder...

UYGULAMA FAALİYETİ

İşlem Basamakları	Öneriler
1. Yapısal veri tanımlanabilen bir dilde "işaretçi" oluşturunuz.	Her dil işaretçi imkânı sağlamaz. Uygun dili bulmanız gereklidir.
2. İşaretçiyi bir değişkene bağlayınız.	Bir değişken tanımlayıp, ona ait işaretçi değişken tanımlayınız. İşaretçiyi değişkenin adresine bağlayınız. Doğrudan işaretçiyi kullanarak değer atamaları yapınız.
3. Kayıt yapısında bir değişken tanımlayınız.	Bir yapı tanımlayıp, işaretçi ile yapının adresini saklayınız. Değer atayıp, ekrana değeri listeleyiniz.

ÖLÇME VE DEĞERLENDİRME

OBJEKTİF TESTLER (ÖLÇME SORULARI)

Aşağıdaki sorulardan; sonunda parantez olanlar doğru / yanlış sorularıdır. Verilen ifadeye göre parantez içine doğru ise "D", yanlış ise "Y" yazınız. Şıklı sorularda uygun şıkkı işaretleyiniz.

1. İşaretçi tanımlandığında muhakkak bir değişkene bağlanmalıdır. ( )

2. Düğüm (*record*) içinde sadece sayı bilgisi saklayabiliriz. ( )

3. Kayıt ve yapı aynı kavramlardır. ( )

4. Listeler doğrusal bir şekilde birbirine bağlı olarak yapılırlar. ( )

5. Her listenin sonu "nil - null" ile belirtilmelidir. ( )

6. Hangisi bir veri listesi değildir?
A) Bağlı
B) Çift bağlı
C) Grafiksel
D) Yığın

"Bu ülkenin bir çok aklı başında ve konusunda bilgili adamıyla konuştum... Sizi temin ederim ki bu 'Bilişim' denen sözde bilim dalının balonu en geç yıl sonuna kadar sönecek..."

- Prentice Hall yayınevinin işletme kitapları baş editörü, 1957.

ÖĞRENME FAALİYETİ–3

AMAÇ

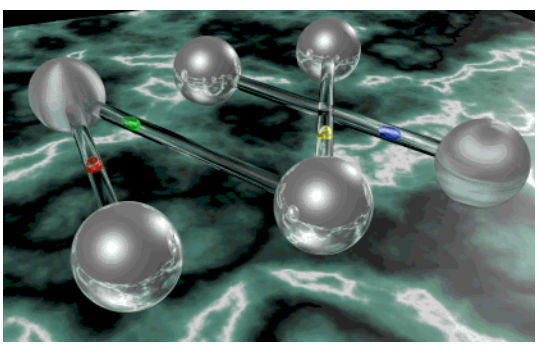
Nesneye yönelik programlama yapabileceksiniz.

ARAŞTIRMA

Bu faaliyet öncesinde hazırlık amaçlı aşağıda belirtilen araştırma faaliyetlerini yapmalısınız.

- Klasik programlama dilleri ile nesne tabanlı programlama dillerinin farkları nelerdir?
- Yapay zekâ ile ilgili programları inceleyiniz. Mesela bilgisayar ile sohbet programları ve oyunlar gibi.

3. NESNE TABANLI PROGRAMLAMA



Yapılan programları rahat anlaşılır, hızlı ve görsel nitelikleri zenginleştirilmiş duruma getirmek için nesne tabanlı programlama dilleri kullanılabilir. Nesne tabalı programlama dilleri ile güvenilir ve hatadan uzak programlama yapılabilir. Yine de **OOP** (*Object Oriented Programming* – nesne tabanlı programlama) tek başına bir programı daha okunaklı hale getiremez.

QBasic ve Visual Basic 6 nesne tabanlı programlama dilleri değildir. C#, C++ ve Java nesne tabanlı dillere örnektir. Hangi dilin OOP olduğu tartışıladursun, biz şimdilik **C++** ile ilk denemelerimizi yapacağız.

3.1. Kolay Programlama Yöntemi

Bilgisayarınızın ne kadar güçlü olduğu önemli değildir, onun sınırlarını yazılım kontrol eder. Yazılımların en büyük problemi "güvenilirlik" konusudur. Program sık sık göçmemeli, hata vermemeli ve garip davranmamalıdır. Daha önceki derslerde bunun öneminden yaşanmış örnekler ile bahsetmiştik.

Güvenilir bir program planlanan zamanda bitmelidir. Tam test etmeden piyasaya sürmek kötü sonuçlar verebilir.

