C PROGRAMLAMA DİLİNE GİRİŞ

En Basit C Programı

C programlama dili, her programcının ihtiyacını görecek genel amaçlı, emreden paradigmalı (kısa ve öz talimatlar birbiri ardına verilerek yapılan programlama), yapısal programlama dilidir. Yapısal programlamanın çerçevesi *TEMEL YAZILIM KAVRAMLARI* başlığı altında anlatılmıştır.

FORTRAN II dilinde talimatların yedinci sütunda başlaması gerekir. Pascal dili gibi C dilinde de talimatlar gelişigüzel (free format) yazılabilir.

Yapısal programlamada programın başladığı yeri belirten bir ana fonksiyon (main function) tanımlanır. C dilinde bu fonksiyon `main()` fonksiyonudur. Yani içinde main fonksiyonu yazılmamış bir c programı çalışmaz. Aşağıda en basit C programı verilmiştir.

```
int main()
{

   return 0;
}
```

C dilinde her fonksiyon tırnaklı parantez karakteri ile başlayan ve biten bir kod bloğuna sahiptir ve fonksiyona ait bu blok, fonksiyon bloğu (function block) olarak adlandırılır. Ana fonksiyonunun başındaki `int`, ana fonksiyonun bir tamsayı (integer) geri döndüreceğini belirtir. Ana fonksiyon içinde kullanılan `return 0;` talimatı (statement) main fonksiyonundan sıfır geri döndürerek çıkıldığını söyleyen talimattır. Ana fonksiyondan çıkıldığında programın sonlanıp işletim sistemine dönüleceğinden işletim sistemine `0` sayısını gönderir. Sıfır dışında bir değerin işletim sistemine gönderilmesi, programın hata ile sonlandığı olarak kabul edilir. Yukarıdaki program derlendiğinde aşağıdaki çıktı elde edilir.

```
...Program finished with exit code 0
```

Bu durumda işletim sistemine 11 sayısını gönderen program aşağıdaki şekilde yazılır;

```
int main()
{

   return 11;
}
```

Program çalıştırıldığında aşağıdaki çıktı elde edilir;

```
...Program finished with exit code 11
```

C Programlama Dili Söz Dizim Kuralları

C dili için oluşturulacak kaynak kod programcı tarafından metin dosyasına yazılırken belli söz dizim kullarına (syntax) uyar. Bunlar aşağıda başlıklar halinde verilmiştir.

Kaynak Kodumuzu Oluşturacak Karakterler

1. İngiliz alfabesindeki büyük harfler:
   `A B C D E F G H I J K L M N O P Q R S T U V W X Y Z`
2. İngiliz Alfabesindeki küçük harfler:
   `a b c d e f g h i j k l m n o p q r s t u v w x y z`
3. Rakamlar:
   `0 1 2 3 4 5 6 7 8 9`
4. Özel Karakterler:
   `< > . , _ ( ) ; $ : % [ ] # ? ' & { } " ^ ! * / | - \ ~ +`
5. Metinde görünmeyen ancak metni biçimlendiren beyaz boşluk (white space) karakterleri:
   a. Boşluk (space)

b. ↲ Yeni satır (new line)
c. ⇤ Satır başı (carriage return)
d. ⇦ Geri (backspace)
e. ⇥ Sekme (tab)

Talimatlar ve Anahtar Kelimeler

Yüksek seviyeli bir dilde yazılmış bir talimat (statement), işlemciye belirtilen bir eylemi gerçekleştirmesini söyleyen cümledir. Yüksek seviyeli bir dildeki tek bir talimat, birkaç makine dili emir kodunu (instruction code) temsil edebilir. Talimatlar, yapılaması gerekeni kısa ve net olarak belirten mantıksal satırlardır (logical sequence).

Anahtar kelimeler (keywords), programlamada kullanılan ve C derleyicisi tarafından özel anlama sahip, önceden tanımlanmış, ayrılmış (reserved) kelimeler olup programı oluşturan talimatlar (statements) bu kelimeler kullanılarak yazılırlar.

auto	else	long	switch
break	enum	register	typedef
case	extern	return	union
char	float	short	unsigned
const	for	signed	void
continue	goto	sizeof	volatile
default	if	static	while
do	int	struct	double

Tablo 6. Anahtar Kelimeler

C dilinde;
- Anahtar kelimeler, küçük-büyük harf ayrımı yapıldığından her zaman küçük harflerle yazılırlar.
- Her talimat (statement), anahtar kelimeler içerecek şekilde yazılır ve noktalı virgül karakteri ile biter.

Açıklamalar

Talimatlar (statement), yapılaması gerekeni kısa ve net olarak belirten mantıksal satırlardır (logical sequence). Programcı kodu yazarken açıklama (comment) yapma gereği duyabilir. Bunu iki şekilde yapar;

1. Kod metninde bulunulan yerden sonra satırın sonuna kadar olan bir açıklama yapılmak istendiğinde, açıklama öncesinde iki adet bölü karakteri kullanılır.
2. Eğer birden çok satırı içeren bir açıklama yapılacak ise açıklama `/*` ile `*/` karakterleri arasına alınır.

