Veri Tipleri

C programlama dilinde, her türlü bilgi veya sayı "**veri**" olarak adlandırılır. Veriler, değişkenlere ya da sabitlere atanan büyüklüklerdir ve bu değerler, bir programın işleyişinde büyük önem taşır. Veriler, bellekte nasıl tutulduklarına ve hangi işlemlere tabi tutulabileceklerine bağlı olarak çeşitli sınıflara ayrılır. Her veri, atandığı değişkenin veya sabitin türüne göre işlem görür. Bu sayede, hangi veri türünün hangi işlemleri gerçekleştirebileceği ve ne kadar bellek alanı kaplayacağı belirlenmiş olur.

Veri Tipi	Veri Boyutu	Değer Aralığı (min- max)
unsigned short int	2 byte	0 – 65 535
short Int	2 byte	-32 768 – +32 767
unsigned long Int	4 byte	0 – 4 294 967 295
long int	4 byte	-2 147 483 648 – + 2 147 483 647
int (16 bit)	2 byte	-32 768 – + 32 767
int (32 bit)	4 byte	-2 147 483 648 – + 2 147 483 647
unsigned int (16 bit)	2 byte	0 – 65 335
unsigned int (32 bit)	4 byte	0 – 4 294 967 295
Char (256 karakter)	1 byte	0 – 255
float	4 byte	1.2 e -38 – 3.4 e +38
double	8 byte	2.2 e -308 – 1.8 e 308

Veri tipleri, bellekte farklı alanlar kaplar ve her veri tipi belirli sınırlar içerisinde çalışır. Örneğin, bir int veri tipi tam sayıları tutarken, bir float veri tipi ondalıklı sayıları saklar. Bunun yanı sıra, char veri tipi bir karakteri tutarken, bir double veri tipi daha hassas ondalıklı sayıları depolar. Bu veri türlerinin her biri, programın işleyişine ve bellek kullanımına farklı etkiler sağlar. Verilerin türüne uygun olarak tanımlanması, programların verimli çalışmasını olanak sağlar.

Değer Atama

C programlama dilinde bir değişkene, program içinde değiştirilmediği sürece aynı kalacak ve istenildiği zaman değiştirilebilecek şekilde bir değer verme işlemine **"atama"** denir. Bu işlem bir değer atama komutu, **" = "** atama operatörü veya **scanf()** atama fonksiyonu ile gerçekleştirilir. scanf() atama fonksiyonu ilerleyen bölümlerde ayrıca ele alınacaktır. Atama operatörü "=" ile atama işlemi değişken tanımlanırken yapılabileceği gibi, program içinde gerekli olan yerlerde de yapılabilir veya kendisine bir değer atanmış bir değişkenin değeri değiştirilebilir. Değişkene değer atama işlemlerinde şu kurallar geçerlidir;

- Değişken tanımlanırken atama yapılabilir ve bu işleme **"ilk değer atama"** denir. Örneğin;

```
int x;
x=9;
```

şeklinde verilen işlemde **x** değişkeni önce tanımlanıp sonra değer atanıyor.

- Bu komutlar birleştirilerek aşağıdaki örnekte olduğu gibi tanımlama sırasında da ilk değer atanabilir.

```
int x=9;
```

- Metin türü değişkenlere değer atarken, verinin aşağıdaki gibi iki tek tırnak (') işaretlerinin arasında yazılması gerekir.
- Aynı anda birden fazla değişkene değer atanabilir.

```
char k='A';
char y='H', d='O', b='X';
```

- Atama işleminde mutlaka soldan sağa doğru **değişken-operatör-değer** sıralaması vardır. Örneğin **a=14**; atamasında **a** değişkeninin karşılığı **14** olur.
- **a=2+4**; gibi bir atamada, **a** değişkeninin karşılığı olarak sağ taraftaki işlem (2+4=**6**) hesaplanıp, sonuç olarak soldaki **a** değişkenine atanır.

Burada, atama operatörü olarak kullanılan "=" sembolünün, matematikteki eşitlik anlamına gelmediği, sadece bir aktarma işlemi için kullanıldığı unutulmamalıdır.

"=" atama operatörünün farklı kullanımı

"=" atama operatörü, aritmetik operatörler **"+, -, *, /, %"** ile birlikte kullanılabilmektedir. Bunun için aritmetik operatörlerden her hangi biri "=" atama operatöründen önce yazılır. Bu durumda değişken verisi üzerinde önce tanımlanan işlem yapılır daha sonra atama gerçekleştirilir. "=" atama operatörü ve aritmetik operatörlerin kullanım şekilleri aşağıdaki tabloda örnekleri ile birlikte gösterilmiştir.

