ALGORİTMA ve PROGRAMLAMA II
2004-2005 BAHAR YARIYILI

Y. Doç. Dr. Aybars UĞUR

Dersin İçeriği

1. C# DİLİNE GİRİŞ
2. VERİ TİPLERİ ve DEĞİŞKENLER
3. PROGRAM KONTROL (DENETİM) YAPILARI
4. DİZİLER, STRING ve KARAKTER DİZİLERİ
5. METOTLAR ve ÖZYİNELEME
6. NESNEYE YÖNELİK PROGRAMLAMA
7. KUYRUKLAR ve YIĞIT
8. BAĞLAÇLI LİSTELER
9. ARAMA, SIRALAMA
10. DOSYALAR ve GUI

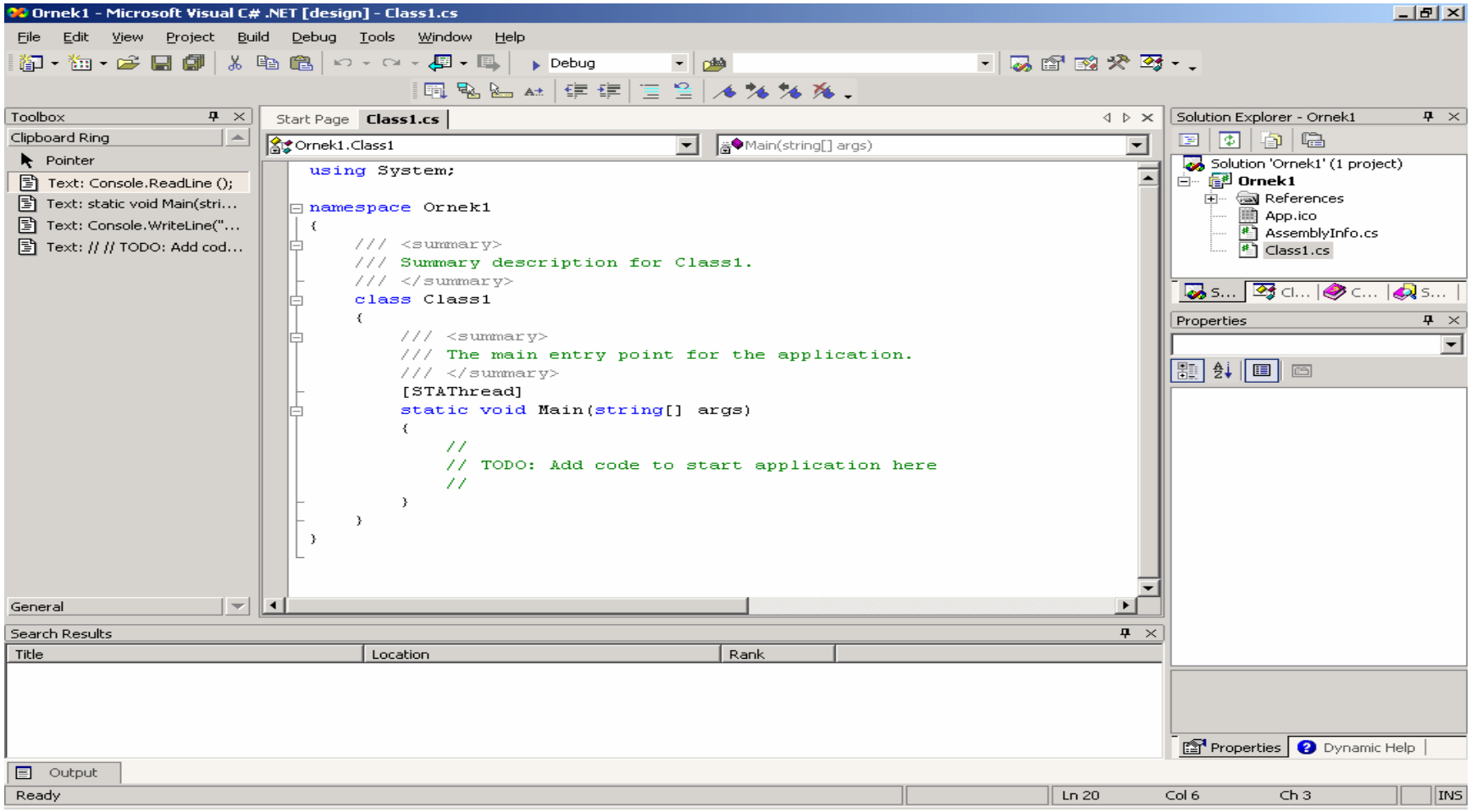
ALGORİTMA ve PROGRAMLAMA

- **Algoritma :** Bir problemin çözümünde kullanılan komutlar dizisi. Bir problemi çözmek için geliştirilmiş kesin bir yöntemdir. Bir algoritma, bir programlama dilinde (Java, C, Pascal gibi) ifade edildiğinde **program** adını alır.
- Algoritmanın Önemi
- Programlama Dilleri

C# DİLİNE GİRİŞ

- C#, “event-driven”, nesne yönelimli ve görsel bir programlama dilidir.
- Web tabanlı uygulamaların ve mobil iletişim cihazlarının yaygınlaşması sonucu, programlama ortamlarında oluşan gereksinimleri karşılamak ve yaşanmaya başlayan sorunları ortadan kaldırmak için .NET platformu ve C# programlama dili geliştirilmiştir. (Microsoft)
- C# Programları, IDE (Integrated Development Environment) kullanılarak hazırlanır. IDE ortamında, programların yazılması, işletilmesi, test edilmesi ve hatalardan arındırılması kolay olduğu için, bu şekilde uygulama yazılması işlemine RAD (Rapid Application Development) adı verilmektedir.