Aşağıda açıklama eklenmiş bir ana fonksiyon örneği bulunmaktadır.

```
/*
  Bu program, açıklama (comment) içeren basit bir ana fonksiyonu olan C
  programıdır. İlhan ÖZKAN, Aralık 2024
*/
int main() // Ana Fonksiyon
{ //fonksiyon bloğu başlangıcı

    /*
     Ana Fonksiyon, programın başladığı yerdir.
     Ana fonksiyonu olmayan bir program çalışmaz.
    */

    return 0; /*fonksiyondan dönüş talimatı noktalı virgül karakteri ile bitmiştir.*/
} //fonksiyon bloğu bitişi
```

Kodun bir başkası tarafından bakımı yapılacağı göz önüne alınarak okunaklı yazılması çok önemlidir. Bu nedenle kodu yazarken her kod bloğu içindeki açıklama ve talimatları bloktan itibaren girintili olarak yazarız.

```
int main()
{
    //Ana fonksiyon bloğuna ait girinti
    {
        //iç bloğa ait girinti...
        return 0;
    }
    return 0;
}
```

Kodu bu şekilde yazmamızın derleyici açısından hiçbir önemi yoktur. Derlemenin ilk aşamasında bütün bu açıklamalar ve beyaz boşluk karakterleri metinden çıkartılır. Aşağıda en kısa C programı verilmiştir.

```
int main(){return 0;}
```

Değişken Tanımlama

Her yapısal programlama dilinde olduğu gibi C dilinde de veri yapıları (data structure) ve kontrol yapıları (control structure) birbirinden ayrıdır. Dolayısıyla veriyi işleyecek kontrol yapısı kodlanmadan önce veriyi taşıyan veri yapıları tanımlanmalıdır.

Değişkenler veri yapılarının temel öğeleridir. Değişkenin (variable) ne olduğu *Değişken* başlığı altında verilmiştir. Değişkenler, belleğin belli bir bölgesini işgal eder ve fiziki olarak bellek bitlerden oluştuğundan ikili sayı (binary) sisteminde veri tutarlar. C dilinde verinin bellekte kapladığı yer ve biçimi için kullanılan ilkel veri tipleri (primitive data type) için anahtar kelimeler aşağıda verilmiştir;

1. Tamsayı olarak kullanılacak veri tipleri
   a. `char` bellekte 1 bayt yani 8 bit yer kaplayan işaretli tamsayıları tutan veri tipidir.
   b. `short` bellekte 2 bayt yani 16 bit yer kaplayan işaretli tamsayıları tutan veri tipidir.
   c. `int` bellekte 4 bayt yani 32 bit yer kaplayan işaretli tamsayıları tutan veri tipidir.
   d. `long` bellekte 8 bayt yani 64 bit yer kaplayan işaretli tamsayıları tutan tipidir.
2. Gerçek sayı veri tipleri
   a. `float` bellekte 4 bayt yani 32 bit yer kaplayan tek hassasiyetli (single precision) kayan noktalı gerçek sayıları tutan veri tipidir.
   b. `double` bellekte 8 bayt yani 64 bit yer kaplayan çift hassasiyetli (double precision) kayan noktalı gerçek sayıları tutan veri tipidir.

C programlama dilinde hiçbir değeri olmayan ve bellekte yer kaplamayan `void` veri tipi vardır. Bu veri tipini hiçbir değer döndürmeyen fonksiyon tanımlamalarında ileride çokça göreceğiz.

Değişkenlerin veri tipine bağlı olarak kimlik verilerek tanımlanması (definition) gerekir. Değişken tanımlaması yapılabilmesi için ilk önce veri tipinin (data type) belirlenmesi ve değişkenin kullanılacağı yerler göz önüne alınarak benzersiz bir kimlik (identifier) verilmesi gerekir.

```
/* Bu program değişken tanımlamak için oluşturulmuştur.*/
int main()
{
    char yas; /* veri tipi "char" ve kimliği "yas" olan değişken tanımlandı*/
    int kat;  /* veri tipi "int" ve kimliği "kat" olan değişken tanımlandı */
    float kilo; /* veri tipi "float" ve kimliği "kilo" olan değişken tanımlandı */
    return 0;
}
```

Yukarıda tanımlaması yapılan `yas`, `kat` ve `kilo` değişkenleri ana fonksiyon olan `main` fonksiyon bloğu içinde tanımlandığından, tanımlanan değişkenler sadece bu blok içinde geçerlidir. Aynı blok içinde bu değişken kimlikleri bir başka değişkene verilemez.

Bildirim (declaration) derleyiciye böyle bir değişken, fonksiyon veya nesne var demenin adıdır. Tanımlama (definition) derlemenin sorunsuz olabilecek şekilde eksiksiz yapılan bir işlemdir.

İşaretli bir tamsayı kullanmak yerine en anlamlı işaret bitini de sayı kabul ederek işaretsiz olarak tamsayıları kullanabiliriz. Yukarıda verilen `char`, `short`, `int` ve `long` tamsayı veri tipleri aslında işaretli

tamsayılardır. Yani önlerinde `signed` sıfatı olduğu varsayılır. Varsayılan (default) olarak bu sıfat varmış diye kabul edilir. Değişkenleri işaretsiz olarak kullanmak istememiz halinde `unsigned` sıfatını tamsayı veri tipinin önünde kullanırız.

```
/* Bu program işaretsiz tamsayı değişken tanımlamak için oluşturulmuştur.*/
int main()
{
    unsigned yas; /* veri tipi "unsigned char" ve kimliği "yas" değişken tanımlandı.
                     yas değişkeni 0 ile 255 arasında bir değer alabilir. */
    unsigned short kat; /* veri tipi "unsigned short" ve kimliği "kat" olan değişken
                           tanımlandı.kat değişkeni 0 ile 65535 arasında değer alabilen
                           bir değişkendir. */
    unsigned int pozitifTamsayi; /* pozitifTamsayi değişkeni 0 ile 4.294.967.295 arasında
                                    değer alabilen bir değişkendir. */
    return 0;
}
```

Değişkenin kapsamı (variable scope); Değişkenin bildiriminin (variable declaration) yapılabileceği, değişkene erişilebileceği (access), değişkenin üzerinde çalışılabileceği program kodu olarak tanımlanır. Değişkenin kapsamı, tanımlandığı kod bloğu (ve varsa bu blok içindeki iç bloklar) ile sınırlıdır.

