

Visual Studio.Net -C#

8. HAFTA

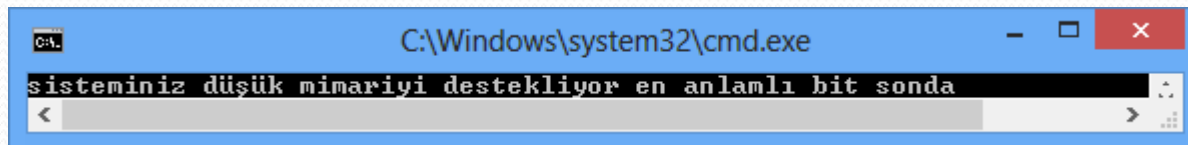
System İsim Alanı

System.BitConverter

- Alt seviye programlama da veriler byte dizisi şeklinde yani bitsel olarak işlenir.
- .NET içerisinde de bite çevirmek amacıyla BitConverter sınıfı bulunur.
- Bu sınıfın **IsLittleEndian** isimli bir özelliği bulunur. Bu özellik işlemci mimarisine göre verileri bellekte depolarken hangi sıralamada yerleştirildiğini kontrol eder.
- En önemli metodu da **GetBytes**'tir. Amacı da farklı sayı türlerini **byte** dizisine çevirmektir.

System.BitConverter

- Mimarinizi öğrenmek için;
- `using System;`
- `class bitconverter`
- `{`
- `public static void Main()`
- `{`
- `if (BitConverter.IsLittleEndian)`
- `Console.WriteLine("sisteminiz düşük mimariyi destekliyor`
`en anlamlı bit sonda");`
- `else`
- `Console.WriteLine("sisteminiz yüksek mimariyi destekliyor`
`en anlamlı bit başta");`
- `}`
- `}`



A screenshot of a Windows Command Prompt window. The title bar shows the path `C:\Windows\system32\cmd.exe`. The command prompt has a black background with white text. The output of the program is displayed on two lines: `sisteminiz düşük mimariyi destekliyor en anlamlı bit sonda`. The text is wrapped to fit the width of the window. The cursor is at the end of the second line.

System.BitConverter

- `int a=258; // 258/256=1 => 258-2561 *1 = 2`
- `//00000000 00000000 00000001 00000010`
- `2563 2562 2561 2560`
- `byte[] dizi=BitConverter.GetBytes(a);`
- `foreach(byte b in dizi)`
- `{ Console.WriteLine(b); }`
- 2
- 1
- 0
- 0
- `byte[] dizi={2,1,0,0};` → en anlamlı bit en sonda
`Console.WriteLine(BitConverter.ToInt32(dizi,0));`
- 258
- Bilinçli tür dönüşümünde kayıplar en anlamlı bitlerde olur.

System.Buffer

- **Buffer** sınıfı ile tür bilgisinden bağımsız bir biçimde **byte** düzeyinde veri işleme yapılır.
- Dizilerin belirli alanları başka bir diziye tür bilgisine bakılmaksızın aktarılabilir. Tüm veriler byte dizisi şeklinde düşünülür.
- **BlockCopy()**, **GetByte()**, **SetByte()** en önemli metotlarıdır.

System.Buffer

- `byte[] kaynak={1,2,3,1};`
- `00000001` , `00000010` , `00000011` , `00000001`
- `short[] hedef=new short[4];`
- `0000000000000000` , `0000000000000000` , `0000000000000000` , `0000000000000000`
- `Buffer.BlockCopy(kaynak,0,hedef,0,4);`
- *hedef dizisinin yeni hâli:*
- `0000001000000001` , `0000000100000011` , `0000000000000000` , `0000000000000000`
- `foreach(short a in hedef) Console.Write(" "+a);`
- Bu program, mimarimiz Little Endian ise ekrana: 513 259 0 0 yazar.
 - Burada derleyici her elemanı bellekte 1 bayt yer kaplayan kaynak dizisinin elemanlarını her elemanı bellekte 2 bayt kaplayan hedef dizisine olduğu gibi kopyaladı. Derleyici burada karşılaştığı ilk baytı düşük anlamlı bayta, ikinci baytı da yüksek anlamlı bayta kopyaladı.
- Bilinçsiz tür dönüşümü `Buffer.BlockCopy` mantığı ile gerçekleşir.

System.Buffer

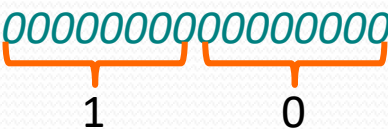

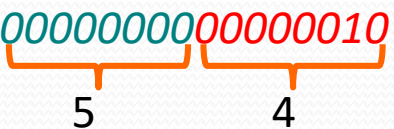
- **ByteLength() metodu**
- Kendisine parametre olarak verilen bir dizideki toplam bayt sayısını bulur. Örneğin kendisine parametre olarak gönderilen 3 elemanlı short türünden bir dizi sonucu 6 sayısı gönderir. Geri dönüş tipi int tir. Örnek:
 - `short[] dizi=new short[4];`
 - `Console.WriteLine(Buffer.ByteLength(dizi)); // 8 yazar`
- **GetByte() metodu**
 - `static byte GetByte(Array dizi, int a)`
 - dizinin a. baytını verir.
- **SetByte() metodu**
 - `static void SetByte(Array dizi,int a, byte deger)`
 - a. baytı deger olarak değiştirir.