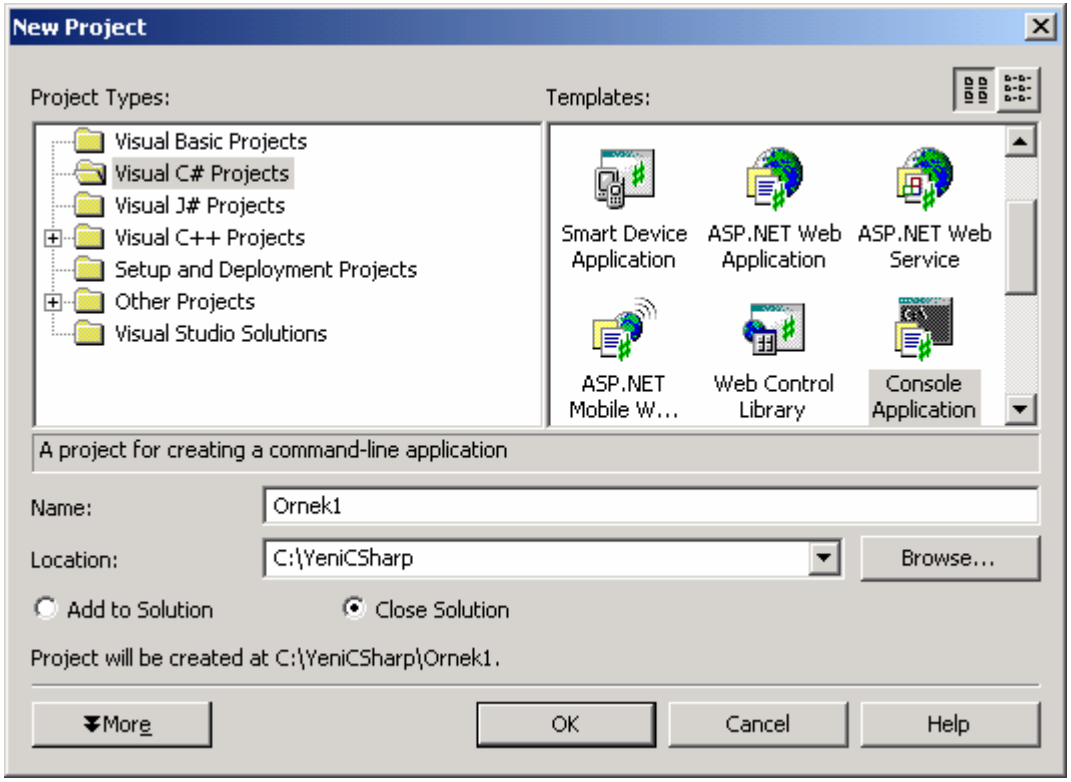
VISUAL STUDIO .NET ORTAMI
C# ile Yeni Proje Açıldıktan Sonra



ÖRNEK 1 (1)
KONSOL UGULAMASI : Merhaba Yazdıran Program

• “New Project” Düğmesi ile yeni bir proje açılır.

• Projeye verilmek istenen isim “Name” kısmına yazılır ve “Location” kısmında Projenin dosyalarının tutulacağı yer belirtilir. Burada Proje ismi olarak “Ornek1” yazılmıştır.



Konsol Uygulaması yapacağımız için “Console Application” simgesi seçilerek “OK” düğmesine basılır. Karşımıza sonraki sayfadaki kod gelir.

ÖRNEK 1 (2)

```
using System;

namespace Ornek1
{
/// <summary>
/// Summary description for Class1.
/// </summary>
class Class1
{
    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    static void Main(string[] args)
    {
    //
    // TODO: Add code to start application here
    //
    }
}
}
```

ÖRNEK 1 (3)

- //
- // TODO: Add code to start application here
- //

Yerine İstenen işi yapacak olan kod yazılır:

```
Console.WriteLine("Merhaba");
```

Kod aşağıdaki hale gelir :

```
static void Main(string[] args)
{
    Console.WriteLine("Merhaba");
}
```

F5'e basılarak veya Menüden Debug-Start ile veya Araç çubuğundaki Start düğmesi ile program çalıştırılır.