Operatör	Açıklama	Örnek	Anlamı
=	normal atama	a = 6	a = 6
+=	ekleyerek atama	a += 2	a = a - 2
-=	eksilterek atama	a -= 7	a = a - 7
*=	çarparak atama	a *= 6	a = a * 6
/=	bölerek atama	a /= 3	a = a / 3
%=	bölüp kalanını atama	a %= 5	a = a % 5

"=" atama operatörü ve aritmetik operatörlerin birlikte kullanımı hesaplama ve atama işlemlerini kısalmaktadır. Ayrıca kod yazımı da daha kısa ve kontrolü daha kolay hâle gelmektedir. Aritmetik operatörlü atama işlemlerinde sadece sabit sayıların kullanılması söz konusu değildir. Atama işleminde değişken değerler de kullanılabilir.

```
{
    int a = 5, c = 3;

    a += 1;     // a = 6
    c /= a;     // c = 2  değerini alır
}
```

Değişken bildirimlerinde dikkat edilmesi gereken durumlar

C dilinde eğer değişken bildirimi blokların içinde yapılacaksa, bildirim işlemi blokların ilk işlemi olmak zorundadır. Başka bir deyişle bildirimlerden önce, bildirim deyimi olmayan başka bir işlemin bulunması geçersizdir. Örneğin;

```
{
	int var1, var2;
	char ch1, ch2, ch3;
	var1 = 10;
	float f; /* Geçersiz */
}
```

Bu örnekte var1, var2, ch1, ch2, ch3 değişkenlerinin tanımlanma yerleri doğrudur. Ancak f değişkeninin bildirimi geçersizdir. Çünkü bildirim deyiminden önce bildirim deyimi olmayan başka bir deyim "var1 = 10;" yer alıyor. Bu durum alttaki bildirimi geçersiz kılar. Aynı program parçası şu şekilde yazılmış olsaydı bir hata söz konusu olmazdı.

```
{
	int var1, var2;
	char ch1, ch2, ch3;
	var1 = 10;
	{ float f; }
}
```

Bu durumda artık f değişkeni de kendi bloğunun başında tanımlanmış olur. Her ";" sembolü ve boş bloklar bir yeni satırı oluşturmaktadır. Bu nedenle C sözdizimine göre oluşan bu deyim yürütülebilir bir deyimdir. Dolayısıyla aşağıdaki kod parçalarında y değişkeninin tanımlaması derleme geçersizdir.

```
{
	int x;;
	int y;   /* Geçersiz */
}
```

```
{
	int x;
	{ }
	int y;   /* Geçersiz */
}
```

```
{
	{ int x; }
	int y;   /* Geçersiz */
}
```

Tamsayı Veri Tipi (int)

Tamsayı türündeki veriler için kullanılan **int** tanımlamasının başına

short,
long ve/veya
signed,
Unsigned

ekleri gelebilir. Bu şekilde tanımlanan bir veri tipi sırası ile *kısa tamsayı, uzun tamsayı, işaretli (signed) tamsayı ve işaretsiz (unsigned) tamsayı* türünde veriler içerebilir. Eğer küçük bir değer aralığı söz konusu ise, örneğin bir sınıftaki yaş ortalaması hesaplanmak isteniyorsa, **short int** kullanmak yeterli olacaktır. Ancak amaç TL cinsinden bir muhasebe hesabıysa, rakamlar çok büyük çıkacağı için, **long int** kullanmamız gerekir. Yaş ortalaması hesabında, kimse negatif bir yaş değerine sahip olamayacağı için, tanımlayacağımız tamsayının işaretsiz olması daha uygun olacaktır. Bunun bize kazandırdığı şey değer aralığının negatif kısmında kalan bölümü de pozitife taşınarak iki kat büyüklükte değerler alabilmesidir.

```
int   t, m, n;
short int  say, m=99;
long int   sonuc;
unsigned int    orta;
unsigned long int    aylik;
```

Tamsayı değişkeni yalnızca **int** olarak tanımlandığı durumda **signed** kabul edilir yani negatif değerler alabilir, **short** veya **long** olması derleyiciden derleyiciye değişir ve genellikle derleyicilerin OPTION menülerinden bunu ayarlamak mümkündür. Örneğin Turbo C++'da **int** tek başına kullanıldığında **short int** anlamına gelirken Visual C++'da **long int** anlamına gelir.

Tek/Çift Duyarlıklı Reel Sayı Veri Tipi (float, double)

Çift duyarlı reel sayılar ile yaklaşık 12 basamak duyarlı reel sayıların tanımlanması sağlanır. Genelde, bu veri tipi bellekte toplam 8 byte'lık bir alan kullanır. Bu veri tipine ait değişkenlerin tanımlanabilmesi için **double** özel amaçlı sözcüğü kullanılır. Reel sayı (**float**) veri tipi ise, çift duyarlıklı reel sayılara (double) oranla daha küçük sayılarla işlem yapılması gerektiği durumlarda kullanılmalıdır.

```
float    x, y;
float    f=12.34, d;
Double   ortalama;
Double   deger=12.3456789;
```

Karakter Veri Tipi (char)

Bir karakter veri tipinin tanımlanabilmesi için C dilinde **char** özel amaçlı sözcüğü kullanılır. Bu veri tipi ile 1 bayt uzunluğunda tek bir karakter değerinin tanımlanması mümkün olur. Karakter veri tipi ile karakter değerlerin bellekte saklanması sağlanır. ASCII tablosunda 128 karakter kodlanmıştır. Her karakter bir tamsayı değeri ile temsil edilir. Karakterler arasında yapılan karşılaştırma işlemleri sırasında bu tamsayı değerleri kullanılır. Örneğin 'A' karakterinin tamsayı karşılığı 65, 'B' karakterinin 66'dır. Yani 'A' karakteri, 'B' karakterinden daha önce sıralanmıştır.

```
char     kod='K', y=66;
char     d, k;
```