Değişken Kimliklendirme Kuralları

C programlama dilinde değişkenlerin kimliklendirilmesinde (identifier definition) aşağıda verilen kurallara uyulur.

1. Kimliklendirmede anahtar kelimeler kullanılamaz!
2. Kimliklendirme rakamla başlayamaz!
3. Kimlikler en fazla 32 karakter olmalıdır! Daha fazlası derleyici tarafından dikkate alınmaz!
4. Kimliklendirmede ancak İngiliz Alfabesindeki Büyük ve Küçük harfler, rakamlar ile altçizgi karakteri kullanılabilir.

```
/* Bu program değişken tanımlama örneklerini içerir. */
int main()
{
    /* Tamsayı değişkenler: */
    char yas;
    int tam_sayi;
    long uzun_tam_sayi;

    /* Gerçek Sayı Değişkenler:*/
    float kilo;
    double reel_sayi;

    /*
       Aynı veri tipinde birden fazla değişkene
       aralarına virgül koyarak kimlik verilebilir.
    */
    int kat1,kat2,kat3;
    float olcek1,olcek2;
    return 0;
}
```

Bütün bu kuralların yanında değişkenlere kimlik verilirken yazılan kodun okunaklılığını artırmak için camel case olarak kimlik verilir. Bu kimliklendirme türünde ilk sözcük tamamıyla küçük, izleyen sözcüklerin ise baş harfi büyük yazılır. Bu kural zorunlu değildir ama koda bakım yapacak sonraki programcıların işini kolaylaştırmak için ahlaki olarak tercih edilir.

```
/* Bu program camelCASE değişken kimliklendirme örneklerini içerir.*/
int main()
{
    int sayac;
```

```
    int ogrenciBoyu;
    int asansorunOlduguKat;
    float asansorAgirligi;
    int ogrenciYasi;
    char ilkHarf;
    char ogrenciAdi[50];
    char ogrenciSoyadi[50];
    float ortalamaNot;
    return 0;
}
```

Sabitler

Programcı, kodlama yaparken bazı sabitleri (constants) ister istemez kullanır. Bunların biri değişmez (literal), diğeri ise değeri değiştirilemeyen sabit değişkenlerdir (const variable). Derleme öncesinde de sonrasında da kaynak koddakinin kelimesi kelimesine aynısı olan değişmezler (literal) kullanılır. Değişkenlere verilen ilk değerler (initial value) ile katsayılar (coefficient) en çok kullanılan değişmezlerdir. Değişkenler tanımlanırken sahip olacağı ilk değerleri belirten sabitler de verilebilir. Aşağıda bu değişmezlere ilişkin örnek verilmiştir.

```
/* Bu program değişkenlere ilk değer (initial value) vermede kullanılan
   için değişmez (literal) örneklerini içerir. */
int main()
{
    int i,j=0; /* "int" veri tipindeki "j" kimlikli değişkene
                  0 tamsayı değişmezi ile ilk değer verildi */

    char c=65; /* "char" veri tipindeki "c" kimlikli deki değişkene
                  65 tamsayı değişmezi ile ilk değer verildi */

    float f=2.5; /* "float" veri tipindeki "f" kimlikli değişkene
                    Türkçe 2,5 olan kayan noktalı değişmezi ile
                    ilk değer verildi.
                    Kod İngilizce yazıldığından, kodun içerisindeki
                    gerçek sayı değişmezleri için ondalık ayracı
                    NOKTA olarak yazılır.*/

    char harf='A'; /* "char" veri tipindeki "harf" kimlikli değişkene
                      'A' karakter değişmezi ile ilk değer verildi.
                      Klavyeden basılan her bir harf de bellekte sayı
                      olarak tutulur.
                      'A' karakterinin kodu 65,
                      'B' karakterinin kodu 66'dır... Bu ANSI Standardıdır.
                      BU nedenle bu talimat aşağıdaki şekilde de yazılabilir;
                      char harf=65;
                      Bellekte 1 bayt yer kaplayan "char" 255 farklı
                      karakteri gösterebilir.
                      Kodu 0 olan karakter ise NULL karakter olarak
                      adlandırılır.
                      Günümüzde ANSI yerine UTF-8 ve UTF-16 karakter
                      kodları kullanılmaktadır. */

    return 0;
}
```

Aşağıdaki kod örneğinde programcı, `3`, `3.14`, `3.14` ve `2.0` olmak üzere dört farklı değişmez kullanılmıştır. Bu değişmezlerin her biri derleyici tarafından farklı olarak değerlendirilir.

```
/* Bu program, değişmez örneğidir. */
int main() {
   int yariCap=3;
   float daireninAlani=3.14*yariCap*yariCap;
```

```
   float daireninCevresi=2.0*3.14*yariCap;
   return 0;
}
```

Değişmezlerin (literal) çokça kullanılması derleme sonrası üretilen icra edilebilir makine kodunu da büyütür. Bunun yerine çok kullanılan değişmezler bellekte 1 kez yer kaplasın diye const sıfatıyla (const qualifier) sabit değişkenler (const variable) tanımlanabilir. Sabit değişkenler etik olarak büyük harfle kimliklendirilir.

```
/* Bu program, const sıfatı örneğidir. */
int main() {
   int yariCap=3;
   const float PI=3.14;
   float daireninAlani=PI*yariCap*yariCap;
   float daireninCevresi=2.0*PI*yariCap;
   return 0;
}
```

Böylece aynı sabit PI değişkeni birden çok yerde kullanılır ve her seferinde aynı bellek bölgesine erişildiğinden bellekten tasarruf edilmiş olunur. Tanımlama (definition) sırasında const sıfatı kullanılan değişkene ilk değer (initial value) değişmez (literal) olarak verilmelidir. Daha sonra bu değişkene bir değer ataması olamaz!