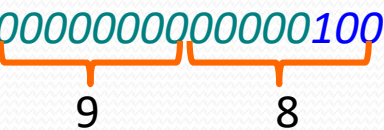
System.Buffer

- Örnek:

- `byte[] dizi={0,3,2,1,4}; Console.WriteLine(Buffer.GetByte(dizi,3)); // Ekran 1 yazar`
 - Bu kod `1` değerini döndürür. Eğer dizinin tipi byte değil de short olsaydı işler değişirdi. Çünkü o zaman hem sıfırla beslenen baytlar da hesaba katılırdı ve hem de bilgisayarımızın mimarisi sonucu etkilerdi. Bu durumu örneklendirelim:

- `short[] dizi={0,3,2,1,4};`

- 0->  , 2->  , 4-> 

- 6->  , 8-> 

- `Console.WriteLine(Buffer.GetByte(dizi, 4)); // Ekran 2 yazar`
- `Console.WriteLine(Buffer.GetByte(dizi,8)); // Ekran 4 yazar`

- Örnek:

- `byte[] dizi={2,1,4}; Buffer.SetByte(dizi,1,5);`
- `Console.WriteLine(dizi[1]); //Ekran 5 yazılır.`
- Yine eğer dizinin türü byte değil de int olsaydı işler değişirdi.

C# Temel Tür Yapıları

- **Convert Sınıfı**

- `using System;`
- `class ortalamaHesapla`
- `{ public static void Main()`
- `{ Console.Write("Vize1 = ");`
- `int vize1 = Convert.ToInt32(Console.ReadLine());`
- `Console.Write("Vize2 = ");`
- `int vize2 = Convert.ToInt32(Console.ReadLine());`
- `Console.Write("Final = ");`
- `int final = Convert.ToInt32(Console.ReadLine());`
- `double a = ((double)vize1 * 0.2 + (double)vize2 * 0.3 +`
- `(double)final * 0.5);`
- `Console.WriteLine("Ortalama = {0}", a);`
- `}`
- `}`

- C# 6.0 dan sonra `{deger}` şeklinde yazımı
- `Console.WriteLine($"Ortalama = {a}");`



```
C:\Documents and Settings\SoNDuRa...
Vize1 = 100
Vize2 = 80
Final = 60
Ortalama = 74
Press any key to continue_
```

String İşlemleri


- Programlarda kullanılan verilerin büyük bir çoğunluğu string türündendir.
- C# da string işlemleri System.String sınıfı içerisinde yer alan üye metod ve özelliklerle yapılır. String veriler için indeksleyici de tanımlanmıştır. Yani bir karakter dizisi gibi işlem görebilir. Fakat karakterler salt okunurdur.
- Stringler değişik şekillerde tanımlanabilirler:
 - `string a = "C# Programlama Dili";`
 - `char[] dizi = {'1','2','3','4','5'};`
 - `String s = new String(dizi);//12345`
 - `String s = new String(dizi,1,2); //dizi[1]'den itibaren 2 eleman stringe atandı. → 23`
 - `String s = new String('x',10);//xxxxxxxxxx`

String İşlemleri

- String sınıfının metotları oldukça fazladır. Önemli metotlarından bazıları şunlardır:
- **string.Concat()**
 - **static string Concat(params Array stringler)**
 - String verilerin ardarda eklenmesini sağlar. + operatörü ile eşdeğerdir.
- **string.Compare()**
 - İki string değeri karşılaştırır. **==** ve **!=** operatörleri ile benzer işlem gerçekleştirir. Fakat Compare metodunun bazı aşırı yüklemeleri ile daha gelişmiş (büyük küçük harf duyarlılığı gibi.) karşılaştırmalar yapılabilir.
 - **static int Compare(string a, string b)**
 - **static int Compare(string a, string b, bool c)**
 - **static int Compare(string a, int indeks1, string b, int indeks2)**
 - **static int Compare(string a, int indeks1, string b, int indeks2, bool c)**
 - kıyaslamada ilk elemanların a[indeks1] ve b[indeks2] sayılmasıdır. Yani stringlerdeki bu elemanlardan önceki elemanlar yok sayılır.

String İşlemleri

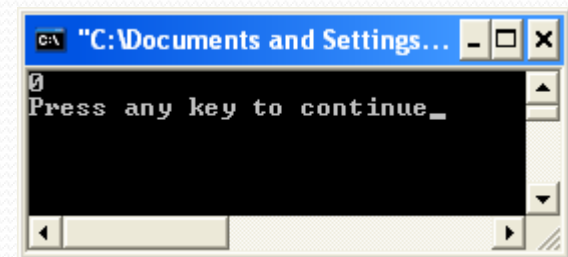
- **String.Concat()**
- `using System;`
- `class Concat`
- `{ public static void Main()`
- `{ String str1 = String.Concat("Bilgisayar", " Öğretmenliği");`
- `Console.WriteLine(str1);`
- `String str2 = String.Concat("Z", "5", "s", "3", "f");`
- `Console.WriteLine(str2);`
- `String str3 = "Bilgisayar" + " Öğretmenliği";`
- `Console.WriteLine(str3);`
- `String str4 = String.Concat(5, "A");`
- `Console.WriteLine(str4);`
- `}`
- `}`



A screenshot of a Windows command prompt window. The title bar shows the file path "C:\Documents and Settings\SoNDuRaK\C...". The window contains the following text: "Bilgisayar öğretmenliği", "Z5s3f", "Bilgisayar öğretmenliği", "5A", and "Press any key to continue_".