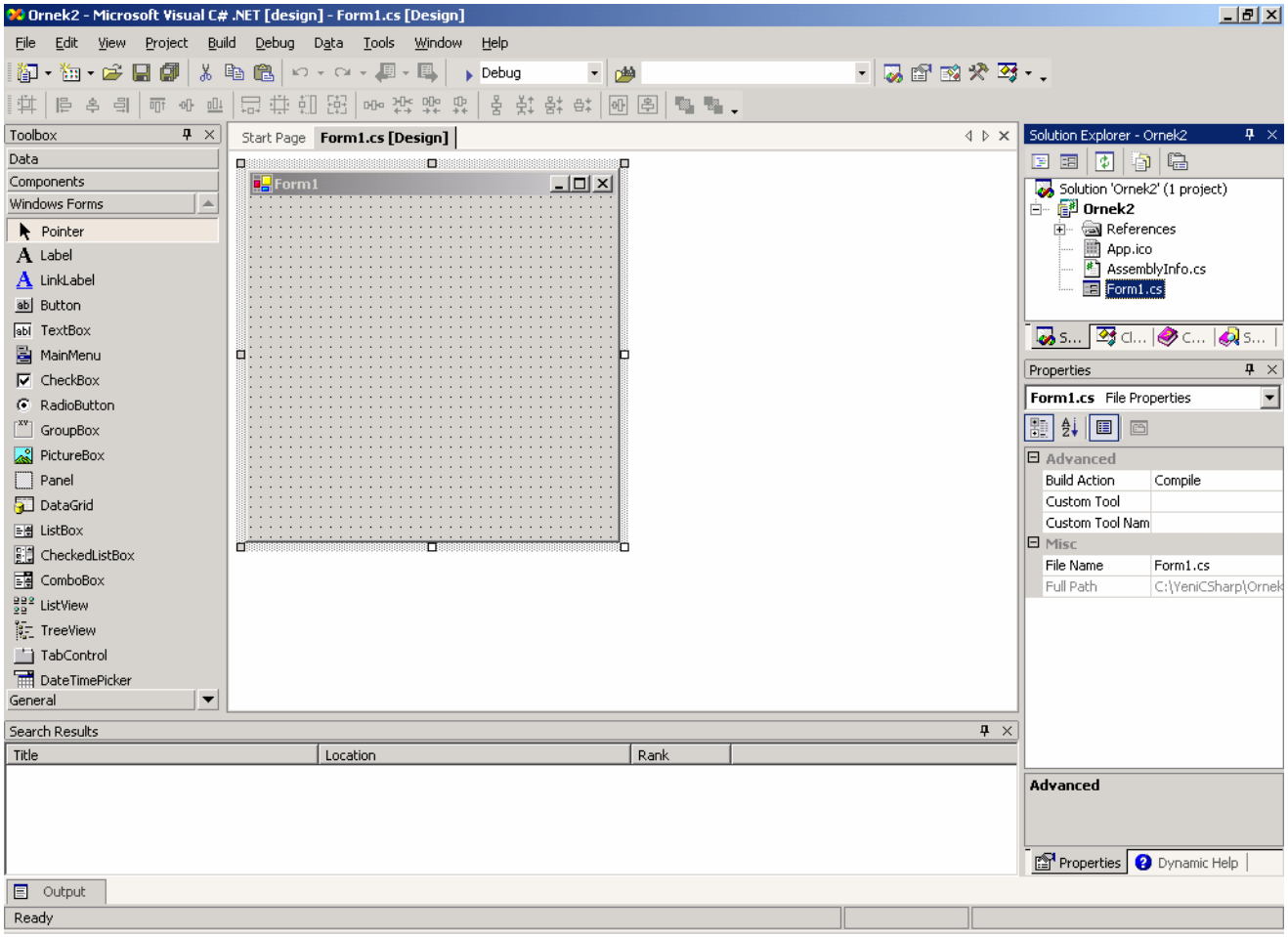
Ekran Çıktısı : Merhaba

ÖRNEK 1 (4)

Kod çalışmakta, Konsola “Merhaba” yazdırılmakta ve hemen C# ortamına geri dönülmektedir. Konsola yazılanları uzun süre görmek için kodun sonuna “Console.ReadLine();” ifadesi eklenebilir. Bu durumda, program girdi bekleyecek ve ancak “Enter” tuşuna basıldığında Konsol penceresi kapanacaktır.

```
static void Main(string[] args)
{
        Console.WriteLine("Merhaba");
        Console.ReadLine();
}
```

