BMM 111
Bilgisayar Programlama-I

4. Ders

Dr. Öğr. Üyesi Mustafa İSTANBULLU

Çukurova Üniversitesi
Mühendislik Fakültesi
Biyomedikal Müh. Böl.

E-mail:mm.istanbullu@gmail.com

Not: Slaytlar, kaynakça bölümünde verilen listeden faydalanılarak hazırlanmıştır.

3. Bölüm Atama ve Girdi/Çıktı Komutları

3.2. Atama Komutu

- Bir programda tanımlanan değişkenlerin bellekteki hücrelere verilen isimler oldukları ve bu bellek hücrelerinin programın çalışması sırasında veri saklamak için kullanıldığı daha önce belirtilmişti.
- Değişkenlerin yani bellek hücrelerinin içinde veri saklamak için kullanılan yöntemlerden birisi atama komutları (assignment statements) dır.
- Atama komutu, atama operatörü olan ‘ = ’ kullanılarak yazılabilir.
- Atama komutunun yapısı aşağıdaki gibidir:

```
değişken = ifade;
```

3. Bölüm Atama ve Girdi/Çıktı Komutları

- Bu atama cümlesi ile, önce atama operatörünün sağ tarafında bulunan ifadenin değeri bulunur.
- Bu ifade aritmetik, ilişkisel veya mantıksal bir ifade olabileceği gibi sabit bir değer de olabilir.
- Bu ifadenin sonucu, daha sonra atama operatörünün sol tarafında belirtilen değişkene atanır.
- Şimdi, atama operatörünü daha iyi anlamak için aşağıdaki program parçasını inceleyelim:

```
int x;
x=5;
```

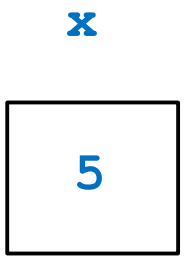
- Bu örnekte ilk komut değişken tanımlama komutudur ve bellekte tamsayı tutabilen **x** isimli değişkenin tanımlanmasını sağlar. Bu anda bellekte bu hücrenin içi boştur: **x**

```
int x;
```

3. Bölüm Atama ve Girdi/Çıktı Komutları

- İkinci komut bir atama komutudur ve komutun sağ tarafındaki bir tamsayı sabiti olan `5`, atama operatörünün solundaki `x` değişkenine atanır. Bu noktadan sonra `x` değişkeninin içinde `5` değeri saklanır.

```
x=5;
```



- Bir değişkenin tanımlaması yapıldıktan sonra, bu değişken için bellekte yer ayrılacağını ve içinin boş olacağını öğrendik.
- Ancak bazı durumlarda bu bellek hücresinde önceden kalmış, istenmeyen değerlerin bulunması söz konusu olabilir ve bu değişken ilk değer ataması yapılmadan bir ifadede kullanılırsa hatalı durumlar oluşabilir.

3. Bölüm Atama ve Girdi/Çıktı Komutları

- Bu nedenle, beklenmeyen hataların olmaması için genellikle tanımlama işleminin hemen ardından ilk değer atama işleminin yapılması önerilir.
- Bu ilk değer atama işlemi ile değişkenin istenmeyen bir değer tutması önlenmiş olur. Şimdi örneklerle değişik tipte verilerin değişkenlere nasıl atandığı incelensin:

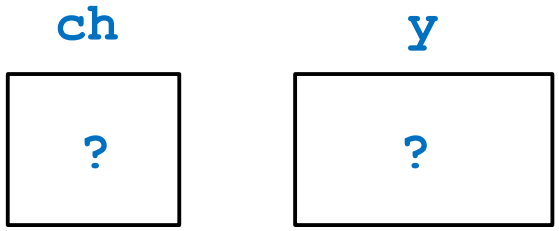
Örnek:

- Aşağıdaki program parçasını inceleyelim.

```
double y;
char ch;
y=10.23;
ch='M';
```

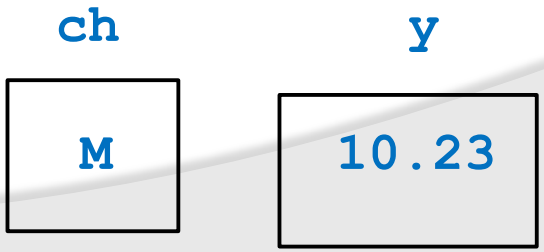
- Burada ilk iki komut ile reel sayı tutabilen `y` ve karakter tutabilen `ch` değişkenleri tanımlanmıştır. Bu noktada bellek görüntüsü aşağıdaki gibi olur.

```
double y;
char ch;
```



- Üçüncü komutta atama operatörünün sağındaki `10.23` sabit değeri, sol taraftaki `y` değişkenine atanır. Son komut ile atama operatörünün sağ tarafındaki `'M'` karakter sabiti, sol tarafta bulunan `ch` değişkenine atanır. Bu noktadan sonra `y` değişkeni `10.23`, `ch` değişkeni ise `'M'` değerini aşağıdaki gibi tutacaktır.

```
y=10.23;
ch='M';
```



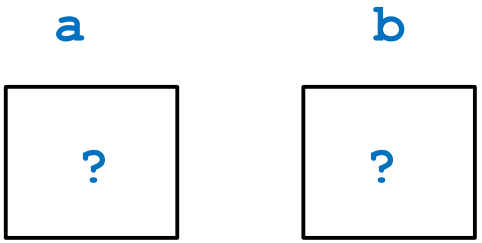
Örnek:

- Aşağıdaki program parçasını inceleyelim,

```
1  int a, b;
2  a=5;
3  b=2+7;
4  a=b+1;
5  b=b+3;
```

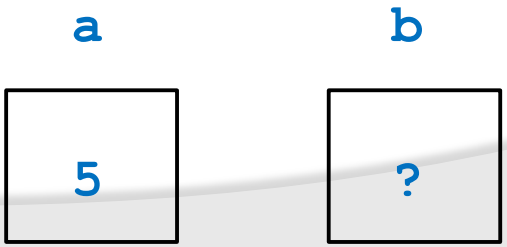
```
1    int a, b;
```

Buradaki 1. satır **a** ve **b** isimli değişkenlerin tanımlanmasını sağlar. Bu noktada bellekteki görünüm aşağıdaki gibidir.