String İşlemleri

- `String.Compare()`
- `using System;`
- `class class1`
- `{ public static void Main()`
- `{ string a="Ali";`
- `bool b=true; //false büyük küçük duymaz`
- `string s="aLi";`
- `int c=String.Compare(a,s,b);`
- `Console.WriteLine(c);`
- `}`
- `}`



String İşlemleri

- `stringnesne.CompareTo(string str)` : İki stringi karşılaştırır. Değerler eşit ise 0, nesne değeri küçük ise negatif, büyük ise pozitif değer döndürür.
- `stringnesne.IndexOf()`
 - string içersinde alt stringlerin aranmasını sağlar. Geriye aranan alt stringin bulunduğu konumu ya da bulunamadı (-1) bilgisini döndürür.
 - `int IndexOf(string a)`
 - `int IndexOf(char b)`
- `stringnesne.LastIndexOf()`
 - IndexOf ile aynı çalışır Farkı ise aranan karakterin en son nerede görüldüğünün indeksini geri döndürür.
 - `int LastIndexOf(string a)`
 - `int LastIndexOf(char b)`

String işlemleri

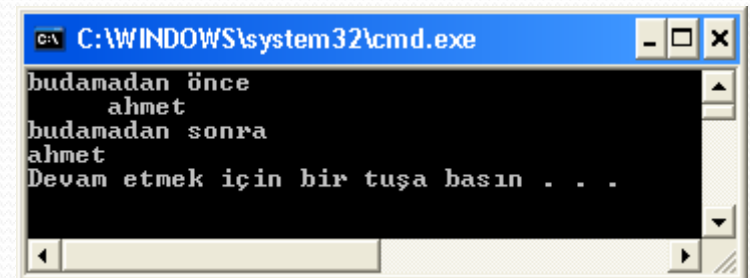
- `IndexOf`
- `using System;`
- `class arama`
- `{ public static void Main()`
- `{ string yazı="firat üniversitesi";`
- `Console.WriteLine(yazı.IndexOf("ver"));`
- `Console.WriteLine(yazı.IndexOf('t'));`
- `Console.WriteLine(yazı.IndexOf('c'));`
- `}`
- `}`



String İşlemleri

- `stringnesne.Trim()`
 - Bir string ifadenin başındaki ve sonundaki boşlukları ya da belirlenmiş karakterleri atar.

```
• using System;  
• class tarih {  
•     public static void Main()  
•     { string a = "    ahmet    ";  
•       Console.WriteLine("budamadan önce");  
•       Console.WriteLine("'" + a);  
•       a = a.Trim ();  
•       Console.WriteLine("budamadan sonra");  
•       Console.Write("'" + a);  
•     }  
• }
```



A screenshot of a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The window shows the output of the C# program: "budamadan önce", "ahmet", "budamadan sonra", and "ahmet". Below the output, there is a prompt "Devam etmek için bir tuşa basın . . ." (Press a key to continue).

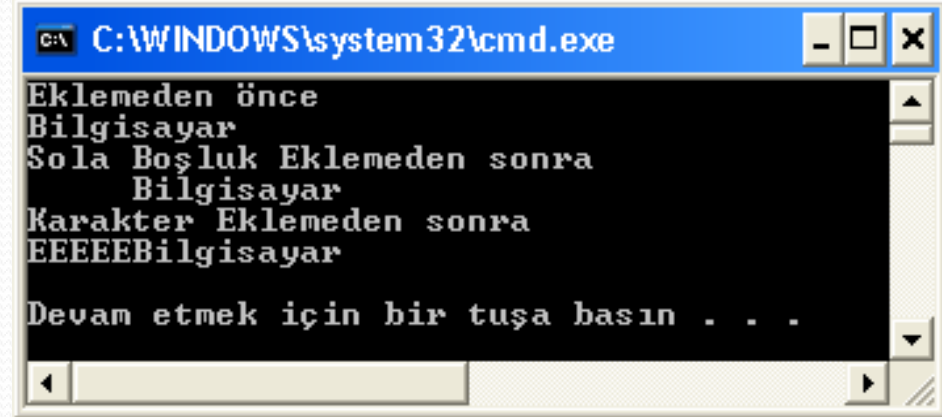
String İşlemleri

- **PadRight, PadLeft**

- Bir string'in sağına ya da soluna yeni karakterler ilave etmek için kullanılır.
- `string PadRight (int boyut);`
 - stringin uzunluğu boyuta eşit olana kadar stringin sağına boşluk ekler
- `string PadRight (int boyut, char a);`
 - stringin uzunluğu boyuta eşit olana kadar stringin sağına 'a' karakterini ekler
- `string PadLeft (int boyut);`
 - stringin uzunluğu boyuta eşit olana kadar stringin soluna boşluk ekler
- `string PadLeft (int boyut, char a);`
 - stringin uzunluğu boyuta eşit olana kadar stringin soluna 'a' karakterini ekler

String İşlemleri

- `using System;`
- `class tarih`
- `{`
- `public static void Main()`
- `{`
- `string a = "Bilgisayar";`
- `Console.WriteLine("Eklemeden önce");`
- `Console.WriteLine(a);`
- `a = a.PadLeft (15);`
- `Console.WriteLine("Sola Boşluk Eklemeden sonra");`
- `Console.WriteLine(a);`
- `Console.WriteLine("Karakter Eklemeden sonra");`
- `a = "Bilgisayar";`
- `a = a.PadLeft (15, 'E');`
- `Console.WriteLine(a);`
- `Console.WriteLine();`
- `}`
- `}`



```
C:\WINDOWS\system32\cmd.exe
Eklemeden önce
Bilgisayar
Sola Boşluk Eklemeden sonra
Bilgisayar
Karakter Eklemeden sonra
EEEEEBilgisayar

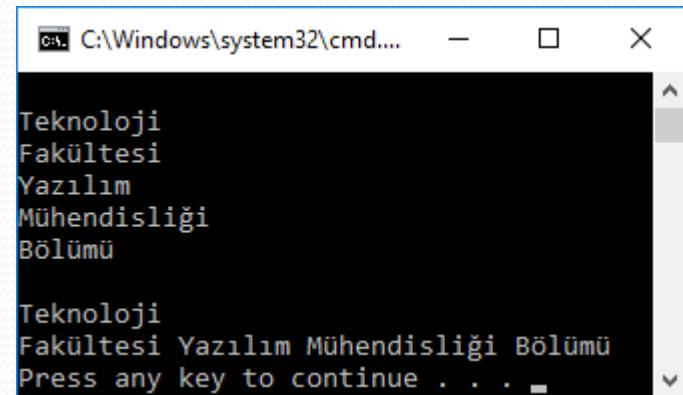
Devam etmek için bir tuşa basın . . .
```