VISUAL STUDIO .NET ORTAMI
C# ile Windows Projesi Açıldıktan Sonra



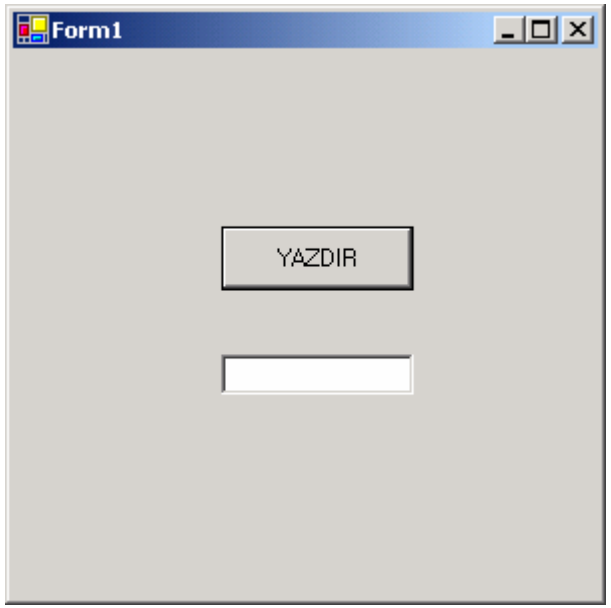
ÖRNEK 2 (1)
WINDOWS UYGULAMASI

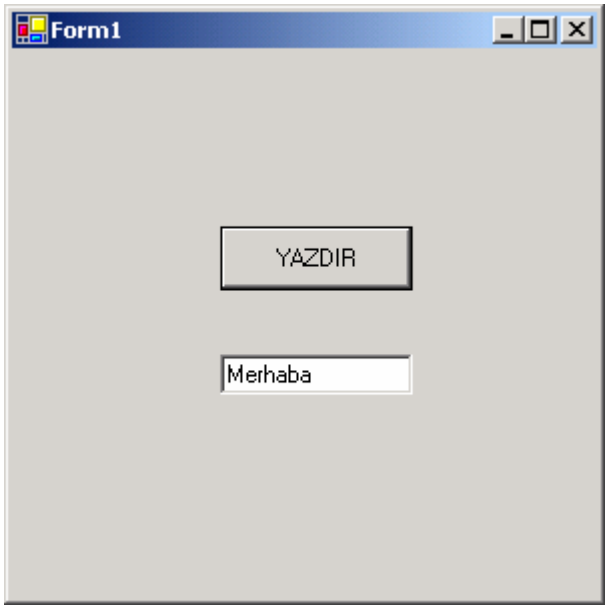
- “Windows Forms” Araç Kutusundan Düğme “Button” seçilerek Form üzerine ilgili boyutlarda yerleştirilir. “Properties” penceresinden “Text” sahası bulunarak “YAZDIR” kelimesi yazılır. Düğmenin üzerindeki metni belirtir.
- “Windows Forms” Araç Kutusundan Metin Kutusu “TextBox” seçilerek Form üzerine yerleştirilir. “Text” sahasındaki “textBox1” silinerek, form açılışında metin kutusunun boş açılması sağlanır.
- “YAZDIR” düğmesine çift tıklanarak kod ekranına gelinir. Programın işletimi sırasında “YAZDIR” düğmesine basıldığında yapılacak olan işlemleri içeren kod yazılır. Metin Kutusuna “Merhaba” yazdırmak için kod şu şekilde değiştirilir : (textBox1 metin kutumuza C#’ın verdiği isimdir. Değiştirilebilir)

```
private void button1_Click(object sender, System.EventArgs e)
{
    textBox1.Text = "Merhaba";
}
```

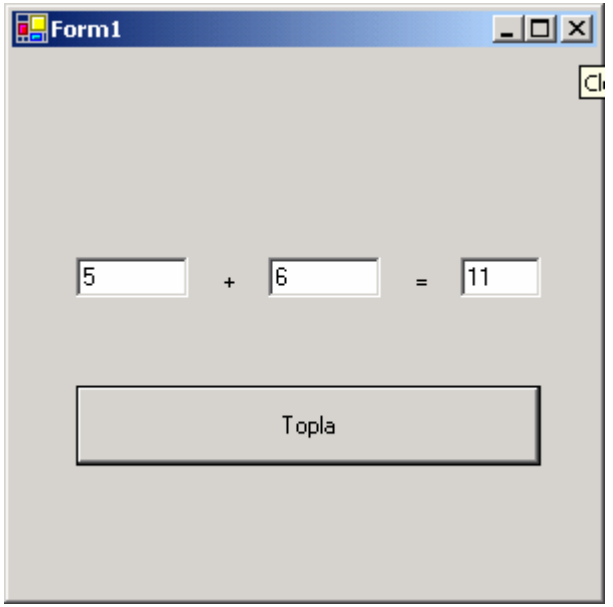
ÖRNEK 2 (2)
İşletimden Sonraki Pencere Görünümü

Yazdır Düğmesine Basıldıktan Sonra





ÖRNEK 3
LAB'DA YAPILACAK



New Project – Visual C# Projects – Windows Application

- İki Etiket
- Üç Metin Kutusu
- Bir Düğme eklenir.

.....

Düğme için aşağıdaki kod yazılır:

```
private void button1_Click(object sender, System.EventArgs e)
{
  textBox3.Text = "" + (Double.Parse (textBox2.Text) +
                        Double.Parse (textBox1.Text));
}
```

C# Programının İşletimi
CMD Konsol Uygulaması

- Aşağıdaki program herhangi bir isim ile kaydedilir. (Burada Ornek1.cs adı verilmiştir)
- “csc Ornek1.cs” komutu verilerek derlenir. (path ayarı!)
- “Ornek1” programı çalıştırılır.

```
using System;

class Welcome1
{
  static void Main(string[] args)
  {
    Console.WriteLine("Merhaba");
  }
}
```

C# VERİ TİPLERİ

Veri Tipi	Anlamı
int	tamsayı (32 bit) (-2,147,483,648..2,147,483,647)
long	uzun tamsayı (64 bit)
short	kısa tamsayı (16 bit) (-32768..32767)
float	kayan noktalı sayı (kns) (32 bit) (1,5E-45..3,4E+38)
double	çift duyarlıklı kns (64 bit) (5E-324..1.7E+308)
byte	8 bit işaretsiz tamsayı (8 bit) (0..255)
bool	true/false değerleri
char	karakter (16 bit)
uint	işaretsiz tamsayı (32 bit) >4 milyar
ulong	(64 bit) (0..18,446,744,073,709,551,615)
ushort	(16 bit) (0..65535)
decimal	(128 bit) (1E-28..7,9E+28) (hatasız)

SINIF ADI	KısaAdı	Tanımı
System.Object	*object*	Base class for all CTS types
System.String	*string*	String
System.SByte	*sbyte*	Signed 8-bit byte
System.Byte	*byte*	Unsigned 8-bit byte
System.Int16	*short*	Signed 16-bit value
System.UInt16	*ushort*	Unsigned 16-bit value
System.Int32	*int*	Signed 32-bit value
System.UInt32	*uint*	Unsigned 32-bit value
System.Int64	*long*	Signed 64-bit value
System.UInt64	*ulong*	Unsigned 64-bit value
System.Char	*char*	16-bit Unicode character
System.Single	*float*	IEEE 32-bit float
System.Double	*double*	IEEE 64-bit float
System.Boolean	*bool*	Boolean value (*true/false*)
System.Decimal	*decimal*	128-bit data type exact to 28 or 29 digits—mainly used for financial applications where a great degree of accuracy is required