Eski zamanlarda programlar küçük olduğu için pek plan ve organizasyon yapmaya gerek duyulmazdı. Çalışmayan programı programcı tekrar yazıyordu. Küçük programlarda "**dene ve hata bul**" işlemi pek zor değildir. Büyük programlarda böceklerin daha fazla saklanacak yerleri vardır. Milyonlarca satır programda "dene ve hata bul" pek işe yaramaz. Bu nedenle örneğin bir mimar gökdeleni plansız olarak inşa edemez.

Önceki derslerde de anlatıldığı gibi, büyük programları alt programlara bölerek kolay programlanır hâle getirmiştik. Ana program yayımlanmadan önce alt program kodları kopyala yapıştır ile birleştirilir ve derlenir. Burada siz de bir şüphe duyuyor musunuz? Mesela çalışma zamanında alt programlar diğer alt programlardan etkileniyor olabilir mi?

OOP'de temel olan konu da alt modüllerin diğer modüllere müdahale etmemesidir. Yani aralarında yalıtım vardır. Modüllere "***object* – nesne**" de denir.

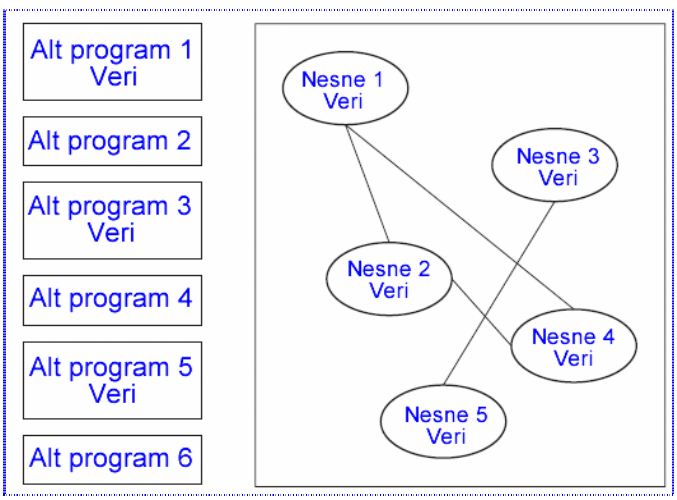
Bir nesnenin iki temel özelliği vardır:

- **Veri**: Özellikler de denir.
- **Komutlar**: Metod da denir. Veriyi kontrol eden kodlardır.

Nesne diğer kodlara müdahale etmeyeceği için kazancımız şunlar olur:

- **Güvenilirlik**: Program çalışmadığında, böcekli nesne alınır ve düzeltilir, tüm programın elden geçirilmesine gerek kalmaz.
- **Tekrar kullanılabilme**: Teorik olarak nesneleri alıp başka program içinde rahatlıkla kullanabilirsiniz. Böylece yeni program yazarken daha hızlı işlerinizi bitirebilirsiniz.

Bu kazançları "***inheritance* – miras alma, kalıt alma, devralma**" sayesinde elde ederiz. Var olan kodu tekrar kullanabilmek, programcının daha fazla ilerlemesini sağlar.

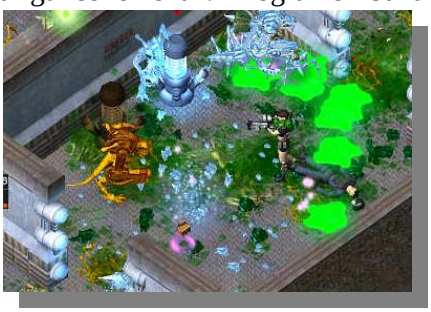


Resim 3.1: Klasik programlama ve nesne tabalı programlama

Klasik programlamada alt program diğer alt programın verisine ulaşabilir. Nesne tabanlı programlamada ise nesneler verilerini korurlar. Bu korumaya "***encapsulation* – kuşatma**" denir.

3.2. Nesne Kullanımı

Programlamada en büyük sorun, var olan kodu güncellemektir. Programcı buna yıllarını verir. Zaten çalışan bir programı güncellemek yepyeni program yazmaktan kolaydır. Nesne tabanlı programlamada, sadece güncellenmesi gerekli nesne üzerine çalışıldığından programcının işi kolaylaşır.Nesne tabanlı programlama bu sebeple çok tutulmuştur.



Uzaylıları vurduğuz bir oyunda, ekrana gelen uzaylıların görünümünü değiştirmek için yapmanız gereken; uzaylı nesnesini alıp, değiştirip, yeniden eski yerine koymaktır. Sadece şekil değil, hareketlerini de bu şekilde güncelleyebilirsiniz.

Nesnedeki Veriyi Saklama veya Gösterme

Nesneler de birbirleri ile veri alışverişinde bulunabilir. "*private*, *public* ve *protected*" bu işlem için yapılmıştır.