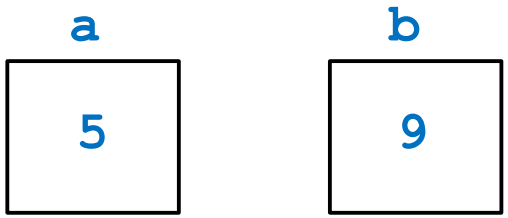

```
2    a=5;
```

Atama komutu olan 2. satır ile, **5** tamsayı sabiti, **a** değişkenine atanır. Bu noktada bellekte **b** değişkeni hala boştur.



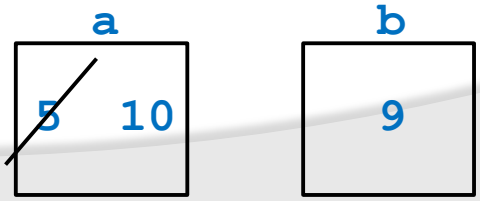
```
3   b=2+7;
```

3. satır da bir atama komutudur, ancak burada atama operatörünün sağ tarafında bir aritmetik ifade bulunmaktadır. Önce bu ifadenin sonucu bulunur. Daha sonra bu sonuç, **b** isimli değişkene atanır.



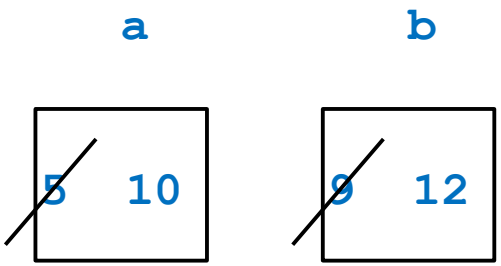

```
4   a=b+1;
```

4. satırdaki atama komutunda operatörün sağ tarafındaki ifadenin sonucu, **b** değişkeninin içinde tuttuğu 9 değerine 1 eklenerek 10 bulunur. Bu sonuç, operatörün sol tarafındaki **a** değişkenine atanır. Dolayısıyla **a** değişkeninin içinde bulunan 5 değeri silinerek yerine 10 değeri atanır.



```
5    b=b+3;
```

5. Satırdaki komutta ise, atama operatörünün sağ tarafında bulunan ifadenin değeri, **b** değişkeninin içinde tuttuğu 9 değerine 3 eklenerek, 12 olarak bulunur. Sonra bu sonuç, solda bulunan **b** değişkeninin yeni değeri olarak kaydedilir. Dolayısıyla bu noktadan sonra **b** değişkeninin değeri olan 9 silinerek yerine 12 değeri atanır.



Örnek:

Aşağıdaki tanımlama cümlelerinin olduğunu kabul edersek, her atama komutu sonrası değişkenlere hangi değerlerin atanacağını bulalım.

```
int k;
double z;
```

Atama Komutu	Değişken Değeri
`k=14;`	k tamsayı değişkenidir. 14 tamsayı sabiti, k'ya atanır.
`k=3.7;`	k tamsayı değişkeni olduğundan 3.7 değerinin ondalık kısmı otomatik olarak atılır ve 3 tamsayı değeri atanır.
`z=5;`	z reel değişkenidir, ancak atama operatörünün sağ tarafında 5 tamsayı sabiti vardır. Bu durumda 5 tamsayısı otomatik olarak reel sayıya dönüştürülür ve 5.0 değeri, z'ye atanır.

3. Bölüm Atama ve Girdi/Çıktı Komutları

Görüldüğü gibi reel bir değer, tamsayı tutan bir değişkene atandığında, reel sayının ondalık kısmı otomatik olarak atılarak bir tamsayıya dönüştürülecektir.

Benzer şekilde bir tamsayı değeri reel sayı tutan bir değişkene atandığında, yine otomatik olarak bu tamsayı bir reel sayıya dönüştürülerek değişkene atanacaktır.

3. Bölüm Atama ve Girdi/Çıktı Komutları

Değişkenlere İlk Değer Ataması

Değişkenlere ilk değerleri tanımlama sırasında da atanabilir. Örneğin, aşağıdaki komutlarla x değişkeni önce tanımlanmış daha sonra değişkene 5 değeri atanmıştır.

```
int x;
x=5;
```

Bu komutlar birleştirilerek, ilk değer atama işlemi, tanımlama sırasında da yapılabilir.

```
int x=5;
```

3. Bölüm Atama ve Girdi/Çıktı Komutları

3.3. Operatörler

- Operatörler, derleyiciye belirli bir matematiksel yada mantıksal işlemi yapması gerektiğini söyleyen tanımlamalardır.
- C dilinde; aritmetik, ilişkisel ve mantıksal operatörler ve bit bit operatörleri olmak üzere dört sınıf operatör bulunmaktadır.

Aritmetik Operatörler

- Aritmetik operatörler, aritmetik işlemlerin yapılması amacıyla kullanılan operatörlerdir. Şimdi, bu operatörlerin nasıl kullanıldıkları incelensin:

Tekli (unary) eksi (-)

- Bu operatör, önüne geldiği sayıyı negatif hale getirir.

```
-3          -9.6
```

Tekli (unary) artı (+)

- Bu operatör, önüne geldiği sayıyı pozitif hale getirir.

```
+5          +7.8
```

Çıkarma Operatörü (-)

- İki değerin birbirinden çıkarılmasını sağlar, sonuç pozitif veya negatif olabilir.

```
13-1 → 12    7-9 → -2    2.9-0.3 → 2.6
```

Toplama Operatörü (+)

- İki değerin toplanmasını sağlar, sonuç pozitif veya negatif olabilir.

```
3+1 → 4    -5+2 → -3    1.1+0.3 → 1.4
```

Çarpma Operatörü (*)

- İki değerin birbiriyle çarpılmasını sağlar, sonuç pozitif veya negatif olabilir.

```
2*3 → 6      -5*2 → -10      2.1*2 → 4.2
```

Bölme Operatörü (/)

- Bir değerin diğer bir değere bölünmesini sağlar, sonuç pozitif veya negatif olabilir.

```
5/2 → 2      -3.0/2 → -1.5      6/2 → 3
```