Değişkenler

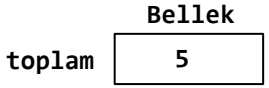
Bir program içinde belirli verileri tutmak için kullanılan parametrelerdir. Kullanım amacına göre her hangi bir veri tipi olabilir. Ancak C dilinde değişkenlerin isimlendirilmesi, tanımlanması ve kullanımı ile ilgili dikkat edilmesi gereken belirli kurallar vardır. C'de komut olarak tanımlanan bazı kelimeler (if, while, do, main, case vb.) değişken ismi olarak kullanılamaz. Eğer programda bir hata olmadığından eminseniz ve hala derleyici hata veriyorsa bu durumda değişkenlere verdiğiniz isimleri kontrol etmeniz gerekebilir.

Örneğin donanım satan bir bilgisayar şirketine yazılım ürettiğinizi varsayalım ve anakart, disket sürücü, işlemci, kasa gibi ürünlerin bilgilerini birer değişkende tutmak ve İngilizce isimleriyle tanımlamak istediğinizi kabul edelim ('mainboard', 'floppy', 'cpu', 'case' vb.). İlk üçünde bir sorun olmasa da, kasa için kullanmış olduğunuz '**case**' aynı zamanda, 'durum' anlamına gelen bir C komutudur. Bu yüzden program hata verecektir. Programcı bu tür çakışmalara karşı her zaman tedbirli olmalıdır.

```
int      toplam;
```
Veri tipi değişken ismi;

Şeklindeki C program cümlesi derleyiciye bellekte **toplam** isimli bir değişkenin (dolayısıyla bu isimdeki bir bellek hücresinin) tamsayı değerler alabilecek şekilde oluşturulması gerektiğini ifade eder. Böylece derleyici, **toplam** değişkeni için gerekli büyüklükteki bellek alanını ayırır. Bu değişkenin aldığı değeri 5 olarak varsayalım. Bilgisayar bellekte **toplam** değişkeni için bir yer ayıracak ve **toplam** değişkeni için tanımlanan 5 değerini bellekteki bu alana yerleştirecektir. Aslında, bu değişkenin bellekte tutulduğu alanın ilk adresi bilgisayar tarafından işaretlenir ve daha sonra bu değişkene ait değere ulaşılmak istendiğinde bu adres kullanılır. Ancak programcılar için bu adres çok da anlamlı değildir ve hatırlanması zordur. Bu nedenle değişkenlerin bellekte saklanan değerlerine ulaşmak için, değişkene verilen isim kullanılır.

Sağ tarafta görüldüğü gibi bellek hücresinde tutulan 5 değerine, bu hücre için tanımladığımız değişken ismi ile ulaşmamız mümkün olur.



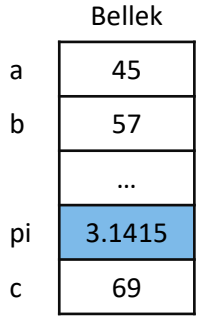
Sabitler ve Sabit Değerler

Değişkenler bellekte verinin sakladığı hücrelere verilen sembolik adlar olup, programın yürütülmesi süresince içerikleri değiştirilebilir. **Sabitler** de bellekte verinin sakladığı hücrelere verilen sembolik adlardır; ancak bildirilirken verilen değerler programın yürütülmesi süresinde değiştirilemez. Kısaca, sabite verilen başlangıç değeri değiştirilemez, kendisine atama yapılamaz.

Sabit Değerler

Bir bağıntı içinde veya iki değişkenin karşılaştırılması gibi işlemlerde kullanılan ve genellikle sabit olarak belirlenen değerlere "sabit değerler" denir. Sabit değerler, değişkenlere atanan ve işlem sırasında değişmeyen sayısal, karakter veya diğer veri türlerini ifade eder. Bu tür sabitler, verilerin bellek üzerindeki gösterim biçimlerine göre farklı şekilde saklanır ve işlenir.

Örneğin, işaretsiz bir tamsayı, bellekte doğrudan ikili sayı (binary) formatındaki karşılığıyla tutulur. Bunun yanı sıra, karakter verileri de bellekte sayısal değerlerle ifade edilir. Her bir karakter, belirli bir karakter kümesindeki (örneğin ASCII ya da Unicode) sayısal değeriyle ilişkilendirilir ve bu sayısal değer de ikili tabanlı bir gösterimle bellekte saklanır. Yandaki küçük tabloda gösterilen **a**, **b** ve **c** değişken, **pi** ise sabittir.



Sabit Bildirimi

Sabit bildirimi, başlangıç değeri verilen değişken bildirimi gibi yapılır; ancak veri tipinin önüne **const** anahtar sözcüğü koyulmalıdır. Örneğin,

```
const float pi = 3.141592;
const double e = 2.71828182845905;
const int EOF = -1;
const chart [] = "Bir Tuşa Dokun";
```

gibi sabit bildirimleri geçerli olup bunların içerikleri program boyunca değiştirilemez, yalnızca kullanılabilir.