Bir değişkeni sabit tanımlamanın üstünlükleri aşağıda sıralanmıştır;

- Gelişmiş Kod Okunabilirliği: Bir değişkene const sıfatı vermek, diğer programcılara değerinin değiştirilmemesi gerektiğini belirtir; bu da kodun anlaşılmasını ve sürdürülmesini kolaylaştırır.
- Gelişmiş Veri Tipi Güvenliği: const kullanılması, değerlerin yanlışlıkla değiştirilmemesini sağlayabilir ve kodumuzda hata ve eksiklik olma olasılığını azaltabilir.
- Gelişmiş Optimizasyon: Derleyiciler, değerlerinin program yürütme sırasında değişmeyeceğini bildikleri için const değişkenlerini daha etkili bir şekilde optimize edebilirler. Bu, daha hızlı ve daha verimli kodla sonuçlanabilir.
- Daha İyi Bellek Kullanımı: Değişkenlere const sıfatı vermek, değerlerinin bir kopyasını oluşturmayı engeller; bu da bellek kullanımını azaltabilir ve performansı artırabilir.
- Gelişmiş Uyumluluk: Değişkenlere const sıfatı vermek, kodunuzu const değişkenleri kullanan diğer başlıklarla daha uyumlu hale getirebilir.
- Gelişmiş Güvenilirlik: const kullanarak, değerlerin beklenmedik şekilde değiştirilmemesini sağlayarak kodunuzu daha güvenilir hale getirebilir ve kodunuzdaki hata ve eksiklik riskini azaltabilirsiniz.

Kod içerisinde değişmezleri (literal) tekrar tekrar yazmamanın bir yolu da `#define` ön işlemci yönergesi (preprocessor directive) ile makro tanımlamaktır. Ön işlemciler, kaynak kodları derlemeden önce derleyiciyi yönlendirerek kaynak kodlar üzerinde işlem yapmayı sağlar. Aşağıdaki örnekte derleyici, derlemeye geçmeden `PI` görülen yerleri `3.14` olarak değiştirmesi söylenir. Derleme bu ön işlemci işleminden sonra gerçekleşir. Değişmez değerleri değiştiren makro kimlikleri büyük harfle kimliklendirilir. Makrolar sonunda talimatlar gibi noktalı virgül karakteri ile bitmez! Çünkü ön işlemci yönergeleri kodların tasnifi ve derleme ön hazırlığını yaparlar.

```
/* Bu program, #define ön işlemci yönergesi örneğidir. */

#define PI 3.14  /* Bu makro, derleme öncesinde PI görülen yerlere
                    3.14 yazılmasını sağlar. */
int main() {
   int yariCap=3;
   float daireninAlani=PI*yariCap*yariCap;
   float daireninCevresi=2.0*PI*yariCap;
   return 0;
}
```

Ön işlemci yönergesini yerine getiren derleyici aynı kodu aşağıdaki şekle çevirerek derleme işlemine geçer. Ayrıca ön işlemci yönergeleri yerine getirilmeden önce bütün açıklamalar ve beyaz boşluklar (white space) kaynak koddan kaldırılır. Bu durumda ön işlemci yönergeleri sonrasında kaynak kodumuz aşağıdaki hale döner;

```
int main() {
   int yariCap=3;
   float daireninAlani=3.14*yariCap*yariCap;
   float daireninCevresi=2.0*3.14*yariCap;
   return 0;
}
```

`#define` ön işlemci yönergesi çok tekrarlanan değişmezler için kullanılabileceği gibi aşağıdaki şekilde de kullanılabilir;

```
/* Bu program, #define ön işlemci yönergesinin bir başka
   kullanımı örneğidir. */
#define BEGIN int main() {
#define END }
#define PI 3.14
BEGIN
   const char ILKHARF='A';
   const float AVAGADROSAYISI=6.022e23;
   float yarimRadyan=PI/2;
   return 0;
END
```

Yazdığımız Kod Nasıl İcra Edilir?

Derleyici tarafından işlemcinin anlayacağı makine diline çevrilen programımız aslında yazdığımız talimatları (statement) sırasıyla çalıştırır. C dilinde her talimat noktalı virgül ile biter.

		boy	kilo	bki	boyKare
	int main() {				
1	float boy=1.80;	1.80	?	?	?
2	float kilo=100.0;	1.80	100.0	?	?
3	float bki;	1.80	100.0	?	?
4	float boyKare;	1.80	100.0	?	?
5	boyKare=boy*boy;	1.80	100.0	?	3.24
6	bki=kilo/boyKare;	1.80	100.0	30.86419753	3.24
7	boy=1.50;	1.50	100.0	30.86419753	3.24
8	return 0;	1.50	100.0	30.86419753	3.24
	}				

Tablo 7. Örnek bir C Programının İcra Sırası

Yapısal programlamada program, ana fonksiyondan başlar. Bu nedenle icra, `main` fonksiyonu içindeki ilk talimatla başlar ve solda verilen sırayla icra edilir. Sağda ise her icra sonrası değişkenlerin değerleri gösterilmiştir. Her talimatın icrasında neler olduğu aşağıda verilmiştir;

1. `boy` değişkenine `1.80` değeri atanır.
2. `kilo` değişkenine `100.0` atanır.
3. `bki` değişkeni tanımlanır.
4. `boyKare` değişkeni tanımlanır. Daha sonraki birkaç satır açıklama olduğundan icra edilecek bir şey yoktur.
5. `boy` ile `boy` değişkeni çarpılır ve sonuç `boyKare` değişkenine atanır. Artık `boykare` değişkeninin değeri bu noktadan sonra `3.24` olmuştur.
6. `kilo` değişkeni `boyKare` değişkenine bölünür ve sonuç `bki` değişkenine atanır. Artık `bki` değişkeninin değeri bu noktadan sonra `30.86419753` olmuştur.
7. `boy` değişkenine `1.50` atanmıştır. Bu atama sonrası `boyKare` ve `bki` değişkenlerinin değeri değişmemiştir. Çünkü bu değişkenleri değiştiren 5. ve 6. talimatlar icra edilmiştir.
8. Programdan çıkılırken işletim sitemine `0` geri döndürülür.