Mod Operatörü (%)

- İki tamsayı değerinin birbirine bölünmesinden kalan değeri verir. Sadece tamsayı değerleri için tanımlıdır.

```
5%2 → 1      10%3 → 1      4%2 → 0
```

3. Bölüm Atama ve Girdi/Çıktı Komutları

Aritmetik Operatör Kuralları

Aritmetik operatörlerin kullanılması sırasında uyulması gereken bazı kurallar aşağıda belirtilmiştir:

- İki operatör yan yana kullanılamaz. Örneğin (2+/3) geçersiz bir ifadedir.
- İki tamsayı işleminin sonucu tamsayıdır.

```
2+3 → 5      5/2 → 2      2-4 → -2
```

- Bir işlemde sayılardan birisi reel sayı ise sonuç reel sayı olur.

```
2.0+3→ 5.0   5/2.0→2.5   3.0*2.0→ 6.0   2.0-4→ -2.0
```

- İşlem sırası parantez kullanılarak belirtilebilir. Parantez kullanıldığı durumlarda, işlem en içteki parantezden başlar, dışa doğru ilerler.

Parantezlerin olmadığı durumda ise aşağıdaki öncelik tablosu geçerlidir:

Öncelik Sırası	Operatör	Özellik
En Yüksek	( )	İçten dışa
↓	- +	Tekli Operatör (sağdan sola)
	* / %	İkili Operatör (soldan sağa)
En Düşük	+ -	İkili Operatör (soldan sağa)

İkili operatörlerde operatörün sağında ve solunda işlenen (operand) bulunurken; tekli operatörlerde sadece operatörün sağında işlenen bulunur.

Örneğin 6*8 de çarpım operatörü "*" , ikili bir operatördür. Diğer yandan, -35'de yer alan eksi operatörü "-" tekli bir operatördür.

3. Bölüm Atama ve Girdi/Çıktı Komutları

- Az önceki tablodan görüleceği gibi operatörlerin bir arada kullanıldığı ifadelerde, öncelik sırası en yüksek operatör " ( ) " dir. İşlemler öncelik sırası en yüksek operatörden başlar ve daha sonra önceliği daha az olan operatöre geçer. Bu yolla karmaşık işlemlerin yapılması mümkün olur.

- Yine aynı tabloda verilen operatörlerin bir diğer özelliği ise soldan sağa veya sağdan sola işlem görmesidir.

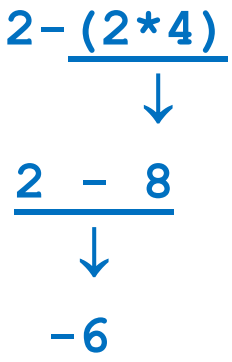
- Bu özellik aynı öncelik sıralamasına sahip operatörlerin ardarda kullanılmasında uygulanır. Örneğin `3+6+10` ifadesinde ardarda toplam operatörü kullanılmıştır. "+" operatörü soldan sağa olduğu için ilk önce soldaki işlem `3+6` yapılır, daha sonra elde edilen sonuç `10` ile toplanır. Dolayısıyla işlem sırası `(3+6)+10` şeklindedir.

Örnek

`(2-(2*4))` işleminin sonucunu bulunuz.

Çözüm

İşleme önce en içteki parantezden başlanır.



Örnek

`2+4/2*3-1` işleminin sonucunu bulunuz.

Çözüm

```
2 + 4 / 2 * 3 - 1
    ↓
2 + 2 * 3 - 1
    ↓
2 + 6 - 1
  ↓
  8 - 1
    ↓
    7
```

Örnek

`9/2*4.2+5/2-1.1` işleminin sonucunu bulunuz.

Çözüm

```
9 / 2 * 4.2 + 5 / 2 - 1.1
  ↓
4 * 4.2 + 5 / 2 - 1.1
    ↓
16.8 + 5 / 2 - 1.1
       ↓
16.8 + 2 - 1.1
     ↓
18.8 - 1.1
     ↓
17.7
```

3. Bölüm Atama ve Girdi/Çıktı Komutları

İlişkisel ve Mantıksal Operatörler

- Değişkenler arasındaki ilişkilerin belirlenmesi amacıyla ilişkisel operatörler kullanılmaktadır. Mantıksal operatörler ise, ilişkisel operatörlü işlemlerin birleştirilmesinde kullanılırlar.
- İlişkisel ve mantıksal operatörlerin kullanımı ile oluşturulan ifadelerin sonucunda hedefimiz doğru ya da yanlış şeklinde bir sonuca ulaşmaktır.
- Yani bu tür operatörlerin kullanıldığı işlemlerde sonuç doğru ya da yanlış olabilir.
- İlişkisel ve mantıksal operatörler şunlardır:

```
>, >=, <, <=, ==, !=, &&, ||, !
```

3.4. İsim Sabitleri

- C dilinde bir isim sabitinin tanımlanması amacıyla

```
# define
```

komutu kullanılır.

- Sabitler değiştirilemediğinden, bu tanıtıcılara değer ataması yapılamaz.
- Bir isim sabitinin genel tanımlama şekli aşağıdaki gibidir:

```
# define sabit_adı değer
```

- Örneğin, PI sayısını isim sabiti olarak tanımlayan komutu yazalım.

```
# define PI 3.1415
```

- Bu komutla, program içinde kullanılmış tüm PI isimli sabitler, 3.1415 değeri ile değiştirilir ve işlemler buna göre yapılır.

- Aşağıdaki isim sabitinin tanımlandığı düşünülsün:

```
# define ORAN 5
```

Buna göre aşağıdaki program parçası incelensin:

```
maas = saat * 30 + ORAN * 10;
pirim = maas * ORAN;
```

Bu komutlardaki tüm ORAN isimleri, önişlemci tarafından `# define` komutunda belirtilen 5 değeriyle değiştirilecektir ve program aşağıdaki hali alacaktır.

```
maas = saat * 30 + 5 * 10;
pirim = maas * 5;
```

3. Bölüm Atama ve Girdi/Çıktı Komutları

3.5. Veri Tipi Dönüşümü

- Bir değişkenin veri tipine tanımlama sırasında karar verilir ve program yürütülmesi sırasında da bu durum değişmez.
- Ancak programın bazı bölümlerinde, değişkenlerin değerlerinin veya sabitlerin veri tiplerinin başka veri tiplerine dönüştürülmesi gerekebilir. Bu durum veri tipi dönüşümü (type casting) olarak adlandırılır.
- Veri tipi dönüşümleri otomatik olarak yapılabildiği gibi bazı durumlarda programcı tarafından tanımlanarak da yapılabil-mektedir.