Değişken Bildirimleri

Bir C programında kullanılacak değişkenler önceden bildirilmelidir. Değişkenin nerede bildirildiği oldukça önemlidir. Bilindiği üzere bir C programı birçok fonksiyonların birleştirilmesinden oluşmaktadır. Eğer bir değişken, programdaki fonksiyonların bir çoğunda kullanılacaksa **genel (global)** olarak; yalnızca tek bir fonksiyon içinde kullanılıyorsa, o fonksiyonun içinde **yerel (local)** olarak bildirilmesi söz konusudur.

Yerel Bildirim

Yerel değişkenler kullanıldığı fonksiyonun içerisinde bildirilir; yalnızca bildirildiği fonksiyon içerisinde tanınır ve kullanılabilir. Bildirildiği fonksiyon dışında tanınmaz. Yerel değişkenler bildirildiği fonksiyon yürütüldüğü anda bellekteki veri alanında yer kaplarlar. Örneğin aşağıdaki **yerel.c** programında bulunan fonksiyonlarda,

```
main() {
      int m,n; /* m ve n yerel değişkenlerdir */
      m=7;
      n=ikikat(m);
      printf("%d nin iki kati=%d dir.",m,n);
}
int ikikat(k)
int k;
{
      int gecici;  /* gecici de yerel değişkendir */
      gecici=k*2;
      return gecici;
}
```

yalnızca yerel değişken kullanılmıştır. **main()** fonksiyonunun yerel değişkenleri **m** ve **n**'dir. Bu iki değişkenin tuttuğu değerler **ikikat()** adlı fonksiyon içinde tanınmayacağından kullanılamazlar. Çünkü bir yerel değişken yalnızca bildirildiği fonksiyonda geçerlidir. **main()**'de **m** adlı değişkene 7 değeri atanmıştır; bunun iki katını hesaplayan **ikikat()** adlı fonksiyona, bu değer gönderilmelidir. Görüldüğü gibi **m**, **ikikat()**'a gönderilmektedir. **ikikat()** adlı fonksiyonda, gönderilen **m**'nin değeri **k** adlı değişkende tutulmaktadır. **ikikat()** adlı fonksiyonun yerel değişkeni **gecici**'dir. **k**'nın iki katı hesaplanıp **gecici** adlı değişkene atanır ve **return** ile çağıran fonksiyona gönderilir.

Genel Bildirim

Genel değişken bütün fonksiyonların dışında bildirilir. Bir değişken bütün program boyunca sürekli olarak kullanılıyorsa, genel olarak bildirilmelidir. Böyle bir değişken yerel olarak bildirilirse, her fonksiyona gönderilmesi gerekir. Bu da gereksiz parametre aktarımına neden olur. Programcı, değişkenleri bildirmeden önce hangilerinin yerel ya da genel olacağını belirlemelidir. Örneğin, yerel değişken bildiriminde verilen **yerel.c** program genel değişken kullanılarak tekrar yazılırsa, aşağıda **genel.c** kodunda görüleceği gibi parametre aktarımına gerek kalmamaktadır.

```
int m,n;   /* değişkenler geneldir */

void ikikat ();
main() {
      m=7 ;
      ikikat() ;
      printf("%d nin iki kati=%d dir",m,n);
}

void ikikat()
{
      n=m*2;
}
```

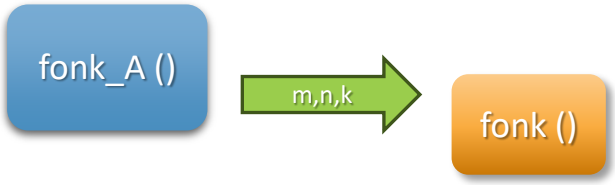
Yukarıdaki program bir öncekiyle aynı işi yapmaktadır. Ancak kullanılan değişkenlerin bildirimi bütün fonksiyonların dışında yazılarak, genel yapılmıştır. **m** ve **n**, programın genel değişkenleridir. Dikkat edilirse, değişkenlerin bildirim yerleri programı da değiştirmektedir. Önceki programda **ikikat()** adlı fonksiyon, sonucu **return** ile göndermekteydi; burada, **n** ve **m** adlı değişkenler genel olduğu için, **ikikat()**'a ne parametre gelmekte ne de sonuç gönderilmektedir. Çünkü genel değişkenler bütün fonksiyonlar içinde kullanılırlar, **ikikat()** adlı fonksiyonun tipi, **return** ile birşey göndermediği için hiçbir şey anlamında **void**'dir.