- **Private (özel)**: Nesne verisini ve kodunu paylaşıma açmaz.
- **Public (paylaşıma açık)**: Herkes nesnenin verisini ve kodunu kullanabilir.
- **Protected (korumalı)**: Eğer varolan bir nesne miras alma yöntemiyle kopyalanıp yeni bir nesne oluşturulursa, yeni nesne sadece paylaşıma açık (public), korumalı (protected) veri ve komutları miras alır. Özel (Private) veri ve komutlar eski nesnede kalır, yeni nesneye aktarılmaz.

Yeni bir nesne oluşturmanın ilk adımı "***class* – sınıf**" oluşturmaktır. Sınıf daha önceki derste gördüğümüz yapıya benzer. Bir sınıf, veriyi kullanmak için veri ve kod tanımlamalarını tanımlar. Bir sınıf nesne değildir. Nesneyi bir değişken gibi tanımlayıp, sınıftan oluşturursunuz.

- Bir sınıftan nesneler türetebilirsiniz:

```
class canavar
{
public:
int x_koordinati;
int y_koordinati;
void hareketEt(int, int);
void ilklendirme(int, int);
};
```

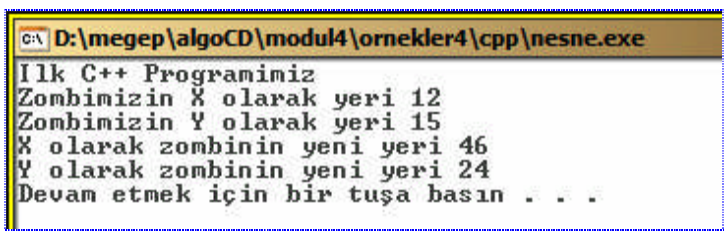
İşte ilk sınıfımızı yazdık. Gördüğünüz gibi sınıfımızın "x_koordinati ve y_koordinati" olarak tam sayı türünde 2 özelliği var. "hareketEt ve ilklendirme" olarak da 2 metodumuz var.

- Bu hazırlıktan sonra, yapmamız gereken metotların kodlarını hazırlamaktır:

```
void canavar::hareketEt(int yeni_x, int yeni_y)
{
x_koordinati = x_koordinati + yeni_x;
y_koordinati = y_koordinati + yeni_y;
}
void canavar::ilklendirme(int ilk_x, int ilk_y)
{
x_koordinati = ilk_x;
y_koordinati = ilk_y;
}
```

- Son olarak da ana programımızı yazıyoruz:

```
int main(int argc, char *argv[])
{
canavar zombi; //'canavar' sınıfından 'zombi' nesnesi üretiyoruz
zombi.ilklendirme(12, 15);

printf("Ilk C++ Programimiz\n");
cout << "Zombimizin X olarak yeri " << zombi.x_koordinati << "\n";
cout << "Zombimizin Y olarak yeri " << zombi.y_koordinati << "\n";

zombi.hareketEt (34, 9);

cout << "X olarak zombinin yeni yeri " << zombi.x_koordinati << "\n";
cout << "Y olarak zombinin yeni yeri " << zombi.y_koordinati << "\n";

system("PAUSE");
return 0;
}
```



Resim 3.2: C++ örneğimizin çıktısı

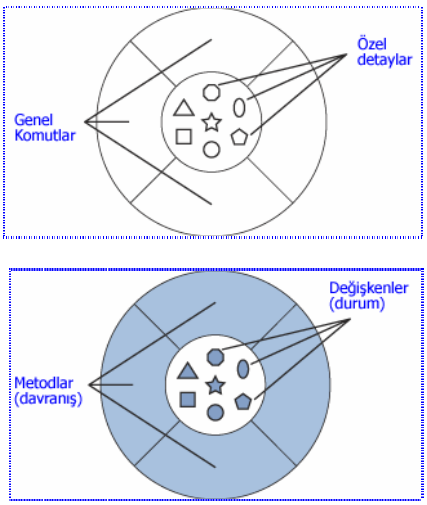
Genel Olarak OOP Deyimleri

Bu özet sizi profesyonel OOP'ci yapmasa da, bu temel kelimeleri anlamaya çalışalım:

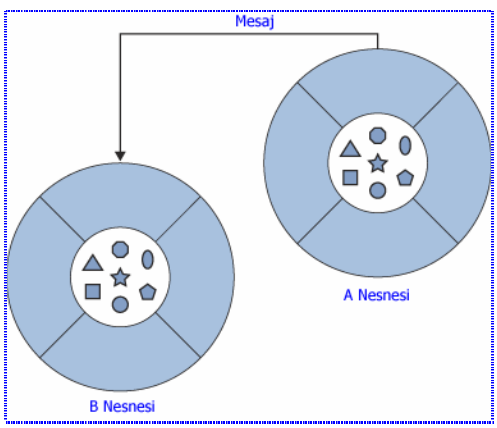
- **Encapsulation (kuşatma)**: İlgili veri ve kodları tek bir yerde toplamak
- **Inheritance (miras alma)**: Bir nesneden diğerine veri ve kod geçirmek
- **Method (olay, komut)**: Nesnenin verilerini manipüle eder
- **Object (nesne)**: Bir ünitede gruplanmış veri ve komut koleksiyonudur

Nesnenin genellikle "***initialize* – ilkleme**" olayı olmalıdır. Varsayılan olarak ilk kez nasıl oluşturulacağını belirtiriz.

Aslında bu yapılanlar klasik programlamadaki alt programlara benziyor. Ama şimdiye kadar hiç bir yapıya olay eklememiştik ve bu kadar gelişmiş değişken tanımlaması gerekmemişti.



Resim 3.3: Sınıf (*class*) ve nesne (*object*)



Resim 3.4: Nesnelerin mesajlaşması

3.3. Dil Seçimi

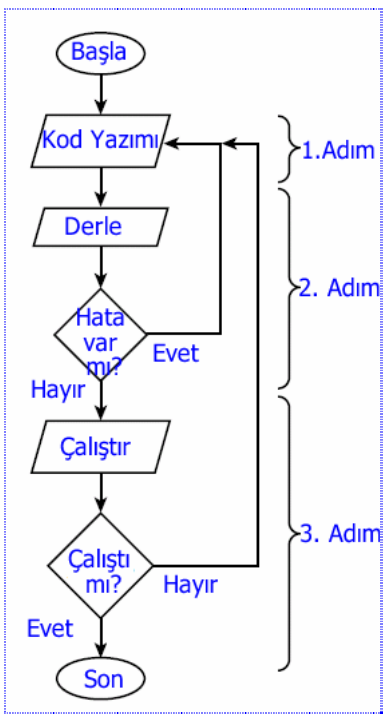
Nesne tabanlı programlama dilleri ana olarak şu şekilde gruplandırılırlar:

- **Karma (hybrid) nesne tabanlı programlama dili**: Eski bir dil üzerine OOP özellikleri eklenmiş ise dil karma olur. Mesela Pascal üzerine yapılmış Delphi, C üzerinde C++ gibi…
- **Saf (pure) nesne tabanlı programlama dili**: SmallTalk, Eiffel, C# ve Java gibi doğrudan OOP özelliklerine sahip olarak yazılmış diller.

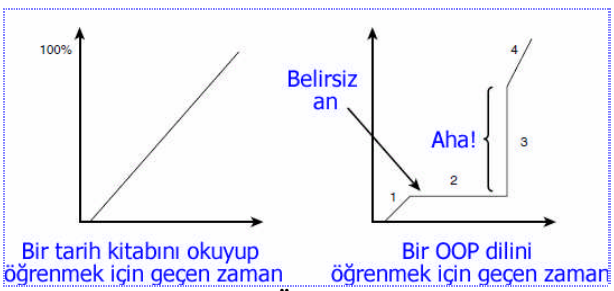
Karma bir dil eski dili bilen birinin, nesne tabanlı dile geçişini kolaylaştırmak için yapılmıştır. En az eğitim ve deneyim ile OOP'ye geçişi sağlar. Programın küçük bir kısmını nesne tabanlı, kalan büyük kısmını eski klasik programlama yöntemleri ile yazabilirsiniz.

Aslında en iyisi doğrudan saf bir OOP dili ile programlamaya başlamaktır. Böylece programlarımız karmaşık ve düzensiz olmaz. Tabii tek başına OOP ile düzenli ve kolay okunur program yazamazsınız. Yazılım tekniğinizi, problem çözüm yöntemlerinizi geliştirmelisiniz.

Yazılımı kullanacak olanlar, sizin gece geç saatlere kadar çalıştığınızı, hangi programlama dilini kullandığınızı umursamaz. Zamanında biten, çalışır halde olan program satılır, rağbet görür. Ondan sonra da artık siz patron olursunuz, kravatınızı takar, rakibinizi geçmek için politikalar geliştirirsiniz.



