

C PROGRAM YAPISI

```
//Ornek : mil olarak girilen sayıyı km 'ye çevirir
```

açıklama satırları

Önişlemci direktifleri

```
#include <stdio.h>
#define MIL_KM 1.609
```

```
//printf, scanf fonksiyonlarına ait kutuphane
//çevirme sabiti
```

Ana Fonksiyon

```
main()
{
```

sabit

```
float mil ;
float km ;
```

değişkenler

```
//girisin okunması
```

```
printf ("Mil uzunlugunu giriniz :\n");
scanf ("%f" , &mil);
```

Fonksiyonlar

```
//mil 'in km 'ye çevrilmesi
km = MIL_KM * mil;
```

Operatörler

```
//sonucun ekrana yazdırılması
printf ("%3.1f mil = %4.2f km 'dir", mil, km);
```

Noktalama işaretleri

Ana Fonksiyon Başlangıç ve bitişi

```
}
```

C programları belirli bir yapıya sahiptir ve kendine özgü bazı bileşenleri bulunmaktadır. İlk bilinmesi gereken nokta:

Bir C programı bir veya daha fazla fonksiyondan oluşmakta ve herbir fonksiyon bir veya daha fazla sayıda deyim içermektedir.

C'de bir fonksiyon, **altprogram** (subroutine) olarak da adlandırılır ve bunlar programın bir başka yerinden isimleri ve/veya parametreleri kullanılarak çağrılabilir.

C Program Yapısı

1. C’de küçük büyük harf ayrımı söz konusudur.
2. Verilen program 3 temel bloktan oluşmaktadır : Ön işlemci direktif(ler)i, main fonksiyonu ve açıklama satırları.

ÖN İŞLEMCI DİREKTİFLERİ

1. C programları kendi derleyicisi ile de ilişki halindedir. Bu ilişki C **önışlemcisi** yardımıyla sağlanır. Önışlemciler çeşitli emirlerden oluşabilir. Bu emirler C derleyicisinin kaynak kodunu denetlemekte kullanılır. Önışlemci emirleri C programı içinde (#) işareti ile başlar. C'nin en çok kullanılan önışlemci emirleri **#include** ve **#define** ile tanımlanmaktadır. Önışlemci emirleri (;) işareti ile sonlandırılmaz. Ön işlemci direktifleri, programın başında yürütülecek özel deyim veya komutların bir topluluğudur.
2. Ön işlemci direktifleri kare karakteri (#) ile başlar, arkasından ön işlemci ismi gelir ve gerekli argümanların dizilmesiyle son bulur.
3. include direktifi, açılış parantezleri içinde belirtilen bir dosya ismi gerektirir.
4. < , > açılış parantezleri, ilgili dosyanın C nin bir parçası olan include klasörü içinde olduğunu belirtir.
5. < , > açılış parantezleri yerine “ , ” çift tırnak da kullanılabilir. Çift tırnak, ilgili dosyanın geçerli klasör veya C'nin include klasörü içinde bulunabileceğini bildirir.
6. **#include** direktifi, genellikle **.h** uzantılı dosyaları kabul eder. Bu dosyalar header dosyaları olarak adlandırılır : **stdio.h** gibi
7. **stdio.h** dosyası, **standart C kütüphanesinde** tanımlı bazı fonksiyonların prototip tanımlamalarını içerir.
8. **stdio.h** için C kütüphanesinde tanımlanan fonksiyonlar **standart I/O** (temel giriş çıkış: klavye ve ekran) işlemlerini gerçekleştirir.
9. Ön işlemci, header dosyasının içeriğini, yazıldığı noktadan itibaren programa ekler. Ön işlemci direktifi, kendisini takip eden kod için geçerlidir. Bu sebeple genellikle programın başına yazılırlar.
10. Bir programda, birden fazla ön işlemci direktifi kullanılabilir. Bu durumda her direktif ayrı satırda yazılmalıdır.
11. **define** direktifi **makro** tanımlamak için kullanılır.