Formal Bildirim

Fonksiyonların bazı argümanları olabilir. Bunlara **formal parametre** adı verilir ve fonksiyonlara parametre aktarımı için kullanılır. Formal parametreler de uygun şekilde bildirilmelidir; bunlar da yerel değişkenler gibi yalnızca tanımlandığı fonksiyon içerisinde geçerlidirler. Formal parametre bildirimi hemen fonksiyon adının altında yapılabilir. Aşağıdaki program yapısının formal parametreleri **m**, **n** ve **k**'dir ve **fonk()** yürütüldüğü sürece bellekte yer kaplarlar.

```
fonk(m, n, k)
int m, n;
char k;
{
   int a;
   .
   . fonksiyon işlemleri;
   .
}
```

m, n ve k formal parametrelerdir, m ve n tamsayı, k karakter tiptedir, a ise fonksiyonun yerel değişkenidir. Formal parametreler fonksiyona değer aktarımı yapılacağı zaman kullanılırlar.



Formal bildirim aşağıdaki gibi de (yeni standart için geçerli) olabilir. Eğer aşağıdaki gibi yazılırsa, her değişkenin önüne tipi, ayrı ayrı yazılmalıdır.

```
fonk(int m, int n, char k)
{
   int a;
   .
   . fonksiyon işlemleri;
   .
}
```

Harici (Extern) Bildirim

Büyük programlar uygun olarak parçalara bölünüp, ayrı ayrı yazılıp derlendikten sonra birbirleriyle bağlantıları kurulabilir, birleştirilebilir. Bu, derleme süresini kısaltır. Çünkü tamamen doğru olarak yazılmış fonksiyon bir kez derlendikten sonra, programın tamamında kullanılırken tekrar derlenmez; yalnızca bağlantı kurulur. Program uygun parçalara bölünürken, kullanılacak olan genel değişkenler bir program dosyasının birinde normal olarak bildirilirler, diğerlerinde harici bildirim yapılır. Harici bildirim **extern** anahtar sözcüğü kullanılarak yapılır. Aşağıdaki örnekte, programın iki ayrı parçaya bölündüğü varsayılmıştır: A parçasında **x** ve **k** genel olarak bildirilmiştir, ayrı değişkenler B parçasında da genel değişken olarak kullanılacaktır; eğer B parçasında bildirimin önüne **extern** yazılmazsa derleme hatası oluşur. Diğer bir deyişle B parçasında harici bildirim yapılmalıdır.

A PARÇASI (main.c)

```
int x;
char k;

main()
{
   int ...
   .
   . işlemler;
   .
}

fonk()
{
   x=x*6;
}
```

B PARÇASI (module.c)

```
extern int x;
extern char k;

fonk2()
{
   .
   . x=x*x;
   .
}

fonk3()
{
   k='B';
}
```

Bu fonksiyonları tek programda birleştirmek için sırasıyla, gcc main.c -o main.o –c, gcc module.c -o module.o –c ve gcc -o myprog main.o module.o komutları uygulanmalıdır. Böylece tüm programlar **myprog** olarak birleştirilmiş olacaktır.

Statik Değişken Bildirimi

Bilindiği gibi yerel değişkenler yalnızca bildirildiği fonksiyon içinde tanınırlar ve ait olduğu fonksiyon yürütüldüğü anda bellekte yer kaplarlar. Fonksiyon sonlandığında yerel değişkenlere atanan bellek hücreleri, başka kullanımlar için tekrar boş bellek alanına eklenir. Bu nedenledir ki, fonksiyon tekrar yürütülmeye başladığında yerel değişkenlere bellekten yeni hücreler verileceği için eski değerleri olmayacaktır. Fakat bazı uygulamalar var ki, yerel değişkenin tuttuğu değer kaybolmasın, eski değerini korusun istenir. Bu istek daha önce görülen yerel değişken bildirimiyle yapılamaz. Bunu yapabilmek için, yerel değişken bildirimi önüne **static** anahtar sözcüğü yazılmalıdır. Diğer bir deyişle statik değişken bildirimi yapılmalıdır. Statik değişkenler genel değişkenler gibi programın yürütülmesi boyunca bellekte yer işgal ederler, ancak genel değişkenler gibi diğer fonksiyonlarca tanınmazlar. Örneğin,

```
int ver()
{
   static int k;

   k=k+5;
   return k;
}
```

programında **k**, **ver()** adlı fonksiyonun statik yerel bir değişkenidir. Bu fonksiyon her çağrılmasında bir önce gönderdiği değerin 5 fazlasını gönderir.

Statik değişkene iyi bir uygulama olarak, her çağrılışında **fibonacci** serisinin bir sonraki elemanını veren fonksiyonun yazımı verilebilir. Fibonacci serisinin ilk iki elemanının değeri **1**'dir, diğer elemanları kendisinden hemen önce gelen iki elemanın toplamına eşittir. Burada **f1** ve **f2** statik olarak verilmiştir ve fonksiyon ilk çağrıldığında **başlangıç değerlerini alırlar**, ancak daha sonraki çağrımda **en son aldığı değerler** kullanılır.

```
int fibonacci ()
{
   static int f1=1, f2=1;
   int g;

   g=f1+f2;
   f1=f2;
   f2=g;
   return g;
}
```