3. Bölüm Atama ve Girdi/Çıktı Komutları

Otomatik Veri Tipi Dönüşümü

- Otomatik veri tipi dönüşümlerine, atama komutlarından bahsedilirken daha önce değinilmişti.
- Tekrar hatırlatılmak amacıyla, değişken tanımlamalarının ve ilk değerlerinin aşağıdaki şekilde olduğu varsayılsın:

```
double r=0.5, p=5.2, s;
int i=15, q=10, w;
char ch;
```

- Şimdi aşağıdaki her atama komutu için değişkenlerin alacağı değerleri bulalım:

```
s = i / q;
```

3. Bölüm Atama ve Girdi/Çıktı Komutları

- Şimdi aşağıdaki her atama komutu için değişkenlerin alacağı değerleri bulalım:

```
s = i / q;
```

- Bu komutu incelediğimizde, operatörün sağ tarafındaki ifadenin sonucunun `15/10`, yani tamsayı bölümü sonucu `1` olacağını buluruz.

- Bu sonuç, soldaki `s` değişkenine atanacak ve ifadenin sonucu olan `1` tamsayısı, `double` veri tipine otomatik olarak dönüştürülecektir. Dolayısıyla `s` değişkenine `1.0` değeri atanır.

```
w = r * p;
```

- Bu atama komutunda ise, operatörün sağ tarafındaki ifadenin sonucu `(0.5*5.2)` yani `2.6` olarak bulunur ve otomatik olarak veri tipi dönüşümü yapılarak, `2` tamsayı değeri `w` değişkenine atanır.

```
ch = 5*i;
```

- Bu atama komutunun sağ tarafındaki işlem sonucu `(5*15)` bir tamsayıdır ve `75` olarak buluur ve otomatik olarak ASCII tablosuna göre karaktere dönüştürülür. Buna göre `ch`'de saklanan değer `K` karakteri olur.

- Dolayısıyla, atama komutlarını yazarken, C dilinin otomatik veri tipi dönüşümlerini göz önünde bulundurmalıyız.

3. Bölüm Atama ve Girdi/Çıktı Komutları

Tanımlanan Veri Tipi Dönüşümü

- Veri tipi dönüşümünün programcı tarafından tanımlanarak yapılması da mümkündür.
- Bu amaçla veri tipi dönüşümü yapılması istenen değişkenin ya da değerin önünde istenen (veri tipi) tanımlaması kullanılır.

```
(istenilen_veri_tipi) değişken_ismi
```

- Örnek olarak şu komutlar incelensin:

```
int sayi1, sayi2;
double bolum;
sayi1=2;
sayi2=4;
```

- Şimdi aşağıdaki her komut sonrasında **bolum** değişkeninde hangi değerin saklanacağını bulalım.

```
bolum=sayi1/sayi2;
```

- Önce atama operatörünün sağ tarafındaki işlem yapılır. Bu bir tamsayı bölümüdür ve sonuç `0` dır.
- Bu sonuç `bolum` değişkenine atanırken otomatik olarak reel sayıya dönüştürülerek `0.0` değeri bolum değişkenine atanır.

```
bolum=(double)sayi1/(double)sayi2;
```

- Yukarıdaki komutu incelediğimizde, `sayi1` ve `sayi2` değişkeninin önünde dönüştürülmesi istenen veri tipi belirtilmiştir.
- Bu şu anlama gelir: `sayi1` değişkeninin değeri reel sayıya dönüştürülerek kullanılacak, aynı şekilde `sayi2` değişkeninin değeri reel sayıya dönüştürülerek kullanılacaktır.
- Buna göre sağ taraftaki ifadenin sonucu `2.0/4.0` yani `0.5` olacak ve bu değer `bolum` değişkenine atanacaktır.

```
sayi1=(int) 3.6;
```

- Yukarıdaki komutta ise atama operatörünün sağ tarafındaki `3.6` sabiti önce tamsayıya dönüştürülecek (yani `3` değerine), daha sonra bu değer `sayi1` değişkenine atanacaktır.

Örnek

- Aşağıdaki program parçasının sonucunda `reelSayi` değişkeninde hangi değer tutulur?

```
int tamSayi=10;
double reelSayi;
reelSayi=(double)tamSayi;
```

Çözüm

```
10.0
```

Örnek

- Aşağıdaki program parçasının sonucunda `reelSayi` değişkeninde hangi değer tutulur?

```
int tamSayi;
char harf='B';
tamsayi=(int)harf;
```

Çözüm

Bu program parçasında, int veri tipinde tanımlanmış olan `tamSayi` değişkeninin içine, karakter olarak tanımlanmış `harf` değişkeninin değerinin tamsayıya dönüştürülerek atanması söz konusudur.

Buna göre `harf` değişkeninin ASCII kod tablosundaki karşılığı tamsayı olarak `tamSayi` değişkenine atanır. Sonuç `66` dır.

3. Bölüm Atama ve Girdi/Çıktı Komutları

7. Hafta

3.6. Çıktı Fonksiyonu – `printf()`

- **`printf()`** fonksiyonuna daha önce kısaca değinilmişti.
- **`printf()`** fonksiyonu, program sonuçlarının ekranda gösterilmesini sağlayan bir kütüphane fonksiyonudur.
- **`printf()`** fonksiyonu yürütüldüğünde ekranda yeni bir pencere açılacak ve sonuçlar bu pencerede gösterilecektir.
- Bu fonksiyonun bulunduğu programlarda, **`printf()`**'in içinde bulunduğu **`<stdio.h>`** dosyası **`# include`** komutu ile program başında belirtilmelidir.