DEĞİŞKENLER

Tip *değişken ismi*

```
int sayi1; // int veri tipidir.
/* sayi1 değişkeninin tamsayı tipinde olduğunu
belirtir */
.....
sayi1=5;
float sayi=5.7f;
```

VERİ TİPİ DÖNÜŞÜMLERİ

```
int i=10;
float f;
f=i;

double sayi;
int karekok = (int) Math.Sqrt(sayi);
```

YAZDIRMA KOMUTU : Console.WriteLine

```
Console.WriteLine("Not Ortalaması = " + ort);

Console.WriteLine("Şubat {0} veya {1} gündür", 28,29);

Console.WriteLine("Sayı\tKaresi");
Console.WriteLine("{0}\t{1}",5,5*5);
Console.WriteLine("{0,8}{1,10}",7,7*7);

Console.WriteLine("10/3 = {0:#.##}",10.0/3.0);
```

OPERATÖRLER - I

Hesaplamalarda kullanılan operatörler :

Aritmetik : +, -, *, /, % (Mod, kalan), ++, --

Mantıksal : &&, ||, !, &, |, ...

İlişkisel :

== (eşittir)

!= (eşit değildir)

>, <, >=, <=

OPERATÖRLER - II

Atama Operatörü : değişken = deyim;

```
int x = 5;
double sayi = -3.5;
int a, b, c;
a=b=c=100; // Atama zinciri
```

Bileşik Atamalar : +=, -=, *=, /=, ...

```
x-=10; // x = x - 10 ile eşdeğer
```

YAPISAL PROGRAMLAMA

Yapısal Programlamada üç tür denetim yeterlidir:

- Sıra (Sequence)
- Seçim (Selection)
- Tekrar (Repetition)

Sıralı işletim ?

PROGRAM DENETİM YAPILARI

- SEÇİM YAPILARI
  - if
  - if/else
  - switch
- TEKRAR YAPILARI (Döngüler)
  - while
  - do/while
  - for
  - foreach

SEÇİM YAPILARI : IF

if (koşul) ifade

if(notu>=60) Console.WriteLine(“Geçti”);

Koşul : bool veri tipindedir. true veya false olabilir.

İfade bloğu :
if(notu>=60)
{ Console.WriteLine(“Geçti”); sayac++; }

SEÇİM YAPILARI : IF/ELSE

```
if (koşul) ifade;
else ifade;

if(notu>=60)
  Console.WriteLine(“Geçti”);
else
  Console.WriteLine(“Kaldı”);
```

KÜMELENMİŞ (İÇİÇE) IF'LER (NESTED IF)

Verilen bir sayının işaretini (negatif, pozitif veya 0) bulan C# kod parçası :

```
if (i==0) Console.WriteLine("İşaretsiz");
else
  if(i<0) Console.WriteLine("Negatif");
  else Console.WriteLine("Pozitif");
```

CONDITIONAL OPERATOR (?:)

```
Console.WriteLine(notu>=60?”Geçti”:”Kaldı”);
```

Eşdeğer ifade :

```
if (notu>=60)
  Console.WriteLine(“Geçti”);
else
  Console.WriteLine(“Kaldı”);
```

IF-ELSE-IF MERDİVENİ (CASCADING IF)

```
if(koşul) ifade
else if (koşul) ifade
else if (koşul) ifade
.....
else ifade;
```

```
if (notu>=90)
  str = “A”;
else if (notu>=80)
  str = “B”;
else if (notu>=70)
  str = “C”;
else if (notu>=60)
  str = “D”;
else str = “E”;
Console.WriteLine(str);
```

SEÇİM YAPILARI : SWITCH

```
switch (deyim) {
  case sabit1:
    ifade1;
    break;
 case sabit2:
    ifade2; break;
....
  default :
    ifade;
    break;
}
```

```
switch(sayi) {
  case 0 :
    Console.WriteLine(“ Sayı 0”);
    break;
 case 1 :
    Console.WriteLine(“ Sayı 1”);
    break;
 case 2 :
    Console.WriteLine(“ Sayı 2”);
    break;
  default : Console.WriteLine(“X”);
    break;
}
```

DÖNGÜLER : FOR

- for(int sayac=0; sayac<10; ++sayac)

for(başlangıç; devam koşulu; kontrol değişkeni değişimi)

```
for(int sayac=0; sayac<10; ++sayac)
{
  ifadeler;
}
```

foreach döngüsü için diziler bölümüne bakınız.

DÖNGÜLER : WHILE ve DO-WHILE

```
while(koşul) ifade

do {
  ifadeler
} while (koşul);

int sayac=1, toplam=0;
while(sayac<10) { toplam+=sayac; sayac++; };
```

DÖNGÜDEN ÇIKMAK : BREAK

```
for(int x=1; x<10; ++x)
{
  toplam+=x;
  if (x==5) break;
}
Console.WriteLine(”1 ile 5 arasındaki sayıların
   toplamı = ”+toplam);
```

BREAK sadece en içteki döngüden çıkar.