String İşlemleri

- **bool string.StartsWith(string a)** : Metodu çağıran "**string a**" ile başlıyorsa **true**, diğer durumlarda **false** değeri üretir.
- **bool string.EndsWith(string b)** : Metodu çağıran "**string b**" ile bitiyorsa **true**, diğer durumlarda **false** değeri üretir.
- **string[] string.Split()**
 - Belli bir biçime sahip olan toplu string verileri, belirtilen ayırıcı karakterlerden ayırıp ayrı bir string dizisi üretir.
 - `string [] Split(params char[] ayırıcı)`
 - `string [] Split(params char[] ayırıcı, int toplam)`
 - İkinci metotta ise bu parçalama işlemi en fazla toplam sayısı kadar yapılır.
- **string string.Join()**
 - Split metodunun tersi gibi çalışır. Ayrı stringleri belli bir ayırıcı karakter ile birleştirip tek bir string üretir.
 - `static string join(string ayırıcı, string[] yazılar)`
 - `static string join(string ayırıcı, string[] yazı, int başla, int toplam)`
 - İkinci metotta ise [başla] 'dan itibaren toplam kadar yazı elemanı birleştirilir.

String İşlemleri

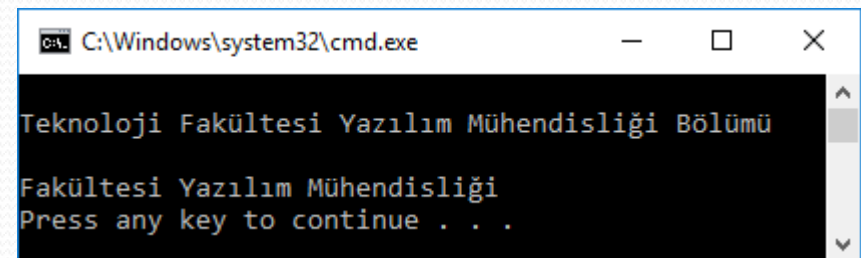
- `using System;`
- `class Ayirmaişlemi`
- `{ public static void Main()`
- `{`
- `string str="Teknoloji Fakültesi Yazılım Mühendisliği Bölümü";`
- `char[] ayırıcı={' '};`
- `string[] ayır=str.Split(ayırıcı);`
- `foreach(string i in ayır)`
- `Console.WriteLine(i);`
- `Console.WriteLine();`
- `ayır = str.Split(ayırıcı,2);`
- `foreach (string i in ayır)`
- `Console.WriteLine(i);`
- `}`
- `}`



```
C:\Windows\system32\cmd...  
Teknoloji  
Fakültesi  
Yazılım  
Mühendisliği  
Bölümü  
  
Teknoloji  
Fakültesi Yazılım Mühendisliği Bölümü  
Press any key to continue . . .
```

String İşlemleri

- `using System;`
- `class bireştirmeişlemi`
- `{ public static void Main()`
- `{`
- `string [] str={"Teknoloji", "Fakültesi", "Yazılım",`
`"Mühendisliği", "Bölümü"};`
- `string birles=String.Join(" ",str);`
- `Console.WriteLine(birles);`
- `Console.WriteLine();`
- `birles = String.Join(" ", str,1,3);`
- `Console.WriteLine(birles);`
- `}`
- `}`



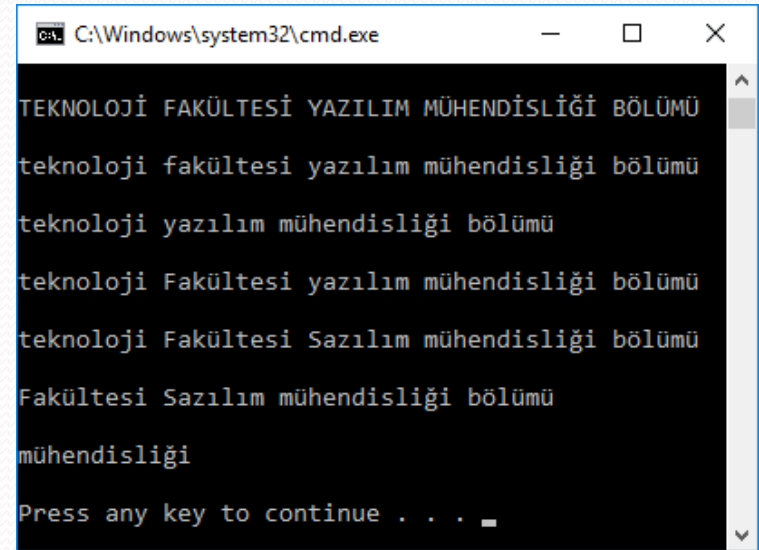
A screenshot of a Windows Command Prompt window titled "C:\Windows\system32\cmd.exe". The window displays the output of the C# program: "Teknoloji Fakültesi Yazılım Mühendisliği Bölümü" on the first line, "Fakültesi Yazılım Mühendisliği" on the second line, and "Press any key to continue . . ." on the third line. The window has standard Windows window controls (minimize, maximize, close) in the title bar.

String İşlemleri

- `stringnesne.ToUpper()`
 - Stringin karakterlerinin hepsini büyük harfe çevirip geri döndürür.
- `stringnesne.ToLower()`
 - Stringin karakterlerinin hepsini küçük harfe çevirip geri döndürür.
- `stringnesne.Remove(int indeks, int adet)`
 - String içinden belli sayıda karakterin atılmasını sağlar.
- `stringnesne.Insert(int indeks, string str)`
 - String içine belirli indeksten itibaren yeni bir string eklemek için kullanılır.
- `stringnesne.Replace(char | string c1, char | string c2)`
 - String içindeki belirlenen karakter ya da karakterleri başkalarıyla değiştirir.
- `stringnesne.Substring(int indeks | int indeks, int toplam)`
 - String ifadenin belli bir kısmının elde edilmesini sağlar.