3. Bölüm Atama ve Girdi/Çıktı Komutları

`printf()` fonksiyonu

`printf()` fonksiyonunun iki farklı yapısı vardır. Bunlardan ilkinin genel yapısını görelim:

```
printf("format dizgisi");
```

- Bu komutta görülen " " işaretleri arasındaki format dizgisi olduğu gibi ekranda gösterilecektir. Örneğin aşağıdaki komut ile

```
printf("Bu bir ciktidir.");
```

- Ekranda şu çıktı görülecektir:

```
Bu bir ciktidir.
```

Örnek

- Aşağıdaki komutları inceleyerek program çıktısını bulunuz.

```
1   # include <stdio.h>
2   int main(void)
3   {       printf("gecen ogrenci sayisi");
4           printf("=30,");
5           printf(" kalan ogrenci sayisi=");
6           printf("10");
7           return(0);
8   }
```

Çözüm

```
gecen ogrenci sayisi=30, kalan ogrenci sayisi=10
```

Örnek

- Aşağıdaki komutları tek **printf** komutu olarak yazınız.

```
printf("abc");
printf("3040");
printf("def");
printf("2030");
```

Çözüm

Bu komutların yürütülmesiyle aşağıdaki çıktı elde edilir.

```
abc3040def2030
```

- Eğer kullanıcı, çıktıların ayrı satırlarda gösterilmesini istiyorsa özel bir karakter olan " \ ", yeni satır (new line) karakteri kullanılmalıdır.

- **printf()** komutu içinde " **\n** " nerede bulunursa, çıktı o noktadan sonra yeni satıra geçer. Örneğin,

```
printf("Bu 1. satir.\nBu 2. satir.");
```

ise ekranda görülecek çıktı aşağıdaki gibidir:

```
Bu 1. satir.
Bu 2. satir.
```

Örnek

- Aşağıdaki program parçasının çıktısı nedir?

```
printf("gecen ogrenci sayisi");
printf("=30\n");
printf("kalan ogrenci sayisi=");
printf("10");
```

Çözüm

```
gecen ogrenci sayisi=30
kalan ogrenci sayisi=10
```

3. Bölüm Atama ve Girdi/Çıktı Komutları

Değişkenler ve `printf()`

`printf()` komutunun ikinci yapısı, değişkenlerin veya ifadelerin değerlerinin ekranda gösterilmesini sağlar.

Bunun için kullanmamız gereken **`printf()`** fonksiyonunun genel yapısı aşağıdaki gibi olur.

```
printf("format_dizgisi", çıktı_listesi);
```

buradaki ***`çıktı_listesi,`*** ekranda içeriğinin gösterilmesini istediğimiz değişken veya ifadelerin listesidir. ***`format_dizgisi`*** ise bu değerlerin nasıl gösterileceğini tanımlar.

- Örneğin, `75` değerini tutan `x` isimli bir tamsayı değişkeninin içeriğini ekranda göstermek istiyorsak,

```
printf("%d", x);
```

komutunu kullanabiliriz.

- Bu komut ile çıktı listesinde bulunan `x` değişkeninin içindeki değer, format dizgisi içinde bulunan `%d` tanımlamasının olduğu yere konulup ekranda gösterilir.
- `%d` işareti ile belirlenen ve çıktı listesinde bulunan değişken veya ifadelerin içeriklerinin format dizgisi içinde nereye konularak gösterileceğini tanımlayan belirleyiciler yer belirleyici (placeholder) olarak adlandırılır.
- Buna göre çıktı aşağıdaki gibi olacaktır:

75

Örnek

- `x` tamsayı değişkeni `5` değerini tutuyorsa aşağıdaki komutun çıktısını bulalım.

```
printf("x=%d", x);
```

Çözüm

Burada önce `" "` arasındaki `x=` karakterleri ekranda gösterilir.

Daha sonra çıktı listesinde bulunan `x` değişkeninin değeri `%d` ile gösterilen yere konulup ekranda gösterilerek aşağıdaki çıktı elde edilir.

```
x=5
```

Örnek

- `a` değişkeni `10` değerini tutuyorsa, aşağıdaki komutların çıktısı nedir?

```
printf("a degiskeni %d\n", a);
printf("%d", a*2);
```

Çözüm

Bu komutlar yürütüldüğünde aşağıdaki satırlar ekranda gösterilecektir.

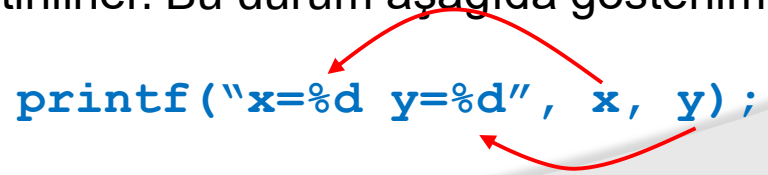
```
a degiskeni 10
20
```

3. Bölüm Atama ve Girdi/Çıktı Komutları

- **`printf()`** komutu ile birden çok değişkenin içeriklerinin ekranda gösterilmesi istenebilir.
- Örneğin, `x` ve `y` isimli iki tamsayının değerleri sırasıyla `5` ve `10` olsun. Bu değişkenlerin değerlerini ekranda göstermek için şu komutu yazalım:

```
printf("x=%d y=%d", x, y);
```

- Birden fazla değişkenin içeriği ekranda gösterilmek isteniyorsa, bu değişkenler yukarıdaki komutta olduğu gibi, çıktı listesinde virgülle ayrılarak gösterilmelidirler.
- Bunun yanında, her değişken için, format dizgisinde bir yer belirleyici belirtilmelidir.
- Çıktı listesindeki değişkenler, format dizgisindeki yer belirleyiciler ile soldan sağa doğru eşleştirilirler. Bu durum aşağıda gösterilmektedir:



- Dolayısıyla ekran çıktısı **`x=5 y=10`**

Örnek

- `a` değişkeninin değeri `10` ise aşağıdaki `printf()` komutunun çıktısı nedir?

```
printf("a=%d\nb=%d", a, 2*a);
```

Çözüm

Bu komut yürütüldüğünde aşağıdaki satırlar ekranda gösterilir.

```
a=10
b=20
```

3. Bölüm Atama ve Girdi/Çıktı Komutları

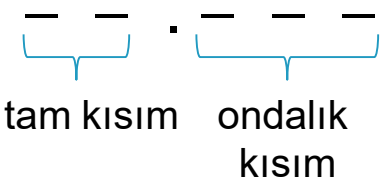
Şimdi aşağıdaki örneği inceleyelim:

Örnek

```
printf("%6.3f", 12.236);
```

Çözüm

Burada yer belirleyicinin önüne konan `6.3` sayısı, bu sayıyı gösterirken toplam 6 kolonluk yer kullanılacağını, ondalık kısım içinse bu alanın son 3 kolonunun kullanılacağını belirtir. Reel sayıları gösterirken nokta `'.'` bir kolon olarak algılanır.