Kayıt Tipli (Register) Değişken Bildirimi

Diğer bir değişken bildirimi, **register** anahtar sözcüğünü tamsayı ve karakter değişken bildirimlerinin önüne koyarak yapılır. Bu bildirime kayıt tipli değişken bildirimi denir. Bu tip değişkenler için bellek hücresi kullanılmaz; doğrudan işlemcinin genel amaçlı saklayıcıları kullanılır. Bir değişken bellekte değil de işlemcinin saklayıcısında tutuluyorsa, o değişkene erişim daha hızlı olur. Çünkü bellekte tutulan değişkenlere erişmek, en azından bellekten bir okuma ve belleğe bir yazma işlemi yapmayı gerektirir. Eğer değişken kayıt tipli bildirilmişse bu işlemler gerekmez. Genelde döngü sayaçları gibi çok kullanılan değişkenler kayıt tipli olarak bildirilirler.

```
fonk(a, k)
register int a;
register char k;
{
   register int m, n;
   .
   .
}
```

Yukarıdaki programda **a** ve **k** formal parametrelerdir ve saklayıcı tipli bildirilmiştir. Kayıt tipli diğer değişkenler fonksiyonun yerel değişkenleri olan **m** ve **n**'dir. Görüldüğü gibi dört tane kayıt tipli değişken bildirilmiştir. En fazla kaç tane bildirileceği işlemciye ve kullanılan derleyiciye bağlıdır. 16 bitlik işlemcilerde genelde en fazla iki tane kullanılabilir; iki taneden fazlası önemsenmez, kayıt tipli olmadığı varsayılır.

Kayıt tipli değişkenlerin önüne **&** işaretçi operatörü koyularak kullanılmamalıdır, işlemci saklayıcısının adresi olmaz, bellek hücrelerinin adresi vardır. Dolayısıyla, yalnızca bellekte saklanan değişkenlerin önüne **&** işaretçi operatörü koyularak kullanılabilir.

Operatörler

Hazırlanan bilgisayar programı içerisinde gerekli olan aritmetik işlemlerin yapılmasını, karşılaştırma ve mantık yürütme durumlarının gerçekleşmesini sağlayan işaretlere **"operatör"** denilmektedir. C dilinde "aritmetik", "karşılaştırma", "mantıksal" ve "bit bit" operatörleri olmak üzere dört sınıf operatör bulunmaktadır.

Aritmetik Operatörler

Değişkenler veya sabitler üzerinde temel aritmetik işlemleri gerçekleyen operatörlerdir. Bunlar;

Operatör	Açıklama	İşlem	Sonuç
+	Toplama	2 + 2	4
-	Çıkarma	3 – 2	1
/	Bölme	4 / 2	2.0
*	Çarpma	3 * 2	6
%	Mod alma (Kalan)	12 % 5	2
++	1 arttırma	i++, ++i	
--	1 eksiltme	i--, --i	

Yukarıdaki operatörlerin ilk dört tanesi bütün programlama dillerinde olan temel aritmetik operatörlerdir; ancak son üç tanesine diğer dillerde pek rastlanmaz. Artık bölme operatörü "%" tamsayı ve karakter değişkenlerde kullanılabilir. Yaptığı işlem, bölme işlemi sonunda kalanı hesaplamaktır. "**--**" ve "**++**" operatörleri tek bir değişken üzerinde işlem yaparlar ve tamsayı değişkenlerde kullanıldığında bir azaltma ve bir arttırma işlemini gerçekler; ancak genelde, değişkenin bildirildiği tipin kümesindeki bir sonraki elemanı elde etmek için kullanılır: Bir m değişkenin tipi tamsayı ve değeri 7 ise m++ işlemi sonunda m'nin değeri 8 olur, çünkü tamsayılar kümesinde 7'den sonra 8 gelir; eğer değişkeninin tipi karakter ve içeriği 'A' ise işlemi sonucunda 'B' olur, çünkü ASCII tabloda 'A' dan sonra 'B' gelmektedir.

"--" ve "++" operatörleri

Arttırma (++) ve azaltma (--) operatörleri tek bir değişken üzerinde işlem yaparlar ve bir atama işlemiyle birlikte kullanılmaz ise operatör değişkenin önünde ya da arkasında bulunabilir; işlem açısından bir farklılık yoktur. Örneğin;

```
#include <stdio.h>
void main(){
    int m=7, j=7;        /* başlangıç değerleri 7 */

    m++;                 /* m = m + 1   anlamında */
    ++j;                 /* j = j + 1   anlamında */

    printf("%d, %d", m, j);
}
```

Burada m ve j tamsayı değişkenlerdir. Arttırma operatörü m'nin önünde, j'nin arkasında kullanılmıştır. Her ikisi de değişkenin içeriğini bir arttırmaktadır. Yazma fonksiyonu ekrana 8, 8 yazacaktır.