Resim 3.5: Program yapım aşamaları



Resim 3.6: Öğrenme grafiği

UYGULAMA FAALİYETİ

İşlem Basamakları	Öneriler
1. Sınıf tanımlamak için uygun bir dil seçiniz	Karma veya saf nesne tabanlı programlama dillerini karşılaştırınız.
2. Bir sınıf tanımlayınız	"canavar" örneğindeki gibi siz de sınıf oluşturabilirsiniz. Mesela taşıtlar için "tasit" sınıfı gibi. "canavar" örneğindeki gibi sınıfa "hareketEt" gibi metod ekleyiniz. "tasit" nesnesinden de örneğin araba, kamyon nesneleri yapılabilir.
3. Sınıfa ait olaylar oluşturunuz	
4. Sınıftan nesne oluşturunuz	
5. Nesnenin ilklenme (*initialize*) ve diğer olaylarını yazınız	Program içinde nesnenin olaylarını kontrol ediniz.

ÖLÇME VE DEĞERLENDİRME

A- OBJEKTİF TESTLER (ÖLÇME SORULARI)

Aşağıdaki sorulardan; sonunda parantez olanlar doğru / yanlış sorularıdır. Verilen ifadeye göre parantez içine doğru ise "D", yanlış ise "Y" yazınız. Şıklı sorularda uygun şıkkı işaretleyiniz.

1. Nesne tabanlı programlamada, klasik programlamadaki döngü ve dallanma gibi komutlar yoktur. ( )

2. Her nesne oluşturulur oluşturulmaz, hemen ilklendirilmelidir. ( )

3. Programımızı nesnelere böldüğümüzde, dünyadaki tüm programcılar bizim programımızı anlayabilirler. ( )

4. "***Encapsulation*** **–kuşatma**" nesnenin diğer nesnelerin komutlarına erişememesidir.( )

5. Aşağıdakilerden hangisi ile nesnenin değişken değerlerini korumayız?
A) private - özel
B) void - boş
C) protected - korumalı
D) public – paylaşıma açık

6. Nesne tabanlı programlamada hangisi nesne ile ilgili değildir?
A) Miras alma
B) Kuşatma
C) Alt program
D) Metod

MODÜL DEĞERLENDİRME

PERFORMANS TESTİ (YETERLİK ÖLÇME)

Modül ile kazandığınız yeterliği, öğretmeniniz işlem basamaklarına göre 0 ile 10 puan arasında olacak şeklinde değerlendirecektir.

DEĞERLENDİRME KRİTERLERİ	Puan
Tek isim vererek, birçok veriye dizi kullarak ulaşma	
Bir dizi elemanına değer aktarma	
Döngü içinde diziye veri girme ve gösterme	
Çeşitli veri türlerine sahip değişkenleri yapı ile kontrol etme	
Yapısal veri tanımlanabilen bir dilde işaretçi oluşturma	
Kayıt yapısında bir değişken tanımlama	
İşaretçiyi bir değişkene bağlama	
Sınıf tanımlamak için uygun bir dil seçme	
Bir sınıfı tanımlayan nesne yazma	
Nesnenin ilklenme olayını yazma	
Toplam (en fazla 100 puan olabilir)	

DEĞERLENDİRME

Yaptığınız değerlendirme sonucunda eksikleriniz varsa öğrenme faaliyetlerini tekrarlayınız.

Modülü tamamladınız, tebrik ederiz. Öğretmeniniz size çeşitli ölçme araçları uygulayacaktır. Öğretmeninizle iletişime geçiniz.