DÖNGÜDE ERKEN TEKRAR : CONTINUE

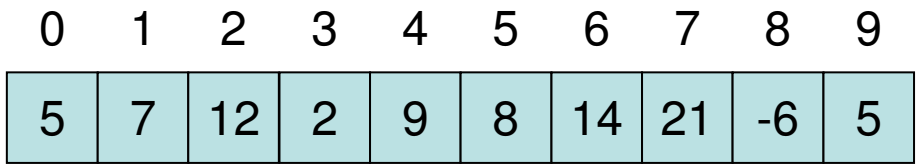
1 ile 100 arasındaki tek sayıları yazan program :

```
for(int i=0; i<100; ++i)
{
  if ( (i%2)!=1) continue;
  Console.WriteLine(i);
}
```

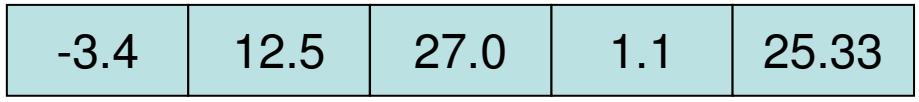
DİZİLER

Dizi (array), aynı tipteki değişkenler topluluğudur.

```
int[] arr = new int[10];
int[] dizi = {5,7,12,2,9,8,14,21,-6,5};
```




```
double[] sayilar; sayilar = new double[5];
```



DİZİ KULLANIM ÖRNEKLERİ

```
dizi[2]++;
dizi[3]=dizi[1]+dizi[2];
Console.WriteLine(dizi[5]);
```

Dizi elemanlarının toplamını bulduran programı yazınız : dizi.Length kullanınız!

FOREACH

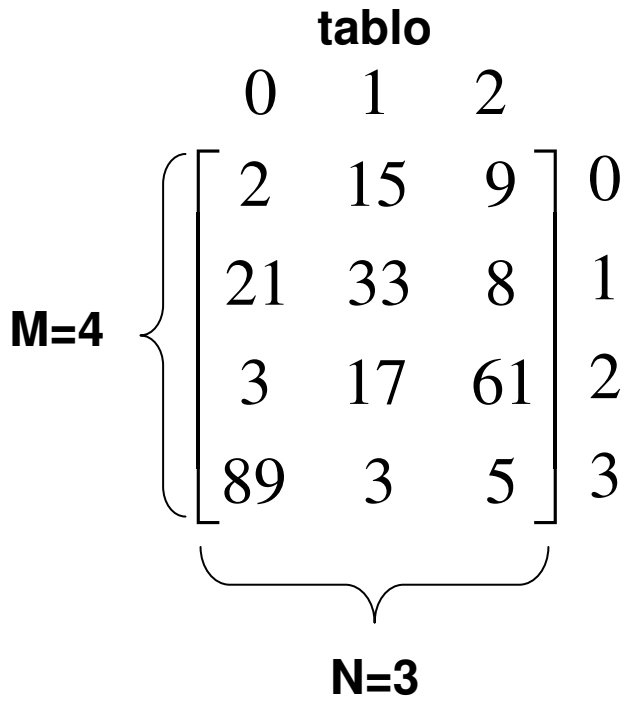
foreach döngüsü, özellikle bir koleksiyonun tüm elemanları üzerinde işlemler yapılacaksa yararlıdır :

```
int toplam=0;
foreach(int i in dizi) toplam+=i;
```

foreach döngüsü “break” kullanılarak daha erken de bitirilebilir.

İKİ BOYUTLU DİZİLER (MATRİSLER)

- M x N
- M satır, N sütun



- Oluşturulması : int[,] tablo = new int[4,3];
- Kullanımı : tablo[0,2]

ÇOK BOYUTLU DİZİLER

- Tip [,...,] isim = new tip[büyüklük1,..., büyüklükN]

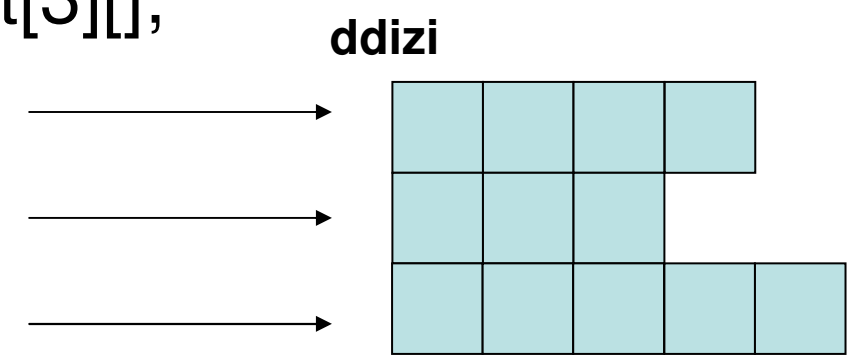
Örnekler

- double[,,] mdizi = new double[4,10,2]; // boyutu 3
- float[,,,] dizi4d = new float[5,5,5,5]; // boyutu 4

DÜZENSİZ DİZİLER (jagged array)

- Her biri farklı uzunluktaki dizilerin oluşturduğu dizidir.
- İki boyutlu dizilerde dikdörtgensel olmayan matrisler elde etmek için kullanılabilir : Her satırı farklı uzunlukta olabilen matris.

```
int[][] ddizi =new int[3][];
ddizi[0]=new int[4];
ddizi[1]=new int[3];
ddizi[2]=new int[5];
```



STRING'LER ve KARAKTER DİZİLERİ

- Karakter dizisi : char[] harfdizi = {'T','e','s','t'};

- String : Karakter dizisinden farklıdır.

string str1 = "Merhaba";

string str2 = new string(harfdizi);

BÖL ve ÇÖZ (Divide and Conquer)

- Yazılım Mühendisliği deneyimleri, büyük programlar geliştirmenin en iyi yolunun küçük program parçaları (modül) yazıp onları birleştirmek olduğunu göstermiştir.
- Böl ve Çöz olarak bilinen bu yöntem aynı zamanda, hatalardan arındırmayı, programı gelişen şartlara göre büyütmeyi, değişiklikler yapmayı kolaylaştırmak ve anlaşılırlığı artırmak gibi birçok avantajı da beraberinde getirmektedir.
- C#'ta temel modüller, sınıf (class) ve metotlardır (method). Metotlar yazılım içinde yeniden kullanılarak kodu ve yazılım geliştirme süresini kısaltmaktadır.