İşleçler

İşleçler (operator) matematikten bildiğimiz işleçlerdir.

$$y = 3 + 2$$

Yukarıda verilen cebirsel ifadede toplama işleci (`+`) iki tarafına bulunan argümanları toplar. Bu argümanlara işlenen (operand) adı verilir. Yüksek düzey dillerde de aritmetik işleçler ve bunun yanında birçok işleç de bulunur. En çok kullanılan işleç, atama (`=`) işlecidir. Atama işleci, sağdaki işleneni soldakine atar. Bu nedenle kodlama yapılırken `2+2=x` veya `a+b=x` yazılamaz!

Aritmetik İşleçler

C programlama dilinde aritmetik işleçlerin çalışma şekli işlenenlerin (operand) veri tipine göre değişir. Bu konu detaylı olarak *Üstü Kapalı Tip Dönüşümleri* başlığında detaylı anlatılmıştır. Aritmetik işlemler ve atama işleci için kurallar;

1. İşlenenlerden birinin bellekte kapladığı yer küçük ise ikisi aynı olacak şekilde bellekte daha fazla yer kaplayanına dönüştürülür ve işlem sonra yapılır ve sonuç dönüştürülen veri tipinde üretilir. Örneğin `char` veri tipindeki değişken ile `long` veri tipindeki bir değişken toplanmaya çalışıldığı zaman `char` veri tipindeki değişkenin değeri `long` veri tipine otomatik dönüştürülür. Sonuç `long` veri tipinde üretilir.
2. İşlenenlerden biri tamsayı diğeri kayan noktalı sayı ise işlem yapılmadan önce işlenenler kayan noktalı sayıya dönüştürülür. Örneğin `int` veri tipindeki değişken ile `float` veri tipindeki bir değişken toplanmaya çalışıldığı zaman `int` veri tipindeki değişkenin değeri `float` veri tipine otomatik dönüştürülür. Sonuç `float` veri tipinde üretilir.

İşleç	İşlenenler (operand) üzerinde yapılan işlem
`+`	Sol ve sağındaki yer alan işlenenleri toplar.
`-`	Soldaki işlenenden sağdakini çıkarır.
`*`	Sol ve sağındaki yer alan işlenenleri çarpar.
`/`	Soldaki işleneni sağdakine böler. Eğer işlenenlerin her iki de tamsayı ise sadece kalan kısmı atılır. Bu durumda son uç yine tamsayı olarak üretilir.
`%`	Kalan (modulus) işleci, sadece tamsayıları işlenen olarak kabul eder. Soldaki işlenenin sağdakine bölümünden kalanı verir. Kalan yine tamsayıdır.

Tablo 8. Aritmetik İşleçler

Aritmetik işleçlere (aritmetic operator) ilişkin yukarıda belirtilen kuralların çalıştığını aşağıdaki örnekte görebilirsiniz.

```
/* Bu program, aritmetik işleç örneğidir. */
int main() {
   int a = 9, b = 4, res;
   float fa=6.6, fb= 2.2, fres;
   res = a + b; // işlem sonucu res=13
   res = a - b; // işlem sonucu res=5
   res = a * b; // işlem sonucu res=36
   res = a / b; // işlem sonucu res=9/4=2 tam 1/4=2
   /*
    toplama işlecinin işlenenleri tamsayı olduğundan,
    bölme sonucundaki tam kısım bölme sonucunu verir.
   */
   res = a / 4.3 ; /* işlem sonucu res tamsayı olduğundan;
                      res=9/4.3=9.0/4.3=2.0930= 2 tam 0.0930=2 */

   res = a % b;   // işlem sonucu res=9%4= 2 tam 1/4=1
   fres= fa / fb; // işlem sonucu res=6.6/2.2=3.00
   return 0;
}
```

C programlama dilinde tamsayı bölme işlemi işlemcinin en basit yetenekleriyle çözülür. Bu nedenle C ve C++ diğer programlama dillerinden ayrılır. Aşağıda buna ilişkin örnek verilmiştir.

```
/* Bu program, aritmetik işleç bölme örneğidir. */
int main () {
   int a = 19 / 2 ; /* a = 2*9+1 = 9 */
   int b = 18 / 2 ; /* b = 9 */
   int c = 255 / 4; /* c = 4*63+3 = 63 */
   int d = 45 / 4 ; /* d = 4*11+1 = 11 */

   double aa = 19 / 2.0 ; /* aa = 9.5 */
   double bb = 18.0 / 2 ; /* bb = 9.0 */
   double cc = 255 / 2.0; /* cc = 127.5 */
   double dd = 45.0 / 4 ; /* dd = 11.25 */

   double ee = 45 / 4 ;   /* ee = 11 çünkü bölme
                             tamsayılar arasında yapılmıştır */
   return 0;
}
```

Tekli İşleçler

Tekli işleçler (unary operator) sadece bir işlenen üzerinde işlem yapar. Tekli demek tek bir işlenen ile işlem yapması anlamındadır.