CEVAP ANAHTARLARI

ÖĞRENME FAALİYETİ 1 CEVAP ANAHTARI

1	D
2	D
3	D
4	Y
5	D
6	A

ÖĞRENME FAALİYETİ-2 CEVAP ANAHTARI

1	D
2	Y
3	D
4	Y
5	Y
6	C

ÖĞRENME FAALİYETİ-3 CEVAP ANAHTARI

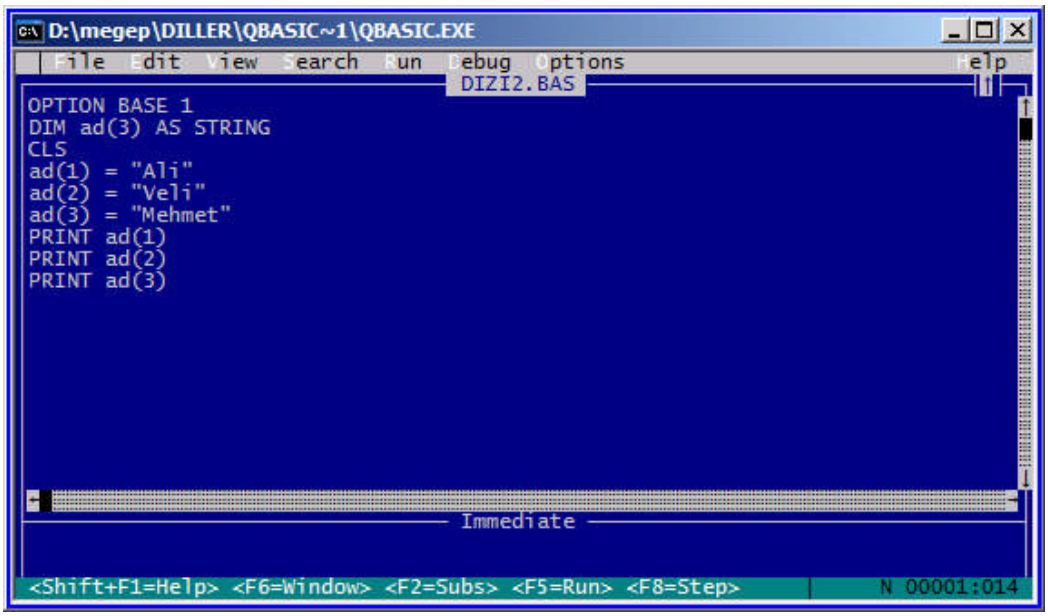
1	Y
2	D
3	Y
4	D
5	D
6	C

Cevaplarınızı cevap anahtarları ile karşılaştırarak kendinizi değerlendiriniz.

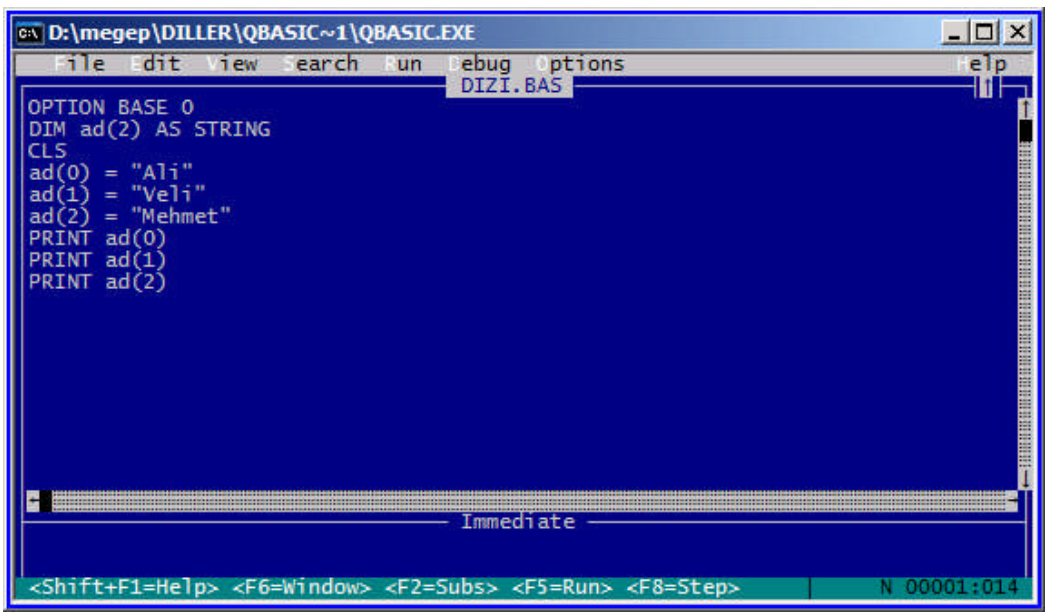
SÖZLÜK

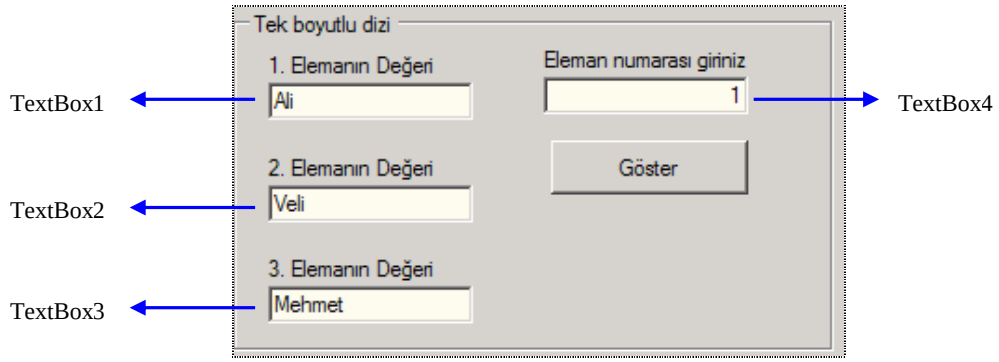
İsim	Okunuş	Anlam
priority	prayoriti	işlemciye yaptırılan işlemlerin öncelik değerleri
procedure	pırısiicır	yordam, prosedür
project	pricekt	proje, tasarı
prompt	prompt	uyarı, MS-DOS sistem hazır simgesi c:\> gibi
protect	prıtekt	korumak, protection – koruma
queue	kyu	kuyruk
real	rııl	ondalıklı sayılar
recognize	rıkıgnayz	tanımak
record	rıkırd	rekor, kayıt
recover	rikavır	bilgileri kurtarmak
recursive	rikörsiv	kendini çağıran fonksiyon
register	recıstır	yazmaç, kayıtlı kullanıcı olmak
registry	recıstri	Windows kayıt bilgileri
reserve	rizörv	ayırmak, rezervasyon
revert	rivört	geri dönüştürmek, kurtarmak
root	ruut	kök, ana dizin
scene	sin	sahne
script	sıkript	yardımcı programlama dili, el yazısı
scroll	skrol	kaydırmak
sector	sektır	bölge, disk iz parçası
sequence	sikuins	sıra
server	sörvır	ana bilgisayar, sunucu
set	set	küme, takım, ayarlamak (setting)