METOTLAR (METHOD)

- Bir işlemin yapılması için bir veya daha fazla ifade kullanmak gerekir. Verilen bir matrisi ekrana yazdırmak gibi. İlgili kodu “yazdir()” adını verdiğimiz bir metot içine yazarak istediğimiz zaman, ismi ile çağırabiliriz.
- .Framework Class Library (FCL), matematik hesaplamalarını, string, karakter, girdi/çıktı işlemlerini ve diğerlerini yapmak için hazır sınıflar ve metotlar içermektedir. Ayrıca değişik alanlarda hazırlanmış veya kendimizin daha önceden hazırladığı metotları da kullanmak mümkündür.

Çok Kullanılan Hazır Metotlar
Bazı Math Sınıfı Metotları

Abs(x)	Mutlak değer	Abs(-5.3) == 5.3
Ceiling(x)	x'i kendinden küçük olmayan en küçük tamsayıya yuvarlar	Ceiling(-9.8) == -9.0
Floor(x)	x'ten büyük olmayan en büyük tamsayıyı döndürür	Floor(-9.8) == -10.0
Cos(x) Sin(x),Tan(x)	Radyan cinsinden trigonometrik fonksiyonlar	Cos(0.0) == 1.0
Exp(x)	e^x	Exp(1.0) yaklaşık 2.718..
Log(x)	Logaritma	Log(2.718) yaklaşık 1.0
Max(x,y), Min(x,y)	Max ve Min fonksiyonları 2 sayıdan büyük/küçük olanı döndürür.	Max(3.7,12.3) == 12.3 Min(3.7,12.3) == 3.7
Pow(x,y)	Üs : x^y	Pow(9.0,.5) == 3.0
Sqrt	Karekök	Sqrt(900.0) == 30.0

METOTLARIN GENEL BİÇİMİ

```
erişim dönüş_tipi isim(parametre listesi)
{
  metodun gövdesi
}
```

Erişim : public, private gibi

Dönüş_tipi : metodun döndürdüğü veri tipi. Değer döndürmüyorsa void.

Parametre listesi : “,” lerle ayrılmış tip ve parametre ismi.

Değer Döndürmeyen Metotlar
void metotları

```
public void yazdir()
{
  Console.WriteLine(Merhaba);
}
```

Çağrılması :

```
yazdir();
```

Değer Döndüren Metotlar

```
public int topla(int a, int b)
{
  return (a+b);
}
```

Çağrılması :

```
int y=topla(5,6);
```

Metodun iki de parametresi var.

Parametre, Argüman, Return

Argüman : Metoda aktarılan değer

Parametre : Argümanı kabul eden değişken

Return : Metottan çıkmak veya geri dönmek

ÖZYİNELEME
RECURSION

Kendini doğrudan veya dolaylı olarak çağıran fonksiyonlara özyineli (recursive) fonksiyonlar adı verilir. Özyineleme (recursion), iterasyonun (döngüler, tekrar) yerine geçebilecek çok güçlü bir programlama tekniğidir. Orijinal problemin küçük parçalarını çözmek için, bir alt programın kendi kendini çağırmasını sağlayarak, tekrarlı işlemlerin çözümüne farklı bir bakış açısı getirir. Aşağıdaki faktöryel metodu özyineli bir metottur ve "Console.WriteLine(factorial(3));" deyimi ile çağrıldığında ekrana 6 yazdırır :

```
public int factorial(int n)
{
  if(n==0)
    return 1;
  else
    return (n*factorial(n-1));
}
```

SINIFLAR (CLASS)

Sınıf bir nesnenin şeklini tanımlayan şablondur. Veri, metot ve diğer bileşenleri içerir. Nesneler ise sınıfın örnekleridir.

```
class Ogrenci
{
  public string ad, soyad;
  public int yas;
  public void yazdir() {
    Console.WriteLine(ad+” “+soyad+” “+yas) };
}

Ogrenci ogrenci1 = new Ogrenci();
```

KUYRUKLAR (1)

Kuyruklar, eleman eklemelerin sondan (rear) ve eleman çıkarmaların baştan (front) yapıldığı veri yapılarıdır. Bir eleman ekleneceği zaman kuyruğun sonuna eklenir. Bir eleman çıkarılacağı zaman kuyrukta bulunan ilk eleman çıkarılır. Bu eleman da kuyruktaki elemanlar içinde ilk eklenen elemandır. Bu nedenle kuyruklara FIFO (First-In First-Out = ilk giren ilk çıkar) listeleri de denilmektedir. Gerçek yaşamda da bankalarda, duraklarda, gişelerde, süpermarketlerde, otoyollarda kuyruklar oluşmaktadır. Kuyruğa ilk olarak girenler işlemlerini ilk olarak tamamlayıp kuyruktan çıkarlar. Veri yapılarındaki kuyruklar bu tür veri yapılarının simülasyonunda kullanılmaktadır. Ayrıca işlemci, yazıcı, disk gibi kaynaklar üzerindeki işlemlerin yürütülmesinde ve bilgisayar ağlarında paketlerin yönlendirilmesinde de kuyruklardan yararlanılmaktadır. C#'ta Kuyruk yapısı hazır olarak vardır : (Koleksiyon sınıflarından) Queue sınıfının bazı metotları

Enqueue metodu **:** Kuyruğun sonuna eleman ekler.