İşleç	İşlenenler (operand) üzerinde yapılan işlem
-	Kendisinden sonra gelen işlenenin işaretini değiştirir. İşlenen pozitif ise negatif yapar, negatif ise pozitif yapar.
+	Kendisinden sonra gelen işlenenin işaretini pozitif yapar.
++	Artırım (increment) işleci, işlenenin sağında veya solunda kullanılabilir. İşlenen solda ise işlenenin değeri kullanılmadan önce 1 artırılacaktır (pre-increment). İşlenen sağda ise işlenenin değeri kullanılmadan sonra 1 artırılacaktır (post-increment).
--	Eksiltme (decrement) işleci, işlenenin sağında veya solunda kullanılabilir. İşlenen solda ise işlenenin değeri kullanılmadan önce 1 eksiltilecektir (pre-decrement). İşlenen sağda ise işlenenin değeri kullanılmadan sonra 1 eksiltilecektir (post-decrement).
!	Mantıksal DEĞİL işleci işlenenden önce kullanılır. İşleneninin mantıksal durumunu tersine çevirmek için kullanılır.
sizeof()	`sizeof()` işleci kendisinden sonra gelen işlenenin bellek boyutunu bayt cinsinden döndürür.
&	Adres işleci (address operator), kendisinden sonra gelen işlenenin bellek adresini döndürür.

Tablo 9. Tekli İşleçler

Aşağıda tekli işleçlerin kullanımına ilişkin örnek programı lütfen inceleyiniz.

```
int main() {
   int a = 5, b = 5, res;

   res= -a; // tekli eksi (unary minus): res=-5;

   res = ++a+2; /* ++a işlemi önce-artırım (pre-increment) olarak
                   adlandırılır. a değişkeni işleme girmeden önce
                   artırılır. İşlem sonucu res=8, a=6 */
   res = 2+b++; /* b++ işlemi sonra-artırım (Post-Increment) olarak
                   adlandırılır. b değişkeni işleme girdikten sonra
                   artırılır. İşlem sonucu res=7, b=6 */

   a=5; b=5;    /* iki talimat(statement) yan yana yazılmıştır.
                   Daha fazla da yazılabilir. */
   res = --a+2; /* Pre-Decrement: işlemden sonra res=6, a=4 */

   res = 2+b--; /* Post-Decrement: işlemden sonra res=7, b=4 */
```

```
   a=1; b=0;
   res=!a; // işlem sonucu res=0
   res=!b; // işlem sonucu res=1

   res=sizeof(char); /* char veri tipinin bellek miktarını öğrenme: işlem sonucu res=1 */
   res=sizeof a;     /* a değişkeninin bellek miktarını öğrenme: işlem sonucu res=4 */
   return 0;
}
```

İlişkisel İşleçler

İlişkisel işleçler (relational operator), işlenenlerin arasındaki ilişkiyi verirler. C programlama dilinde, yalnızca doğru (true) ya da yanlış (false) değer tutabilen mantıksal veri tipi (data type) bulunmaz. Programcı kodlama yaparken tamsayı veri tiplerini bunun için kullanır. Tamsayının değeri `0` değilse YANLIŞ, sıfırdan farklı ise DOĞRU olarak anlamlandırılır.

İşleç	İşlenenler (operand) üzerinde yapılan işlem
==	Eşit mi? İşleci: İki işlenenin değeri birbirine eşit ise 1, değilse 0 verir
!=	Farklı mı? İşleci: İki işlenenin değeri birbirinden farklı ise 1, değilse 0 verir
>	Büyük mü? İşleci: soldaki işlenenin değeri soldakinden fazla ise 1, değilse 0 verir
<	Küçük mü? İşleci: soldaki işlenenin değeri soldakinden az ise 1, değilse 0 verir
>=	Büyük veya eşit mi? İşleci: soldaki işlenenin değeri soldakinden fazla ya da iki işlenen eşit ise 1, değilse 0 verir
<=	Küçük veya eşit mi? İşleci: soldaki işlenenin değeri soldakinden az ya da iki işlenen eşit ise 1, değilse 0 verir

Tablo 10. İlişkisel İşleçler

Aşağıda verilen ilişkisel işleçlerin kullanımına ilişkin örnek programı lütfen inceleyiniz.

```
int main() {
   int a = 5, b = 6, res;

   res = a<b;  // işlem sonucu res=1
   res = a<=b; // işlem sonucu res=1
   res = a>b;  // işlem sonucu res=0
   res = a>=b; // işlem sonucu res=0
   res = a==b; // işlem sonucu res=0
   res = a!=b; // işlem sonucu res=1
   return 0;
}
```

Bit Düzeyi İşleçler

Bit düzeyi işleçler (bitwise operator) özellikle ikilik tabandaki tamsayılarda karşılıklı bitler üzerinde yapılan işlemlerdir. Aslında bu işlemler çarpma, toplama, bölme ve çıkarma işlemleri için kullanılır.

İşleç	İşlenenler (operand) üzerinde yapılan işlem
&	Bit düzeyi VE: Tamsayının karşılıklı bitleri VE işlemine tabi tutulur. Sonuç tamsayıda karşılaştırılan iki bit 1 ise sonuç bit 1, değilse 0 olacak şekilde işlem yapar.
|	Bit düzeyi VEYA: Tamsayının karşılıklı bitleri VEYA işlemine tabi tutulur. Sonuç tamsayıda karşılaştırılan iki bitin herhangi biri 1 ise sonuç bit 1, değilse 0 olacak şekilde işlem yapar.
^	Bit düzeyi ÖZEL VEYA: Tamsayının karşılıklı bitleri ÖZEL VEYA işlemine tabi tutulur. Sonuç tamsayıda karşılaştırılan iki bitin her ikisi farklı ise sonuç bit 1, değilse 0 olacak şekilde işlem yapar.
~	Bit düzeyi NOT: Tekli işleç olup sonrasında gelecek sayının bitlerini 1 ise 0, 0 ise 1 olacak olacak şekilde işlem yapar.
<<	Bit düzeyi sola kaydırma: Tekli işleç olup sonrasında gelecek sayının bitlerini bir bit sola kaydırır. En anlamlı bit taşar. Taşan bu bit 1 ise işlemcinin durum kaydedicisinde taşma bayrağı 1 olur.