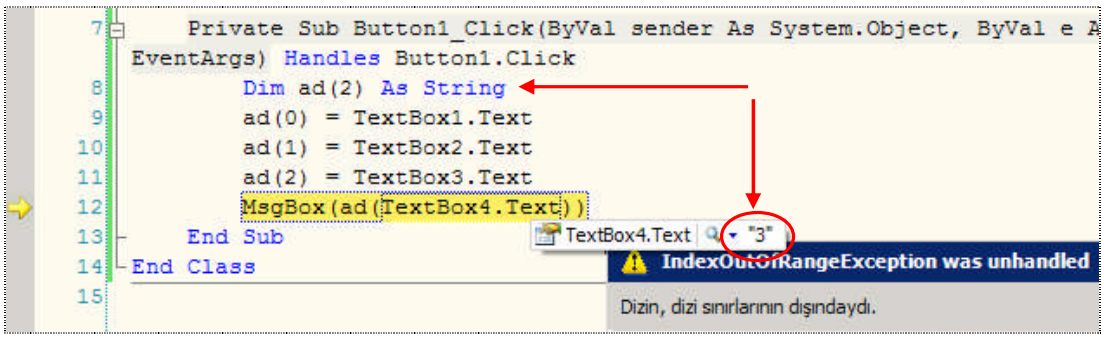
KOD ÖRNEKLERİ



? Her iki örnekte de 3 isim bilgisi saklanır. Ekran görüntüleri aynıdır. Acaba neden değişkenlerin tanımlandığı yer olan DIM kısmında birinci örnekte 3, ikinci örnekte 2 yazmaktadır?







Dizinin sınırları dışında bir değer çağrılması hatası; "*Index Out Of Range*"

```
OPTION BASE 1
DIM ad(3) AS STRING
DIM i, j AS INTEGER  _
CLS

FOR i = 1 TO 3
 PRINT i, ". ismi giriniz"
 INPUT "", ad(i)
NEXT

PRINT "Girilen isimlerin listesi:"
FOR j = 1 TO 3
 PRINT ad(j)
NEXT
```

? Visual Basic çok boyutlu dizi örneğinde eğer "DataGridView1"de yeterli veri girilmemiş ise nasıl bir hata meydana gelebilir? Nasıl bir önlem alırsınız?

Visual Basic dilinde çok boyutlu dizi örneği

```
Private Sub Button4_Click
      Dim bilgi(1, 5) As String 'İki boyutlu dizi tanımlama kısmı
      Dim i, j As Integer
      For i = 0 To 1
          For j = 0 To 5
              bilgi(i, j) = DataGridView1.Item(i, j).Value
          Next
      Next
      MsgBox("Arama sonucu " & bilgi(TextBox5.Text,     _
TextBox6.Text))
 End Sub
```

? Dinamik dizi örnek programında, dizi hazırlanmadan arama yapılırsa nasıl bir hata meydana gelebilir? Nasıl bir önlem alırsınız?

Visual Basic dilinde dinamik dizi örneği

```
Dim DinamikDizi() As String 'dizi boyutu belli değil

Private Sub Button6_Click
      ReDim DinamikDizi(TextBox8.Text) 'kullanıcı boyutu girdi
      Dim i As Integer
      For i = 0 To TextBox8.Text 'kullanıcı değerleri giriyor
          DinamikDizi(i) = InputBox("Eleman değeri")
      Next
End Sub

Private Sub Button5_Click 'aranan eleman ekrana gelir
      MsgBox(DinamikDizi(TextBox7.Text))
End Sub
```