String İşlemleri

- `using System;`
- `class digermetotlar`
- `{`
- `public static void Main()`
- `{`
- `string str="Teknoloji Fakültesi Yazılım Mühendisliği Bölümü";`
- `str=str.ToUpper();` `Console.WriteLine(str+"\n");`
- `str=str.ToLower();` `Console.WriteLine(str+"\n");`
- `str=str.Remove(9,10);` `Console.WriteLine(str+"\n");`
- `str=str.Insert(10,"Fakültesi ");` `Console.WriteLine(str+"\n");`
- `str=str.Replace('y','S');` `Console.WriteLine(str+"\n");`
- `str=str.Substring(10);` `Console.WriteLine(str+"\n");`
- `//10. karakterden sonrasını gösterir.`
- `str = str.Substring(18,12);` `Console.WriteLine(str + "\n");`
- `//18. karakterden sonra 12 karakter gösterir.`
- `}`
- `}`



```
C:\Windows\system32\cmd.exe

TEKNOLOJİ FAKÜLTESİ YAZILIM MÜHENDİSLİĞİ BÖLÜMÜ
teknoloji fakültesi yazılım mühendisliği bölümü
teknoloji yazılım mühendisliği bölümü
teknoloji Fakültesi yazılım mühendisliği bölümü
teknoloji Fakültesi Sazılım mühendisliği bölümü
Fakültesi Sazılım mühendisliği bölümü
mühendisliği
Press any key to continue . . .
```

Biçimlendirme

- Program çıktılarının belli bir düzende olması oldukça önemlidir. Bazı zamanlar standart çıktıların anlaşılması zor olabilir.
- Bu problemi çözmek için biçimlendirme teknikleri kullanılır. Bu teknikler yalnızca biçimlendirmeyi destekleyen komutlar tarafından kullanılabilir.
- **Console.WriteLine()**, **String.Format()** ve **ToString()** metotları biçimlendirmeyi destekleyen metotlardır.
- Bu metotlarda kullanılan **{ }** parantezleri arasındaki ifadeler belli değişkenlerin belli bir düzende biçim metnine aktarılmasını sağlıyor:
 - `Console.WriteLine("Merhaba {0}!", isim)`

Biçimlendirme

- En genel kullanımı
 - { **değişken_no**, **genişlik** : **format** }
 - **int** a=54;
- Sadece değişken verildiğinde değişkenin türüne göre varsayılan ayarlar kullanılır.
- **Genişlik**, yazılacak olan verinin diğer verilerle olan mesafesini ve hangi yöne hizalanması gerektiğini belirler.
 - Console.WriteLine("{0,10} numara",a);
 - Console.WriteLine("{0,-10} numara",a);
 - 54 numara
 - 54 numara
- **Format**, ise verinin türüne göre değişik biçimlendirilmesini sağlar.

Biçimlendirme

Belirleyici	Açıklama	Duyarlılık Anlamı
C/c	Para birimi	Ondalık basamakların sayısını verir.
D/d	Tam sayı verisi	En az basamak sayısını belirtir, gereğinde bos olan basamaklar sıfır ile beslenir.
E/e	Bilimsel notasyon	Ondalık basamak sayısını verir.
F/f	Gerçek sayılar (float)	Ondalık basamak sayısını verir.
G/g	E ve F biçimlerinden hangisi kısa ise o kullanılır	Ondalık basamak sayısını verir.
N/n	Virgül kullanarak gerçek sayıları yazar	Ondalık basamak sayısını verir.
P/p	Yüzde	Ondalık basamak sayısını verir.
X/x	Onaltılık sayı sisteminde yazar.	En az basamak sayısını belirtir, gereğinde boş olan basamaklar sıfırla beslenir.

Biçimlendirme

- `using System;`
- `class Formatlama {`
- `static void Main() {`
- `float f=568.87f;`
- `int a=105;`
- `Console.WriteLine("{0:C3}",a);`
- `Console.WriteLine("{0:D5}",a);`
- `Console.WriteLine("{0:E3}",f);`
- `Console.WriteLine("{0:F4}",f);`
- `Console.WriteLine("{0:G5}",f);`
- `Console.WriteLine("{0:N1}",f);`
- `Console.WriteLine("{0:P}",a);`
- `Console.WriteLine("{0:P}",1);`
- `Console.WriteLine("{0:X5}",a);`
- `Console.WriteLine("{0:C3}",f); }`

105,000 TL

00105

5,689E+002

568,8700

568,87

568,9

%10.500,00

%100,00

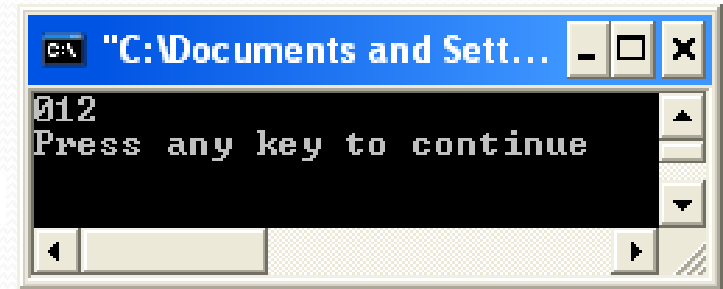
00069

568,870 TL

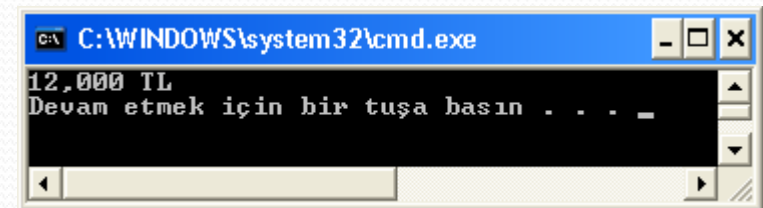
Biçimlendirme

- **String.Format() ve ToString() Metotları ile Biçimlendirme**

- `class StringFormat`
- `{ public static void Main()`
- `{ int a=12;`
- `string str=String.Format("{0:d3}",a);`
- `Console.WriteLine(str);`
- `}`
- `}`