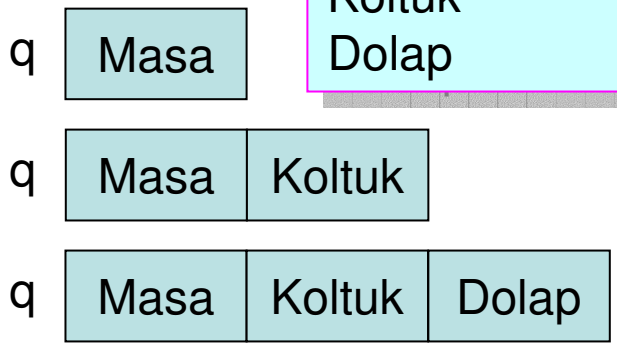
Dequeue metodu **:** Kuyruğun başından eleman siler.

KUYRUKLAR (2)

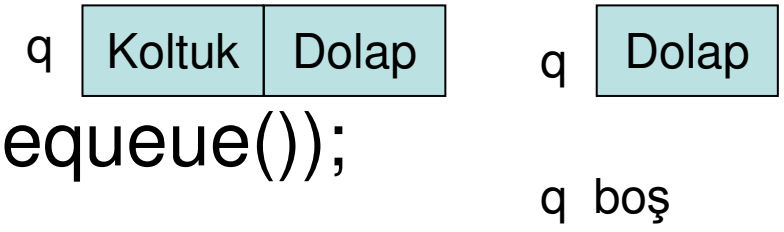
```
Queue q = new Queue();

q.Enqueue ("Masa");
q.Enqueue (“Koltuk");
q.Enqueue ("Dolap");
```

Ekran Çıktısı :

Masa
Koltuk
Dolap




```
while(q.Count!=0)
   Console.WriteLine(q.Dequeue());
```



YIĞIT (1)

Eleman ekleme çıkarmaların en üstten (top) yapıldığı veri yapısına yığıt (stack) adı verilir. Bir eleman ekleneceğinde yığıtın en üstüne konulur. Bir eleman çıkarılacağı zaman yığıtın en üstündeki eleman çıkarılır. Bu eleman da yığıttaki elemanlar içindeki en son eklenen elemandır. Bu nedenle yığıtlara LIFO (Last In First Out : Son giren ilk çıkar) listesi de denilir. C#'ta Yığıt yapısı hazır olarak vardır : (Koleksiyon sınıflarından) Stack sınıfının bazı metotları

Push : Yığıta (sonuna) eleman ekleyen metot.

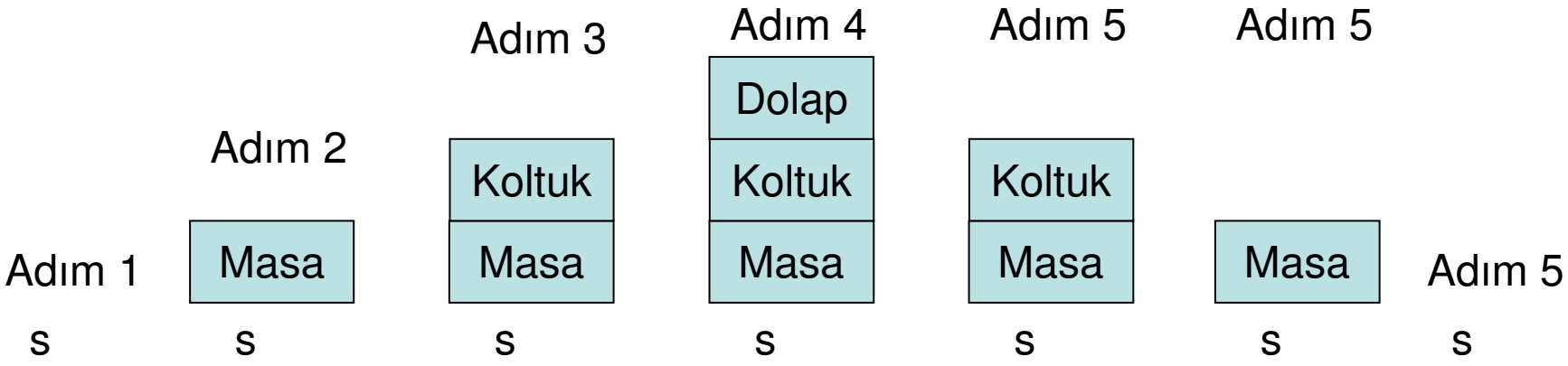
Pop : Yığıttan (sonundan) eleman silen metot.

YIĞIT (2)

1. Stack s = new Stack();
2. s.Push("Masa");
3. s.Push("Koltuk");
4. s.Push("Dolap");
5. while(s.Count!=0) Console.WriteLine(s.Pop());

Ekran Çıktısı :

Dolap
Koltuk
Masa



KAYNAKLAR

- Dr. Aybars UĞUR, **"Veri Yapıları Ders Notları",** 1999.
- Herbert Schildt, **"C# : The Complete Reference"**, McGraw-Hill, 2002 (Türkçe Baskısı : Herkes İçin C#)
- Deitel ve Diğerleri, **"C# : How to Program"**, Prentice-Hall, 2002.
- Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C., **"Introduction to Algorithms"**, Second Edition, MIT Press, McGraw-Hill, 2001.