Buna göre sayının gösterimi şöyle olacaktır:

```
 12.236
```

3. Bölüm Atama ve Girdi/Çıktı Komutları

- **`printf()`** fonksiyonu farklı veri tipleri için aşağıda verilen farklı yer belirleyicilerin kullanılmasını gerektirir. **`printf()`** fonksiyonunda kullanılan yer belirleyiciler aşağıdaki tabloda özetlenmiştir.

Yer Belirleyici	Veri Tipi
`%c`	Karakter
`%d`	Tamsayı
`%e`	Bilimsel gösterim (scientific notation)
`%f (float)` **`%lf (double)`**	Reel sayı (decimal, floating point)
`%g`	`%e` ve `%f`' den hangisi daha kısa ise onu kullanır
`%s`	Dizgi (string)
`%u`	İşaretsiz Ondalık (unsigned decimal)
`%x`	Hexadecimal

Örnek: Aşağıdaki program parçasının çıktısı nedir?

```
int x=3; double y=3.2;

printf("x=%d, y=%f",x,y);
```

Komutlar yürütüldüğünde aşağıdaki satırlar ekranda gösterilecektir:

```
x=3, y=3.200000
```

Burada `y` değişkeni için `%f` yer belirleyicisi kullanılmıştır, çünkü `y`, `double` olarak tanımlanmıştır.

Çıktı incelendiğinde, `3.2` değerinden sonra 5 tane 0 değeri konulmuştur.

Bu gösterim, reel sayıların standart `%f` gösteriminde otomatik olarak derleyici tarafından yapılır.

Yani bir reel sayının ondalık kısmı 6 haneli olacak şekilde 0 değeri otomatik olarak eklenerek gösterilir.

Örnek:Aşağıdaki program parçasının çıktısı nedir?

```
char ch='B';

printf("%c\n",ch);

printf("%d",ch);
```

Komutlar yürütüldüğünde aşağıdaki satırlar ekranda gösterilecektir:

```
B
66
```

Burada ilk `printf` fonksiyonu `ch` değişkeninin içeriği olan `B` karakterini göstererek, bir alt satıra geçecektir.

İkinci `printf` fonksiyonu ise, yine `ch` değişkeninin içeriği olan `'B'` karakterini gösterecektir. Ancak, burada yer belirleyici olarak `%d` kullanıldığından, `'B'` 'nin ASCII tablosundaki karşılığı olan `66` sayısı ekranda gösterilecektir.

3. Bölüm Atama ve Girdi/Çıktı Komutları

Formatlı Çıktı

`printf();` çıktının, kullanıcının istediği gibi görünmesini sağlayacak şekilde değiştirilebilir. Örneğin, aşağıdaki `printf()` komutunu ve çıktısını inceleyelim.

```
printf("%f", 72.62);
```

komutunun çıktısı şöyle olacaktır:

```
72.620000
```

Görüldüğü gibi sayının ondalık kısmı otomatik olarak 6 basamak olacak şekilde ekranda gösterilmiştir. Programcı, sonucun aşağıdaki şekilde görünmesini de isteyebilir.

```
72.62
```

Böyle bir gösterimin `printf()` komutunda belirtilmesi gereklidir. Bunun için değişik veri tipleri için farklı tanımlamalar yapmamız gereklidir.

`int` Veri Tipi

Bir tamsayının ekranda kullanıcının istediği gibi gösterilmesi için yer belirleyici şu şekilde değiştirilmelidir:

`%d` yerine `%nd`

burada `n` bir tamsayıdır. `n`, tamsayının gösterilmesi sırasında ekranda kaç kolonluk yer kullanılacağı belirtilir.

Örnek

```
printf("%4d", 33);
```

Çözüm

Bu komutta yer belirleyicinin önüne konan `4` sayısı, `33` sayısını ekranda gösterirken 4 kolonluk yer kullanılmasını belirtir.

Sayımız 2 basamaklı olduğu halde, sayıyı göstermek için kullanılması istenen yer 4 kolon olduğundan, bu alanda sayı sağa yanaşık olarak gösterilir. Yani ekranda, □□33

görülecektir. Buradaki '□' işareti bir karakterlik boşluk anlamına gelir.

`char` Veri Tipi

Bir karakterin ekranda kullanıcının istediği gibi gösterilmesi için yer belirleyici şu şekilde değiştirilmelidir:

`%c` yerine `%nc`

burada `n` bir tamsayı olup, karakterin gösterilmesi sırasında ekranda kaç kolonluk yerin kullanılacağını belirtir.

Örnek

```
printf("%3c", 'M');
```

Çözüm

␣␣M

`string` Veri Tipi

Bir dizginin ekranda kullanıcının istediği gibi gösterilmesi için yer belirleyici şu şekilde değiştirilmelidir:

`%s` yerine `%ns`

burada `n` bir tamsayıdır. `n`, dizginin gösterilmesi sırasında ekranda kaç kolonluk yerin kullanılacağını belirtir.

Örnek

```
printf("%10s", "Merhaba");
```

Çözüm

□□□`Merhaba`

`double` Veri Tipi

Bir reel sayının ekranda kullanıcının istediği gibi gösterilmesi için yer belirleyici şu şekilde değiştirilmelidir:

`%f` yerine `%n.mf`

burada `n` ve `m` birer tamsayıdır. `n`, reel sayının gösterilmesi sırasında ekranda toplam kaç kolonluk yerin kullanılacağını belirtir. `m` ise, reel sayının ondalık kısmı için kaç kolonluk alan kullanılacağını belirtir.