Bu operatörler bir atama işlemiyle (=) birlikte kullanılırsa, değişkenin önünde ya da arkasında olması farklı sonuçları üretir. Çünkü işlem önceliği değişmektedir, bu açıdan dikkat edilmelidir. Örneğin;

```
#include <stdio.h>
void main(){
    int m=7, j=7, k, z;

    k = ++m;             /* m = m + 1 ve k = m  anlamındadır */
    z = j++;             /* z = j  ve j = j + 1   anlamında */

    printf("%d, %d", k, z);   /* çıktı: 8, 7 */
}
```

Karşılaştırma Operatörleri

Program içerisinde bir değişkenin değerini başka bir değişkenin değeri veya bir sabitle karşılaştırılmasını sağlayan operatörlere ''İlişkisel veya karşılaştırma operatör" adı verilir. Karşılaştırma durumuna göre sonuç "1" veya "0" ya da "True/False" yani "Doğru/Yanlış" olarak algılanır. Karşılaştırma işleminde istenen şart sağlanmıyorsa sonuç her zaman "yanlış" yani "0", şart sağlanıyorsa sonuç "doğru" yani "1" olur. Karşılaştırma operatörleri;

Operatör	Örnek	Anlamı
>	a > b	a, b'den büyük
<	a < b	a, b'den küçük
>=	a >= 7	a, b'den büyük veya b'ye eşit
<=	a <= b	a, b'den küçük veya b'ye eşit
==	a == b	a, b'ye eşit
!=	a != b	a, b'ye eşit değil veya b'den farklı

Karşılaştırma operatörleri özellikle **"if"** gibi kontrol akışı işlemlerinde ve **"for/while"** gibi döngü işlemlerinde her hangi bir koşulun sınanması amacıyla kullanılır. Karşılaştırma sonucu doğru (true) ya da yanlış (false) çıkar. Koşul doğruysa olumlu varsayılır ve buna göre yapılması istenenler işlemler yerine getirilir. Yanlış ise olumsuz varsayılır ve buna göre ya başka işlemler yapılır ya da hiçbir işlem yapılmaz atlanır.

```
if(m>15){
    printf("Yanlis sayi");
}
```

```
if(k){
    printf("Sayı sıfır değil");
}
```

Mantıksal Operatörler

Karşılaştırma ve döngü deyimlerinde koşulun sınanması için karşılaştırma operatörleri kullanılır. Ancak birden fazla koşul olduğunda bunların birleştirilip tek bir koşul durumuna getirilmesi gerekir. Böyle durumlarda **mantıksal operatörler** kullanılır. Diğer bir deyişle, mantıksal operatörler birden çok koşulun birleştirilmesi amacıyla kullanılır. Aritmetik operatörlerde olduğu gibi bu operatörlerde de işlem soldan sağa gerçekleştirilir. Ancak sonuçlar YANLIŞ ve DOĞRU'ları ifade ettiğinden karışıklıkları önlemek için parantezlerle kullanılmaları daha uygundur. Mantıksal operatörler 3 tane olup bunlar, VE (AND) – "&&", VEYA (OR) – "||" ve DEĞİL (NOT) – "!" işaretleridir.

Mantıksal operatörler "VE / VEYA" her zaman iki koşulu karşılaştırmak için kullanılır. Yani "A && B" ya da "A || B" şeklinde kullanılırlar. Ayrıca gruplama suretiyle birden fazla karşılaştırma yapmak da mümkündür. A ve B birer koşul olmak üzere bu üç operatörün doğruluk tabloları şu şekildedir.

VE (&&)

A	B	Sonuç
Doğru 1	Doğru 1	Doğru 1
Doğru 1	Yanlış 0	Yanlış 0
Yanlış 0	Doğru 1	Yanlış 0
Yanlış 0	Yanlış 0	Yanlış 0

VEYA (||)

A	B	Sonuç
Doğru 1	Doğru 1	Doğru 1
Doğru 1	Yanlış 0	Doğru 1
Yanlış 0	Doğru 1	Doğru 1
Yanlış 0	Yanlış 0	Yanlış 0

DEĞİL (!)

A	Sonuç
Doğru 1	Yanlış 0
Yanlış 0	Doğru 1

```
Void main(){
    int sayi;
    printf(”Bir sayı girin :");
    scanf("%d”, &sayi);

    if(sayi>19 && sayi<51){
       printf("Girilen sayı 20 ile 50 arasındadır.");
    } else {
       printf("Yanlış sayı girildi.");
    }
}
```

```
Void main(){
    float pay, payda, sonuc;

    printf(”Pay ve payda değerini girin :");
    scanf("%f, %f”, &pay, &payda);

    if((payda != 0 && pay != 0) && pay > payda){
       sonuc = pay / payda;
       printf("Sonuç = %f", sonuc);
    }
}
```

Bitler Üzerinde İşlem Yapan (bitwise) Operatörler

Bitler üzerinde yani "0" ve "1"ler ile işlem yapmak, bir tamsayı veya karakter değişkenin (short, int, long ve chaf) bir bütün olarak sayısal değeri üzerinde değil de doğrudan bitlerini sınamak, değiştirmek ve öteleme yapmak anlamına gelmektedir. Bu operatörler özellikle modem, yazıcı vb. gibi aletlerin durum kontrollerini yapmak amacıyla kullanılır. Örneğin belirli cihazların iletişim kanalından alınan bir kısa tamsayının ikinci bitinin ne olduğu öğrenilmek isteniyorsa bu operatörleri kullanmaya gerek vardır. Bit üzerinde işlem yapan 6 operatör vardır. Bu operatörler ayrıca atama operatörüyle birlikte de kullanabilmektedir.