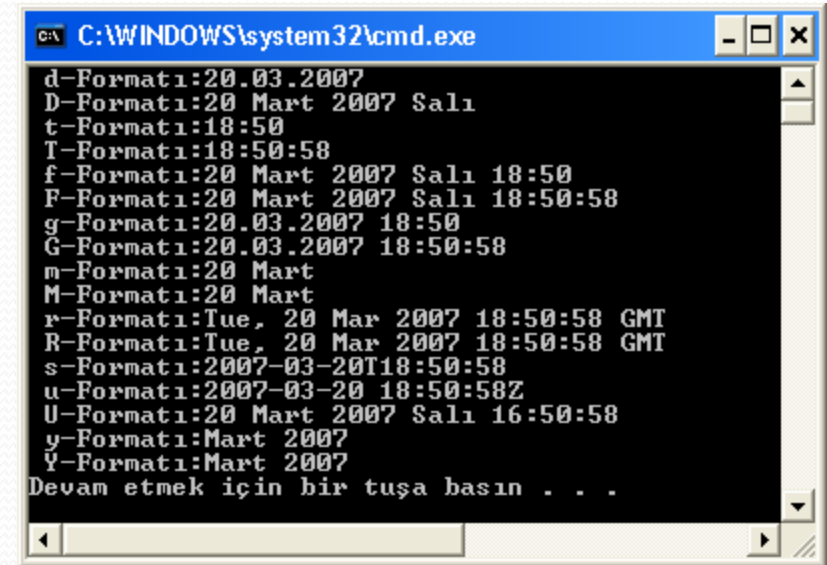


- `class ToString`
- `{ public static void Main()`
- `{ int a=12;`
- `string str=a.ToString("C3");`
- `Console.WriteLine(str);`
- `}`
- `}`



Tarih ve Saat Biçimlendirme

```
• using System;
• class TarihveSaat
• {
•     public static void Main()
•     { DateTime d=DateTime.Now;
•         Console.WriteLine(" d-Formatı:{0:d}",d);
•         Console.WriteLine(" D-Formatı:{0:D}",d);
•         Console.WriteLine(" t-Formatı:{0:t}",d);
•         Console.WriteLine(" T-Formatı:{0:T}",d);
•         Console.WriteLine(" f-Formatı:{0:f}",d);
•         Console.WriteLine(" F-Formatı:{0:F}",d);
•         Console.WriteLine(" g-Formatı:{0:g}",d);
•         Console.WriteLine(" G-Formatı:{0:G}",d);
•         Console.WriteLine(" m-Formatı:{0:m}",d);
•         Console.WriteLine(" M-Formatı:{0:M}",d);
•         Console.WriteLine(" r-Formatı:{0:r}",d);
•         Console.WriteLine(" R-Formatı:{0:R}",d);
•         Console.WriteLine(" s-Formatı:{0:s}",d);
•         Console.WriteLine(" u-Formatı:{0:u}",d);
•         Console.WriteLine(" U-Formatı:{0:U}",d);
•         Console.WriteLine(" y-Formatı:{0:y}",d);
•         Console.WriteLine(" Y-Formatı:{0:Y}",d);
•     }
• }
```



```
C:\WINDOWS\system32\cmd.exe
d-Formatı:20.03.2007
D-Formatı:20 Mart 2007 Salı
t-Formatı:18:50
T-Formatı:18:50:58
f-Formatı:20 Mart 2007 Salı 18:50
F-Formatı:20 Mart 2007 Salı 18:50:58
g-Formatı:20.03.2007 18:50
G-Formatı:20.03.2007 18:50:58
m-Formatı:20 Mart
M-Formatı:20 Mart
r-Formatı:Tue, 20 Mar 2007 18:50:58 GMT
R-Formatı:Tue, 20 Mar 2007 18:50:58 GMT
s-Formatı:2007-03-20T18:50:58
u-Formatı:2007-03-20 18:50:58Z
U-Formatı:20 Mart 2007 Salı 16:50:58
y-Formatı:Mart 2007
Y-Formatı:Mart 2007
Devam etmek için bir tuşa basın . . .
```

Biçimlendirme

- Standart biçimlendirmelerin dışında özel biçimlendirmeler de tasarlanabilir.
- “#” “,” “.” “0” “E” ve “%” karakterleri ile özel biçimlendirmeler oluşturulabilir.
 - # → Rakam değerleri için kullanılır.
 - , → Büyük sayılarda binlikleri ayırmak için kullanılır.
 - . → Gerçek sayılarda ondalıklı kısımları ayırmak için kullanılır.
 - 0 → Yazılacak karakterin başına ya da sonuna 0 ekler.
 - % → Yüzde ifadelerini belirtmek için kullanılır.
 - E0, e0, E+0, e+0, E-0, e-0 → Sayıları bilimsel notasyonda yazmak için kullanılır.
- {0:#,###.##}
- {0:##%}
- {0:#,###E+0}

Biçimlendirme

- `using System;`
- `class özelbiçimlendirme`
- `{`
- `public static void Main()`
- `{ Console.WriteLine("{0:#,###}",1234567);`
- `Console.WriteLine("{0:#.##}",1234.5678);`
- `Console.WriteLine("{0:#,###E+0}",1234567);`
- `Console.WriteLine("{0:#%}",0.123);`
- `}`
- `}`



```
C:\Documents and Settings\SoNDuRaK\...  
1.234.567  
1234,57  
1.235E+3  
12%  
Press any key to continue_
```