12. İki tip makro tanımlanabilir : statik (sabit) ve dinamik
#define NAME value // ile sabit tanımlanır

Örn :
#define PI 3.14
#define UZUNLUK 100

main FONKSİYONU

1. Bir C programı main fonksiyonsuz çalıştırılmaz.
2. Main fonksiyon bloğu, fonksiyon ismi (main), bir çift parantez ve bunların arkasından gelen bir çift küme parantezinden oluşur.
3. Parantez içerisinde fonksiyonun kabul edeceği argümanlar listelenirken, küme parantezleri içinde main fonksiyonunun yerine getireceği deyimler veya komutlar yer alır.
4. Bu deyimler veya komutlar main fonksiyonun gövdesini teşkil eder.
5. C’de her deyim noktalı virgül ile sonlandırılır.
6. Main fonksiyonunun gövdesi, kullanıcı-tanımlı fonksiyonların veya C nin kendi yerleşik kütüphane fonksiyonlarının çağrılmasını içerebilir.
7. Kullanıcı tanımlı fonksiyonlar, main programından önce veya sonra tanımlanabilir.
8. Yukarıdaki örnekte kütüphane fonksiyonları scanf ve printf’in çağrılması görünmektedir.

C nin Temel Elemanları

1. Anahtar kelimeler (keywords)
2. Tanıtıcılar (identifiers)
3. Sabitler (constants)
4. Operatörler
5. Noktalama işaretleri (punctuators)

Tanıtıcılar

1. İki tip tanıtıcı mevcuttur : standart tanıtıcılar ve kullanıcı tanımlı tanıtıcılar.
2. C’de tanıtıcılar, bir değişkenin veya bir fonksiyonun ismidir. Bir tanıtıcı, sadece bir değişken veya bir fonksiyona ait olur.
3. Tanıtıcılar, harflerin, rakamların ve/veya özel karakterlerin arka arkaya dizilmesiyle oluşturulur.
4. Tanıtıcılar, programların değişken elemanlarıdır. toplam, i, N, mil, km, printf, scanf ...
6. Büyük küçük harf ayrımı mevcuttur.

Sabitler

1. Bütün program boyunca aynı değere sahip elemanlar sabit olarak adlandırılır.
2. integer, floating point, character, string gibi değişik tiplere sahip sabitler tanımlanabilir. 1, 10, 23.4567, 3.14, ‘a’, ‘x’, ‘6’, “isim”, “mavi” ...

Operatörler

1. Operatörlerin unary, binary ve ternary şeklinde üç tipi mevcuttur.
2. Unary operatör, bir adet eleman (operand) üzerinde; binary operatör iki operand üzerinde ve ternary operatör üç operand üzerinde bir işlem gerçekleştirir.
3. Operatör Tipleri :
Aritmetik Operatörler : +, -
Atama Operatörleri : =, += ...
Karşılaştırma Operatörleri : <, >=,
Mantıksal Operatörler : &&, ||, ...
.....

Noktalama İşaretleri

1. Noktalama işaretleri, derleyici için sözdizimsel anlama sahiptir.
2. Fakat, bir değer üretecek bir işlem gerçekleştirmezler. {, }, [,], ;, ,

Veri Tipleri

Veri tipi (data type) program içinde kullanılacak değişken, sabit, fonksiyon isimleri gibi tanımlayıcıların tipini ve bellekte ayrılacak bölgenin büyüklüğünü, belirlemek için kullanılır.