>>	Bit düzeyi sağa kaydırma: Tekli işleç olup sonrasında gelecek sayının bitlerini bir bit sağa kaydırır. En anlamsız bit taşar. Taşan bu bit 1 ise işlemcinin durum kaydedicisinde taşma bayrağı 1 olur.

Tablo 11. Bit Düzeyi İşleçler

Aşağıda tamsayılar üzerinde yapılan bit düzeyi işlemler gösterilmiştir;

```
int main() {
   int a = 6; //  0110
   int b = 2; //  0010

   int bitwise_and = a & b; // 0010
   int bitwise_or = a | b;  // 0110
   int bitwise_xor = a ^ b; // 0100
   int bitwise_not = ~b;    // 1101
   int left_shift = b << 2; // 0100
   int right_shift = b >> 1;// 0001
   return 0;
}
```

Mantıksal İşleçler

Mantıksal ifadeler doğru (true) ve yanlış (false) olabilen ifadelerdir. C dilinde mantıksal bir veri tipi yoktur ve `int` veri tipi bunun için kullanılır. Sıfırdan farklı bir tamsayı ifade doğru, sıfır olan tamsayı yanlış olarak değerlendirilir.

C Dilinde VE(AND), VEYA(OR), ÖZEL VEYA(XOR) ve DEĞİL (NOT) gibi mantıksal ifadelerin işlenmesi için aşağıdaki işleçler kullanılır. Mantıksal işleçlere (logical operator) işlenen olarak giren ifadeler öncelikle `int` veri tipine dönüştürülür. Yani bu işleçler için beklenen işlenen (operand) bir tamsayı ifadedir.

İşleç	İşlenenler (operand) üzerinde yapılan işlem
&&	Şartlı Mantıksal VE: Eğer iki işlenen sıfırdan farklı ise 1 verir. Soldaki işlenen yanlış ya da sıfır ise sağdakine bakılmaz.
||	Şartlı Mantıksal VEYA: Eğer iki işlenenden biri sıfırdan farklı ise 1 verir. Soldaki işlenen doğru ya da sıfırdan farklı ise sağdakine bakılmaz.
!	Mantıksal DEĞİL: İşlenenden önce kullanılır. Tekli işleçlerde verilmiştir. İşleneninin mantıksal durumunu tersine çevirmek için kullanılır. Eğer işlenen sıfırdan farklı ise 0, değilse 1 verir.

Tablo 12. Mantıksal İşleçler

Aşağıda verilen ilişkisel ve mantıksal işleçlerin kullanımına ilişkin örnek programı lütfen inceleyiniz.

```
int main() {
   int a = 5, b = 6;
   int res;
   res = a&&b; // işlem sonucu res=1
   res = a||b; // işlem sonucu res=1
   res = !a;   // işlem sonucu res=0
   return 0;
}
```

Atama işleçleri

Atama işleçleri (assignment operator) en çok kullanılan işleçlerdir. Atama işleçleri her zaman sağdaki değeri sola atar! Dolayısıyla atama yapılacak uygun veri tipinde bir değişken (variable) olmak zorundadır.

İşleç	İşlenenler (operand) üzerinde yapılan işlem
=	Sağdaki işleneni sola atar.
+=	Soldaki işleneni sağdaki işlenen üzerine toplar ve sonucu sola atar.
-=	Soldaki işlenenden sağdaki işlenenden çıkarır ve sonucu sola atar.
*=	Soldaki işleneni sağdaki işlenenle çarpar ve sonucu sola atar.

`/=`	Soldaki işleneni sağdaki işlenene böler ve sonucu sola atar. Eğer işlenenler tamsayı ise kesirli kısım göz ardı edilir. Bu durumda sonuç tamsayıdır.
`%=`	Bu işleç tamsayılarla çalışır. Soldaki işleneni sağdaki işlenene böler ve kalanı sola atar.

Tablo 13. Atama İşleçleri

Aşağıda verilen atama işleçlerin kullanımına ilişkin örnek programı lütfen inceleyiniz.

```
/* Bu program, atama işleçleri örneğidir. */
int main() {
  int a = 10; /* a değişkenine 10 değişmezi = işleciyle atanıyor.*/
  a += 10;    /* a=a+10; ile eşdeğer: a değişkenine 10 ekleniyor. */
  a -= 10;    /* a=a-10; ile eşdeğer: a değişkeninden 10 çıkarılıyor. */
  a *= 10;    /* a=a*10; ile eşdeğer: a değişkeni 10 kat yapılıyor.*/
  a /= 10;    /* a=a/10; ile eşdeğer: a değişkeni 1/10 kat yapılıyor.*/
  a %= 10;    /* a=a%10; ile eşdeğer: a değişkeninin 10 a kalanı bulunup
                 ona eşitleniyor. */
  return 0;
}
```

İşleç Öncelikleri

Cebirsel ifadelerde nasıl ki önce çarpma ve bölme sonra toplama işlemleri yapılıyor. C programlama dilinde yazılan ifadelerde de işleç öncelikleri (operator precedence) vardır. İfadelerde bir işlemi öncelikli hale getirmek için parantez `()` içerisine alınır. En içteki parantez en öncelikli olarak gerçekleştirilir. Başka işleçler de bulunmaktadır bunlar ilerleyen bölümlerde yeri geldikçe anlatılacaktır.