Biçimlendirme

- **Düzenli İfadeler**
- Düzenli ifadeler değişken sayıda karakterden oluşabilecek ancak belirli koşulları sağlayan ifadelerdir.
- Örneğin e-posta adreslerini düşünebiliriz. Dünyada milyonlarca e-posta adresi olmasına ve farklı sayıda karakterden oluşabilmesine rağmen hepsi kullanıcıadi@domainismi.domain tipi düzenindedir.
- Örneğin iletisim@microsoft.com bu düzenli ifadeye uymaktadır.
- C#'taki düzenli ifade işlemleri temel olarak System.Text.RegularExpressions isim alanındaki Regex sınıfı ile yapılmaktadır.
- Bir karakter dizisinin oluşturulan düzenli ifadeye uyup uymadığını yine bu isim alanındaki Match sınıfıyla anlarız.
- Düzenli ifadeler başlı başına bir kitap olabilecek bir konudur. Burada sadece ana hatları üzerinde durulacaktır.

Biçimlendirme

- **Düzenli İfadelerin Oluşturulması**
- Bir ifadenin mutlaka istediğimiz karakterle başlamasını istiyorsak **^** karakterini kullanırız. Örneğin **^9** düzenli ifadesinin anlamı yazının mutlaka 9 karakteri ile başlaması demektir. "9Abc" yazısı bu düzene uyarken "f9345" bu düzene uymaz.
- Belirli karakter gruplarını içermesi istenen düzenli ifadeler için **** karakteri kullanılır:
 - **\D** ifadesi ile yazının ilgili yerinde rakam olmayan tek bir karakterin bulunması gerektiği belirtilir.
 - **\d** ifadesi ile yazının ilgili yerinde 0-9 arası tek bir karakterin bulunması gerektiği belirtilir.
 - **\W** ifadesi ile yazının ilgili yerinde alfanumerik olmayan karakterin bulunması gerektiği belirtiliyor. Alfanumerik karakterler a-z, A-Z ve 0-9 aralıklarındaki karakterlerdir.
 - **\w** ifadesi ile yazının ilgili yerinde bir alfanumerik karakterin bulunması gerektiği belirtiliyor.
 - **\S** ifadesi ile yazının ilgili yerinde boşluk veya tab karakterleri haricinde bir karakterin olması gerektiği belirtiliyor.
 - **\s** ifadesi ile yazının ilgili yerinde yalnızca boşluk veya tab karakterlerinden biri bulunacağı belirtiliyor.

Biçimlendirme

- Bu öğrendiğimiz bilgiler ışığında 5 ile başlayan, ikinci karakteri rakam olmayan, üçüncü karakteri ise boşluk olan bir düzenli ifade şöyle gösterilebilir:
 - `^5\D\s`
 - Tahmin edersiniz ki aynı zamanda burada düzenli ifademizin yalnızca 3 harfli olabileceği belirttik. Yukarıdaki `^5\D\s` ifadesine filtre denilmektedir.
- Belirtilen gruptaki karakterlerden bir ya da daha fazlasının olmasını istiyorsak `+` işaretini kullanırız.
 - `\w+`
 - filtresi ilgili yerde bir ya da daha fazla alfanumerik karakterin olabileceğini belirtiyor. "123a" bu filtreye uyarken "@asc" bu filtreye uymaz. `+` yerine `*` kullansaydık çarpıdan sonraki karakterlerin olup olmayacağı serbest bırakılırdı.
- Birden fazla karakter grubundan bir ya da birkaçının ilgili yerde olacağını belirtmek için `|` (mantıksal veya) karakteri kullanılır. Örnek:
 - `m|n|s`
 - ifadesinde ilgili yerde m, n ya da s karakterlerinden biri olmalıdır. Bu ifadeyi parantez içine alıp sonuna `+` koyarsak bu karakterlerden biri ya da birkaçının ilgili yerde olacağını belirtmiş oluruz:
 - `(m|n|s)+`

Biçimlendirme

- Sabit sayıda karakterin olmasını istiyorsak **{adet}** şeklinde belirtiriz. Örnek:
 - **\d{3}-\d{5}**
 - filtresine "215-69857" uyarken "54-34567" uymaz.
- **?** karakteri, hangi karakterin sonuna gelmişse o karakterden en az sıfır en fazla bir tane olacağı anlamına gelir. Örnek:
 - **\d{3}B?A**
 - Bu filtreye "548A" veya "875BA" uyarken "875BBA" uymaz.
- **.** (nokta) işareti ilgili yerde **"\n"** dışında bir karakterin bulunabileceğini belirtir. Örneğin
 - **\d{3}.A**
 - filtresine "123sA" ve "8766A" uyar. "236\nA"; uymaz
- **\b** bir kelimenin belirtilen yazıyla sonlanması gerektiğini belirtir. Örnek:
 - **\d{3}dır\b**
 - filtresine "123dır" uyarken "123dırb" uymaz.