C programlama dilinde dört tane temel veri tipi bulunmaktadır. Bunlar:

```
char
int
float
double
```

Fakat bazı özel niteleyiciler vardır ki bunlar yukarıdaki temel tiplerin önüne gelerek onların türevlerini oluşturur. Bunlar:

```
short
long
unsigned
```

Bu niteleyiciler sayesinde değişkenin bellekte kaplayacağı alan isteğe göre değiştirilebilir. Kısa (**short**), uzun (**long**), ve normal (**int**) tamsayı arasında yalnızca uzunluk farkı vardır. Eğer normal tamsayı 32 bit (4 bayt) ise uzun tamsayı 64 bit (8 bayt) uzunluğunda ve kısa tamsayı 16 biti (2 bayt) geçmeyecek uzunluktadır. İşaretsiz (**unsigned**) ön eki kullanıldığı taktirde, veri tipi ile saklanacak değerın sıfır ve sıfırdan büyük olması sağlanır. İşaretsiz ve işaretsiz verilerin bellekteki uzunlukları aynıdır. Fakat, işaretsiz tipindeki verilerin üst limiti, işaretlinin iki katıdır.

Aşağıdaki Tablo 'da bütün tipler, bellekte kapladıkları alanlar ve hesaplanabilecek (bellekte doğru olarak saklanabilecek) en büyük ve en küçük sayılar listelenmiştir.

Değişken tipleri ve bellekte kapladıkları alanlar

Veri Tipi	Açıklama	Bellekte işgal ettiği boyut (bayt)	Alt sınır	Üst sınır
char	Tek bir karakter veya küçük tamsayı için	1	-128	127
unsigned char			0	255
short int	Kısa tamsayı için	2	-32,768	32,767
unsigned short int			0	65,535
int	Tamsayı için	4	-2,147,483,648	2,147,483,647
unsigned int			0	4,294,967,295
long int	Uzun tamsayı için	8	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
unsigned long int			0	18,446,744,073,709,551,615
float	Tek duyarlı gerçel sayı için (7 basamak)	4	-3.4e +/- 38	+3.4e +/- 38
double	Çift duyarlı gerçel sayı için (15 basamak)	8	-1.7e +/- 308	+1.7e +/- 308

char

1. Tek karakter depolamak için kullanılır.
2. Örn : 'a', 'e', 'k', 'A', 'X', '5' (karakter sabitler)
3. Küçük harf – büyük harf ayrı değerlendirilir.
4. char veri tipinin depoladığı değer aralığı -128 ile 127 arası
5. unsigned char ise 0 – 255 arası değer depolar.

Değişkenler

Değişkenler bilgisayarın geçici belleğinde bilginin saklandığı gözlere verilen sembolik adlardır. Bir C programında, bir değişken tanımlandığında bu değişken için bellekte bir yer ayrılır.

C programı içinde kullanılacak bir değişkenin veri tipini bildirmek için aşağıdaki şekilde bir tanım yapılır.

veri_tipi değişken_adi;

Örneğin, C programı içinde sayma işlemlerini yerine getirmek için sayaç isimli bir değişkenin kullanılması gerektiğini varsayalım. Bu değişken aşağıdaki şekilde bildirilir.

int sayac;

Yapılan bu bildirimde göre, sayaç isimli değişken program içinde tamsayı değerler içerecektir. Bildirim satırları da aynen diğer C deyimleri gibi (;) işareti ile son bulmalıdır.

Bazı uygulamalarda değişkenin bir başlangıç değerinin olması istenir. Böyle durumlarda değişken bildirilirken başlangıç değeri verilebilir. Örneğin:

```
char isim='X', z; // değer atamak zorunlu değil
int sayi=0, n;
float toplam=0.0, sonuc=22.14;
```

Değişken isimleri verirken bazı kurallara uymak zorunludur. Bunlar:

1. C’de bütün değişkenler bir harf ile veya alt çizgi (_) karakteri ile başlamak zorundadır.
2. İlk karakterden sonra harfler, rakamlar ve/veya alt çizgi karakteri gelebilir.
3. İlk 31 karakter gözönüne alıdır. Diğerleri göz ardı edilir.
4. Büyük harfler, küçük harflerden farklı değerlendirilir.
5. ANSI C’nin anahtar kelimeleri değişken ismi olarak kullanılamaz.
 - auto double int struct
 - break else long switch
 - case enum register typedef
 - char extern return union
 - const float short unsigned
 - continue for signed void
 - default goto sizeof volatile
 - do if static while
6. Özel karakterler değişken isimlendirmede kullanılamaz.
Örn : #toplam, \$sayi, ...
7. Değişken adları İngiliz alfabesinde bulunan karakterler (A-Z) veya (a-z) yada rakamlar (0-9) ile yazılmalıdır. Türkçe karakterler, özel karakter veya boşluk karakteri kullanılamaz.
8. Değişkenler için verilen isimlendirme kuralları sabitler için de geçerlidir.
9. Geleneksel olarak, sabitleri büyük harf kullanarak isimlendirmek tercih edilir.