Operatör	Açıklama
&	bit düzeyinde VE
|	bit düzeyinde VEYA
^	bit düzeyinde XOR (özel VEYA)
<<	bit sola kaydırma
>>	bit sağa kaydırma
~	bit düzeyinde DEĞİL (birlerin tamamlayıcısı)

Operatör	Açıklama
&=	bit VE işleminden sonra atama
|=	bit VEYA işleminden sonra atama
^=	bit özel VEYA işleminden sonra atama
<<=	bit sola kaydırma işleminden sonra atama
>>=	bit sağa kaydırma işleminden sonra atama

Örneğin 69 sayısı ikili sistemde "01000101" olarak ifade edilmektedir. Eğer bit sağa kaydırma işlemi yapılırsa, bütün bitler 1 sola kayar ve en sağdaki bit düşer, sağ tarafa ise "0" biti eklenir. Sola kaydırma da ise bütün bitler sola kaydırılır ve en soldaki düşer, sağa "0" eklenir.

```
8
2426
16318421
01000101 = 69    // sağa kaydırma
00100010 = 34    // 2'ye bölünmüş

01000101 = 69    // sola kaydırma
10001010 = 138   // 2 ile çarpılmış
```

```
void main(){
    int x = 69 , b, c;
    b = x >> 1;
    c = x << 1;

    printf("%d, %d", b, c);      // çıktı 34, 138
}
```

Bit işlemlerine örnekler

Genellikle VE operatörü "&" bir değişkenin bazı bitlerini sıfırlama için kullanılır. Örneğin "k = k & 4" işlemine bakılırsa;

```
0000 0100 = 4     // 8 bit gösterim
xxxx xxxx = k

0000 0x00 = k & 4
```

k değişkeninin 4 ile VE işleminde yalnızca 4 sayısının 1 olan bitine denk kısmı değişmez (x aynı kalıyor anlamında), diğerleri sıfır olur. Bu örnek, bilgisayarın iletişim kanalından alınan sayının 3. bitinin sınanması için kullanılabilir.

VEYA operatörü "|" genelde, bir sayısal değişkenin bazı bitlerini birlemek için kullanılır. Örneğin 16 bitlik bir m değişkeni için "m = m | 255" işlemini dikkate alalım;

```
00000000 11111111 = 255     // 16 bit gösterim
xxxxxxxx xxxxxxxx = m

xxxxxxxx 11111111 = m | 255
```

16 bit'te 255 sayısının ilk 8 biti "1" dir. Bu nedenle m değişkeni ile "VEYA" işlemine girdiğinde m değişkenin de ilk 8 biti "1" değerlerini alacaktır.

Bit işlemlerine örnekler

Özel VEYA (XOR) operatörü, karşılaştırılan bitlerin değeri birbirinden farklı ise 1, aynı ise sıfır 0 üretir. Her hangi bir değişkenin kendisiyle XOR işlemine tabi tutulması onun değerini sıfırlayacaktır. Çünkü karşılıklı bitleri aynı olduğundan bütün bitler sıfır değerini olacaktır. XOR operatörünün doğruluk tablosu altta gösterilmiştir.

Bir XOR örneği olarak n = 15 ve m = 60 değişkenlerini dikkate alalım. Bu iki değişkenin XOR işleminde 15 değeri elde edilir.

```
8
2426
1631 8421
0000 1111 = 15  // n
0011 1100 = 60  // m

0011 0011 = n ^ m  // 51
```

özel VEYA / XOR (^)		
A	B	Sonuç
Doğru 1	Doğru 1	Yanlış 0
Doğru 1	Yanlış 0	Doğru 1
Yanlış 0	Doğru 1	Doğru 1
Yanlış 0	Yanlış 0	Yanlış 0

DEĞİL operatörü, bit seviyesinde "0"ları "1"e, "1"leri "0"a çevirir. Örneğin m = 60 değişkenine DEĞİL operatörü uygulanırsa 195 değeri elde edilir.

```
8
2426
1631 8421
0011 1100 = 60  // m

1100 0011 = ~m  // 195
```

Operatörlerin öncelik durumu

Aynı satırda 1'den fazla operatör kullanılması durumlarında operatörlerin birbirlerine göre önceliklerini dikkate alınması gerekir. Bu önceliklere göre uygun bir şekilde gruplamalar yapılması gerekebilir.

Yüksek Öncelik

Düşük Öncelik

Operatörlerin Öncelik Sırası	
Operatörler	Öncelik Yönü
`(), [], ->`	Soldan Sağa
`!, ~, ++, --, +, -, *, &, (type), sizeof`	Soldan Sağa
`*, /, %`	Soldan Sağa
`+, -`	Soldan Sağa
`<<, >>`	Soldan Sağa
`<, <=, >, >=`	Soldan Sağa
`==, !=`	Soldan Sağa
`&`	Soldan Sağa
`^`	Soldan Sağa
`|`	Soldan Sağa
`&&`	Soldan Sağa
`||`	Soldan Sağa
`?:`	Sağdan Sola
`=, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=`	Sağdan Sola
`,`	Soldan Sağa