Örnek

```
printf("%f", 12.236);
```

Çözüm

Yukarıdaki komut, ekranda bu sayının ilk kolondan başlayarak, ondalık kısım için standart 6 kolonluk yer kullanılmasını belirtir.
Sayının ondalık kısmı 3 basamak olduğundan, bunu 6 basamağa tamamlamak için sonuna 0'lar otomatik olarak yerleştirilir.

Ekran görüntüsü şöyledir:

```
12.236000
```

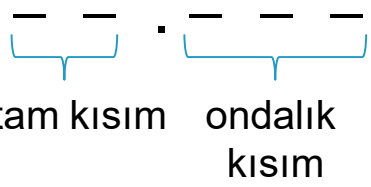
Şimdi aşağıdaki örneği inceleyelim:

Örnek

```
printf("%6.3f", 12.236);
```

Çözüm

Burada yer belirleyicinin önüne konan `6.3` sayısı, bu sayıyı gösterirken toplam 6 kolonluk yer kullanılacağını, ondalık kısım içinse bu alanın son 3 kolonunun kullanılacağını belirtir. Reel sayıları gösterirken nokta `'.'` bir kolon olarak algılanır.



Buna göre sayının gösterimi şöyle olacaktır:

```
12.236
```

3. Bölüm Atama ve Girdi/Çıktı Komutları

Bilimsel Gösterim

Reel sayıları göstermek için bilimsel gösterimin de kullanılabileceğini öğrenmiştik.
Şimdi bilimsel gösterimde formatlı çıktının nasıl olacağını inceleyelim.
Bu gösterimin formatlı çıktısı için yer belirleyicisi aşağıdaki şekilde belirtilmelidir.

`%e` yerine `%n.me`

Burada `n`, sayının gösterilmesi sırasında ekranda toplam kaç kolonluk yerin kullanılacağını belirtir. `m` ise, ondalık kısım için kaç kolonluk alan kullanılacağını belirtir.
Bu gösterim `n` kolonluk alanın son 4 kolonu standart olarak üs gösterimi için kullanılır.

Örnek

```
printf("%10.3e", -0.0536);
```

Çözüm

Önce sayı bilimsel gösterime dönüştürülür. Bunun için nokta, sayının ilk sıfırdan farklı basamağının önüne getirilmelidir. Burada nokta 2 kez sağa kaydırılmalıdır. Bu durumda sayımız 5.36×10^{-2} olmuştur.

Bu noktada sayımız, `%10.3e` yer belirleyicisine uygun olarak gösterilecektir. Toplam kolon sayısı 10 dur. 3 ise ondalık kısım için kaç basamak kullanılacağını gösterir. Standart olarak son 4 kolon üs gösterimi için kullanılacaktır ($10^{-2} \rightarrow$ `e-02`).

Burada sayının işareti olan `-` bir kolon olarak algılanır.

```
-5.360e-02
```

- Formatlı gösterimde yer belirleyicinin önüne pozitif ya da negatif sayılar yazarak ekran görüntüsü farklılaştırılabilir.
- Bu durum aşağıdaki örnekle incelensin:

```
int sayi=44;
printf("%-12d \n", sayi);
printf("%012d \n", sayi);
```

- İlk `printf()` fonksiyonu, ekranda gösterilmesi istenen sayı için toplam 12 karakterlik yer ayrılması gerektiğini ve sayının bu alanda sola dayalı olarak yerleştirilmesi gerektiğini söyler.
- İkinci `printf()` fonksiyonu, 12 kolonluk alanda sayının sağa dayalı olarak yerleştirildikten sonra geride kalan boşlukların sıfır ile doldurulmasını sağlar.
- Bu program parçasının ekran çıktısı aşağıdaki gibi olacaktır:

```
44
000000000044
```

3. Bölüm Atama ve Girdi/Çıktı Komutları

Ters Eğik Çizgi Karakter Sabitleri (\ )

C dilindeki ters eğik çizgi karakter sabitleri özel amaçlarla kullanılmak üzere tanımlanmışlardır.

Daha önce `'\n'` karakter sabitinin yeni satıra geçmek için kullanıldığını belirtmiştik. `'\'` ile kullanılan diğer karakter sabitlerinin listesi aşağıdaki tabloda verilmiştir.

Kod	Açıklama
`\b`	Geriye doğru boşluk (backspace)
`\f`	Form besleme (form feed)
`\n`	Yeni satır
`\r`	Satır başı (carriage return)
`\t`	Sekme (horizontal tab)
`\'`	Tek tırnak karakteri
`\0`	Boş (null)

Örnek

Aşağıdaki komut satırında, geriye doğru boşluk **(\b)** ters eğik çizgi karakteri kullanılmıştır.

Bu tanımlamada, **(\b)** noktasından sonra imleç geriye doğru bir boşluk sildikten sonra format dizgisinin geri kalan kısmının basılması işlemine devam eder.

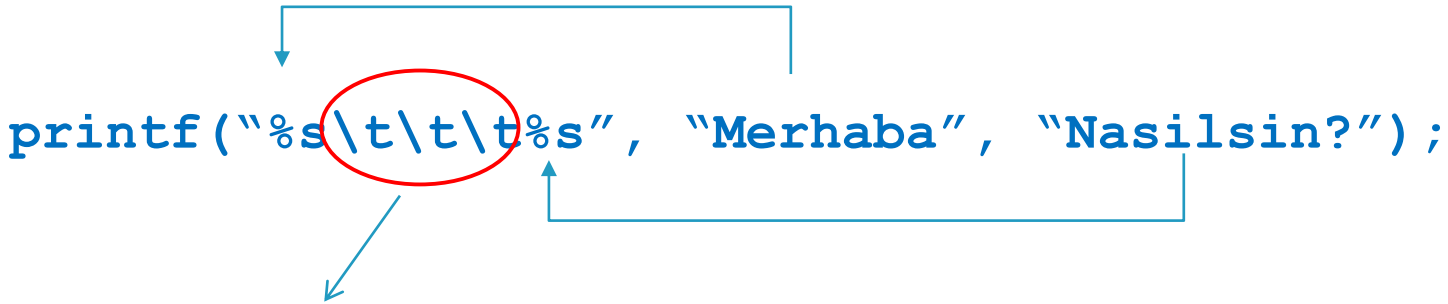
Çıktı:

```
MerhabNasilsin?
```

Örnek

Şimdi de sekme **(\t)** test eğik çizgi karakterinin kullanımını inceleyelim.