Biçimlendirme

- **\B** karakteri sınırlandırma olmayan durumu belirtir.. Örnek:
- `\d{3}dır\B`
 - filtresine "123dır" veya "dır123" uymazken "123dır8" uyar.
- Köşeli parantezler kullanarak bir karakter aralığı belirtebiliriz. Örneğin ilgili yerde sadece büyük harflerin olmasını istiyorsak **[A-Z]** şeklinde, ilgili yerde sadece küçük harfler olmasını istiyorsak **[a-z]** şeklinde, ilgili yerde sadece rakamlar olmasını istiyorsak **[0-9]** şeklinde belirtebiliriz. Ayrıca sınırları istediğimiz şekilde değiştirebiliriz. Örneğin **[R-Y]** ile ilgili yerde yalnızca R ve Y arası büyük harfler olabileceğini belirtiriz.
- **Regex sınıfı**
- Regex sınıfı bir düzenli ifadeyi tutar. Bir Regex nesnesi şöyle oluşturulur:
- `Regex nesne=new Regex(string filtre);`
- Yani bu yapıcı metoda yukarıda oluşturduğumuz filtreleri parametre olarak veririz. Regex sınıfının **Match** metodu ise kendisine gönderilen bir yazının düzenli ifadeye uyup uymadığını kontrol eder ve uyan sonuçları Match sınıfı türünden bir nesne olarak tutar.

Biçimlendirme

- **Match sınıfı**
- Match sınıfının **NextMatch()** metodu bir Match nesnesindeki bir sonraki düzenli ifadeyi döndürür. Yazının düzenli ifadeye uyup uymadığının kontrolü ise Match sınıfının Success özelliği ile yapılır.
- Eğer düzenli ifadeye uygun bir yapı varsa Success özelliğinin tuttuğu değer true olur.
- Örnek:
- `A\d{3}(a|o)+`
- Bu filtreyle düzenli ifademizin şöyle olduğunu söyleyebiliriz:
- İlk karakter A olacak.
- A'dan sonra üç tane rakam gelecek.
- Üç rakamdan sonra "a" ya da "o" karakterlerinden biri ya da birkaçı gelecek.

Biçimlendirme

- using System;
- using System.Text.RegularExpressions;
- class duzenli
- { static void Main()
- { string filtre=@"A\d{3}(a|o)+";
- Console.Write("Yazı girin: ");
- string yazi=Console.ReadLine();
- Regex nesne=new Regex(filtre);
- Match a=nesne.Match(yazi);
- Console.WriteLine(a.Success);
- Console.WriteLine(a.ToString());
- Console.WriteLine(a.Index);
- Console.WriteLine(a.NextMatch());
- Console.WriteLine(a.Index);
- }
- }

Bu programda kullanıcının

A123aA657oA456oao

girdiğini varsayarsak ekran çıktısı şöyle olur.

True

A123a

0

A657o

10

Visual Studio.Net -C#

9. HAFTA

Kalıtım

Sanal Metotlar

Özet Metotlar

Arayüzler

System.BitConverter

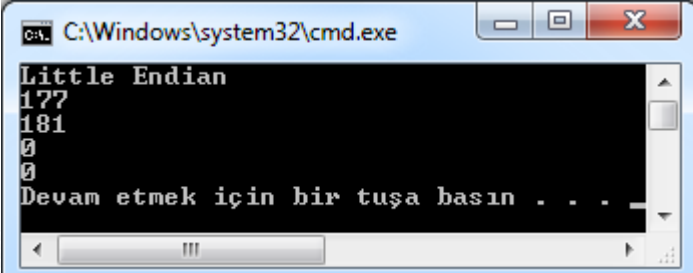
```
using System;

class Program
{
    static void Main()
    {
        if (BitConverter.IsLittleEndian)
            Console.WriteLine("Little Endian");
        else
            Console.WriteLine("Big Endian");

        int a = 46513; // 2560*177+2561*181+2562*0+2563*0=46513

        byte[] b = BitConverter.GetBytes(a);

        foreach (byte x in b)
            Console.WriteLine(x);
    }
}
```



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window displays the output of the C# program: "Little Endian", followed by the bytes "177", "181", "0", and "0" on separate lines. At the bottom, it shows the prompt "Devam etmek için bir tuşa basın . . ." (Press a key to continue).

System.Buffer

Örnek

```
using System;
```

```
class Program
```

```
{
```

```
    static void Main()
```

```
    {
```

```
        byte[] kaynak = { 1, 2, 0, 1 };
```

```
        short[] hedef = new short[5];
```

```
        Buffer.BlockCopy(kaynak, 0, hedef, 0, 4);
```

```
        foreach (short s in hedef)
```

```
            Console.Write(s + " ");
```

```
        Console.WriteLine("\n" + Buffer.GetByte(hedef, 0));
```

```
        Buffer.SetByte(hedef, 5, 3);
```

```
        foreach (short s in hedef)
```

```
            Console.Write(s + " ");
```

```
        Console.WriteLine();
```

```
        Console.WriteLine(Buffer.ByteLength(kaynak));
```

```
        Console.WriteLine(Buffer.ByteLength(hedef));
```

```
    }
```

```
}
```

- Short (2 byte)

- 0.byte

- 1.byte

- 2.byte

- 3.byte

- 4.byte

- 1

- 2*256

- 0

- 1*256

- 0

- 5.byte

- // (SetByte ile 5. byte ekleme)

- 3*256

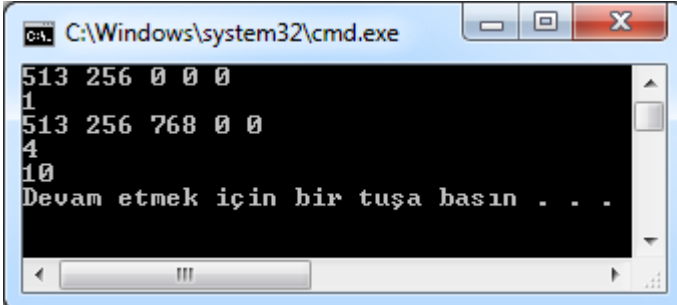
Byte 8 bit

Short 16 bit

$$2*256^1 + 1*256^0 = 513$$

$$1*256^1 + 0*256^0 = 256$$

$$3*256^1 + 0*256^0 = 768$$



```
C:\Windows\system32\cmd.exe
513 256 0 0 0
513 256 768 0 0
Devam etmek için bir tuşa basın . . .
```