C dilinde yapı örneği

```
struct birOgrenci{
          char *Ad;
          int Notu;
}Ogrenciler;

main()
{
Ogrenciler.Ad = "Ali Can";
Ogrenciler.Notu = 34;
printf("Bir öğrencinin adı ve notu: %s %d",
        Ogrenciler.Ad, Ogrenciler.Notu);
}
```

```
Ali Can
55

Hasan Kuzu
65

Murat Bol
67


0 öğrencinin adı ve notu: Ali Can 55
1 öğrencinin adı ve notu: Hasan Kuzu 65
2 öğrencinin adı ve notu: Murat Bol 67
```

Yapı dizisi kullanımı

C dilinde yapı dizisi örneği

```
#include <stdio.h>

struct birOgrenci{
          char *Ad[15];
          int Notu;
}Ogrenciler[2];      //Dizi olarak yapı tanımlanması

main()
{
int i;
clrscr();            //Yapı içine veri aktarma
for (i=0; i<3; i++) {
          fflush(stdin);
          gets(Ogrenciler[i].Ad);
          scanf("%d",&Ogrenciler[i].Notu);
          printf("\n");
          }
for (i=0; i<3; i++){//Veri çıktısı alma
          printf("\n%d öğrencinin adı ve notu: %s %d", i,
          Ogrenciler[i].Ad, Ogrenciler[i].Notu);
          }
}
```

? Aşağıdaki işaretçiler ile ilgili C dilinde olan satırlarını inceleyiniz.

```
char *hata = "Dosya yok!" //metin işaretçilere ilk değer atanabilir
```

```
int *iptr;
iptr = (int *) 0x1b64;
//Belli bir adres işaretçiye atanabilir, ama tehlikelidir, neden?
*iptr = 2001;           //işaretçiye değer aktarımı
printf ("\n%p", iptr);  //ekrana 1B64 yazar
printf ("\n%d", *iptr); //ekrana 2001 yazar
printf ("\n%X", &iptr); //ekrana FFF2 benzeri işaretçinin adresini yazar
```

```
char far *cptr;     //1 MB (0-FFFFF) içinde tanımlanabilen işaretçi
p = (char far *) 0x124532bc    //1245 segment, 32bc offset adresidir
```

```
char far *ekran = (char far *) 0xb8000000;
//B800 DOS'un ekran bilgilerinin tutulduğu yerdir
*(ekran+20) = '#';
*(ekran+21) = 10;
//metin ekranın ilk satırının 11. sütununa # karakterini basar
//Neden 20 deyince 11. sütun oldu?
//Not: Çift adresler veri, tek adresler renk bilgisi olarak kullanılır
```

```
int a[3] = {12, 55, 88};       //üç elemanlı dizi ve ilk değer atamaları
int *iptr;                     //int *iptr = a; da olabilirdi
iptr = a;
```

```
//dizi başlangıç adresi & ile gösterilemez
printf ("\n%d",*(iptr+1));
//ekrana 55 yazar
printf ("\n%d", iptr[2]);
//ekrana 88 yazar; *(iptr+2) ile iptr[2] aynı anlamdadır
```

```
void deneme (int *iptr) //fonksiyona bir değişkenin adresi gelecek
{
*iptr=100;          //işaretçinin gösterdiği yere değer aktarılır
}
void main(void)     //void ne anlama gelir?
{
int x;
deneme (&x);        //x değişkeninin adresi fonksiyona yollanır
printf ("%d", x);   //ekrana 100 yazar
}
```

```
char *ad_oku (void) //değer döndüren fonksiyon
{
 char s[50];
 printf ("Adınız=");
 gets(s);
 return s;          //okunan metnin değeri gönderilir
}
main()
{
 char *cptr;
 cptr= ad_oku();
//dönecek değer ancak bir char tipi işaretçiye aktarılabilir
 printf("%s", cptr);
}
```

? Üstte ekran belleğine ulaşma ile ilgili bir örnek vardır. Sizce belleğin istediğiniz yerine ulaşılması programcıya özgürlük mü sağlıyor, yoksa tam tersine, hazır ekrana yazı yazma komutları dururken böyle işlem yapmak, daha da kısıtlı bir ortam mı sağlıyor?

ÖNERİLEN KAYNAKLAR

- computer.howstuffworks.com
- en.wikipedia.org/wiki/Pointer
- java.sun.com/docs/books/tutorial/java
- www.bloodshed.net/devcpp.html
- www.robsite.de/programme.php?prog=ccompiler
- www.seslisozluk.com
- www.yunus.projesi.com

KAYNAKÇA

- BAĞRIYANIK Tarık, **Programlama Ders Notları ve Uygulamalı Genel Programlama Kitabı** (www.yunus.projesi.com)

- WALLACE Wang, **Beginning Programming for Dummies**, Wiley Basımevi, Indianapolis, 2004