(\t) noktasından sonra imleç bir sekme uzunluğu kadar ileriye atlar ve diğer tanımlamaların basılması işlemine devam eder.



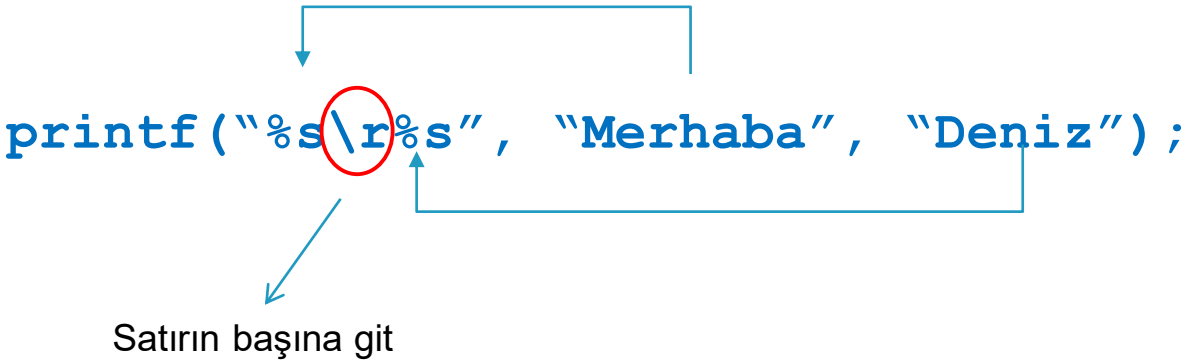
Çıktı:

```
Merhaba                    Nasilsin?
```

Örnek

Şimdi de satır başı **(\r)** test eğik çizgi karakterinin kullanımını inceleyelim.

Bu tanımlamada, **(\r)** noktasından sonra imleç aynı satır üzerinde satır başı yapar ve diğer tanımlamanın gösterilmesi işlemine engel olur.



Çıktı:

Deniz

3.7. Girdi Fonksiyonu – `scanf()`

- **`scanf()`** fonksiyonu kullanıcı tarafından veri girişinin yapılmasına ve bu verilerin girdi listesinde belirtilen değişkenlerde saklanmasını sağlayan bir fonksiyondur. Bu fonksiyonun genel kullanımı aşağıdaki gibidir:

```
scanf("format_dizgisi", girdi_listesi);
```

- ***format_dizgisi*** içinde değeri kullanıcı tarafından girilecek her değişken için yer belirleyicilerin belirtilmesi gereklidir.

- ***girdi_listesi*** ise, içine veri girilecek değişkenlerin listesidir.

Örneğin, kullanıcının girdiği bir tamsayı değerinin toplam isimli bir değişkende saklanmasını istiyorsak, şu komutu yazmalıyız.

```
scanf("%d", &toplam);
```

Burada görülen **&** operatörü, adres operatörüdür.

- Bu durumda yukarıdaki komut, kullanıcı tarafından girilen tamsayı değerini, `toplam` değişkeninin adresinde saklayacaktır.
- Kullanıcı bu değeri klavye yoluyla girdikten sonra ‘enter’ tuşuna basmalıdır. Böylece klavyeden girilen değerin istenen değişkende saklanması sağlanır.
- `scanf()` fonksiyonunda farklı veri tipleri için kullanılacak yer belirleyiciler, daha önce verilen tablodaki yer belirleyiciler ile aynıdır.
- Birden fazla değişkene girdi yapılıyorsa, her değişken için bir yer belirleyici belirtilmeli ve bu değişkenlerin girdi listesinde virgülle ayrılarak belirtilmelidir.
- Kullanıcı, klavyeden girilecek birden fazla sayıyı birbirinden ayırmak için en az bir boşluk veya tab bırakmalıdır.

Örnek

```
int x;  double y;
scanf("%d", &x);
scanf("%lf", &y);
```

örnek girdi aşağıdaki gibi olsun:

```
6.1 ↵
5 ↵
```

Çıktı:

Burada ise ilk `scanf()`, tamsayı beklerken `6.1` reel sayısı girilmiştir. Bu durumda `scanf()` bunu iki sayının ardarda yazılmış hali olarak algılar ve `x`'e `6`, `y`'ye ise `0.1` değerini atar. İkinci girdi bu `scanf()` fonksiyonları tarafından kullanılmayacaktır.

Tablo 3.4 Bazı Matematik Kütüphane Fonksiyonları

Fonksiyon	Açıklaması
`cos(x)`	x parametresi ve fonksiyonun sonucu `double` veri tipindedir. Verilen x değerinin kosünüs karşılığını hesaplar.
`log(x)`	x parametresi ve fonksiyonun sonucu `double` veri tipindedir. Verilen x değerinin logaritma karşılığını hesaplar.
`pow(x,y)`	x, y parametreleri ve fonksiyonun sonucu `double` veri tipindedir. Verilen x değerinin x^y değerini hesaplar.
`sqrt(x)`	x ve fonksiyonun sonucu `double` veri tipindedir. Verilen x değerinin kara kökünü hesaplar.

Örnek:

```
sonuc=sqrt(a*b-c/6.0);
deger=pow(p*q, 5.0)
```

KAYNAKÇA

- Prof.Dr. İbrahim DEVELİ, Bilgisayar Programlama Ders Notları, Erciyes Üniv. Elektrik-Elektronik Müh. Böl.
- H.Turgut UYAR, Programlamaya Giriş Ders Notları,İTÜ, 2004.
- Fedon KADİFELİ,Standart C Programlama Dili, (Tercüme),2000.
- Doç. Dr. Soner ÇELİKKOL, Programlamaya Giriş ve Algoritmalar, Murathan Yayınevi, TRABZON; 2009
- N. Ercil Çağıltay ve ark., C DERSİ PROGRAMLAMAYA GİRİŞ, Ada Matbaacılık, ANKARA; 2009.
- Çeşitli kişilerin internette paylaşıma açtığı notlardan faydalanılmıştır.