Öncelik	İşleç Grubu	İşleçler	Gruptaki Öncelik
1	Sonek (postfix)	`() ++ --`	Soldan Sağa
2	Tekli (unary)	`+ - ! ++ -- sizeof()`	Sağdan Sola
3	Çarpım	`* / %`	Soldan Sağa
4	Toplam	`+ -`	Soldan Sağa
5	İlişkisel (relational)	`< <= > >=`	Soldan Sağa
6	Eşitlik (equality)	`== !=`	Soldan Sağa
7	Bit düzeyi VE	`&`	Soldan Sağa
8	Bit düzeyi ÖZEL VEYA	`^`	Soldan Sağa
9	Bit Düzeyi VEYA	`|`	Soldan Sağa
10	Mantıksal VE	`&&`	Soldan Sağa
11	Mantıksal VEYA	`||`	Soldan Sağa
12	Atama (assignment)	`= += -= *= /= %= <<= >>= &= ^= |=`	Sağdan Sola

Tablo 14. İşleçlerin İşlem Öncelikleri

Aşağıda işleçlerin önceliklerine ilişkin örnek programı lütfen inceleyiniz.

```
/* Bu program, işleç öncelikleri örneğidir. */
int main() {
   int a=10, b=5;
   a= a-b+2; // Öncelik Sırası +, -, = İşlem sonunda a=7
   a= a/b+2; // Öncelik Sırası /, +, = İşlem Sonunda a=7/5+2=3
   a= 2*10-20/2+3+(12-10); // Öncelik Sırası: (-) * / - + +
   /*
    İfadenin çözüm sırası:
    > 2*10-20/2+3+2
    > 20-20/2+3+2
    > 20-10+3+2
    > 10+3+2
    > 13+2
    > 15
   */
   return 0;
}
```

Kayan Noktalı Sayı Hesapları

Kayan noktalı sayıların IEEE-754 standardında ikili tabanda tutulduğu *Değişken* başlığında anlatılmıştı. Kayan noktalı sayılar olarak değişkenlere verdiğimiz onluk tabanda değerler aslında arka planda ikilik tabanda tutulur ve bu durum yazdığımız kodlarda garip davranışların ortaya çıkmasına sebep olur. Buna örnek olarak; hemen hemen her programcının yaptığı ilk hata, aşağıdaki kodun amaçlandığı gibi çalışacağını varsaymaktır;

```
float total = 0;
for(float a = 0; a != 2; a += 0.01f) {
    total += a;
}
```

Acemi programcı bunun 0, 0.01, 0.02, 0.03, gibi artarak, 1.97, 1.98, 1.99 aralığındaki her bir sayıyı toplayacağını ve 199 sonucunu, yani matematiksel olarak doğru cevabı vereceğini varsayar. Bu kodda doğru yürümeyen iki şey olur: Birincisi yazıldığı haliyle program asla sonuçlanmaz. `a` asla 2'ye eşit olmaz ve döngü asla sonlanmaz. İkincisi ise döngü mantığını `a < 2` şartını kontrol edecek şekilde yeniden yazarsak, döngü sonlanır, ancak toplam 199'dan farklı bir şey olur. IEEE754 uyumlu işlemlerde, genellikle bunun yerine yaklaşık 201'e ulaşır. Bunun olmasının nedeni, kayan noktalı sayılarının onluk tabanda kendilerine atanan değerlerin, ikilik tabanda yaklaşık değerlerini temsil etmesidir. Bu duruma aşağıdaki klasik örnek verilir;

```
#include <stdio.h>
int main()
{
    double a = 0.1;
    double b = 0.2;
    double c = 0.3;
    if(a + b == c)
        printf("Bu satır İcra Edilmez!\n"); //IEEE754 standardında işlem yaparsak.
    else
        printf("Kayan Noktalı Sayılar İkili Sayı Sisteminde"
               " Tutulduğundan  Yaklaşık Olarak Hesaplanır." );
    return 0;
}
```

Programcı olarak gördüğümüz şey onluk tabanda yazılmış üç sayı olsa da derleyicinin (ve altta yatan donanımın) gördüğü şey ikili sayılardır. 0.1, 0.2 ve 0.3, ona mükemmel bölmeyi gerektirdiğinden (ki bu onluk sayı sisteminde oldukça kolaydır), ancak ikili sayı sisteminde imkansızdır (bu sayılar, onluk sayı sistemindeki 1/3 sayısı gibi 0.333333333333333... şeklinde kesin olmayan biçimde depolanması gerektiğindendir);

```
#include <stdio.h>
int main()
{
    /*64 bitlik kayan noktalı sayılar, tam sayı kısmı dahil olmak üzere
      53 basamaklı hassasiyete sahiptir *
    double a = 0011111110111001100110011001100110011001100110011001100110011010;
    //onluk sistemdeki 0.1 ikili sistemde kusurlu olarak saklanır
    double b = 0011111111001001100110011001100110011001100110011001100110011010;
    //onluk sistemdeki 0.2 ikili sistemde kusurlu olarak saklanır
    double c = 0011111111010011001100110011001100110011001100110011001100110011;
    //onluk sistemdeki 0.3 ikili sistemde kusurlu olarak saklanır
    double a + b = 0011111111010011001100110011001100110011001100110011001100110100;
    //c değişkeni ile a+b nin onluk sistemde 0,3'e tam olarak eşit olmadığını unutmayın!
    return 0;
}
```