Sabitler

Sabitler C programı içinde **const** sözcüğü ile tanımlanır. Bu sözcük aşağıdaki şekilde kullanılmaktadır:

```
const tip sabit_adi=değeri;
```

Program içinde tamsayıları belirtmek için **int**, karakterler için **char** ve kayan noktalı değerler içeren değişmezleri tanımlamak için **float** sözcüğü kullanılır.

Örneğin:

```
const float    PI = 3.142857;  
const double NOT= 12345.8596235489;  
const int     EOF= -1;  
const char    SINIF = 'S';
```

gibi sabit bildirimleri geçerli olup bunların içerikleri program boyunca değiştirilemez. Yalnızca kullanılabilir. Genellikle, sabit olarak bildirilen değişken isimleri büyük harflerle, diğer değişken isimlerinin ise küçük harflerle yazılması (gösterilmesi) C programcıları tarafından geleneksel hale gelmiştir.

Birçok C programında sabitler **#define** önışlemci komutu ile de tanımlanabilir. Bu komutla sabit bildirimini, bir program parçasına ve makro fonksiyon tanımlaması yapılabilir. Bir program geliştirilirken simgesel sabitlerin kullanılması programın okunurluğunu artırır ve bazen gerekli de olabilir.

```
#define MAX      100  
#define DATA   0x0378    //önışlemci direktifi  
#define YARICAP 14.22
```

Atama Operatörü

C programlama dilinde atama operatörü olarak = işareti kullanılır.

```
i = 25;        // 25, int tipinde bir rakamsal bilgidir  
r = 17.2;     // 17.2, double tipinde bir rakamsal bilgidir
```

Tamsayı (**int**) rakamsal bilgiler, **8 (oktal)** ve **16 (hexadesimal)** sayı tabanında da gösterilebilir. Bunun için sabit rakamın başına, 8 tabanı için **0** (sıfır) ve 16 tabanını için **0x** sembolleri eklenir. 16'lık sistemdeki harfler büyük (A, B, C, D, E ve F) veya küçük (a, b, c, d, e ve f) olabilir. Bu gösterime göre, aşağıdaki atamalar aynı anlamadadır:

```
i = 11;      // i = 11, (10) tabanında
j = 013;    // j = 13, (8) tabanında
k = 0xb;    // k = b, (16) tabanında
l = 0xB;    // l = B, (16) tabanında
```

Gerçel sayılar *ondalıklı* veya *üstel* olmak üzere iki biçimde gösterilebilir. Örneğin 123.456 sayısının aynı anlama gelen dört farklı gösterimi aşağıda verilmiştir. Üstel gösterimde, 1.23456e+2 veya 1.23456E+2 sayısı matematikteki 1.23456×10^2 gösterimi ile eşdeğerdir.

```
x = 123.456;      // ondalıklı gösterimi
y = 123.456e+0;   // üstel gösterim
z = 1.23456e+2;   // üstel gösterim
o = 1234.56E-1;   // üstel gösterim
```

Karakter sabitler, bir harf için tek tırnak, birden çok karakter için çift tırnak içinde belirtilirler.

```
c = 'A'           // bir karakter
d = "C Programlama Dili" // bir karakter kümesi
```

Değişken Bildirim Yerleri ve Türleri

C programları içinde farklı amaçlara yönelik değişken tanımlamaları yapılabilir. Değişken tanımlama tipleri aşağıdaki şekilde sıralanır.

1. **Yerel** değişkenler
2. **Evrensel** değişkenler
3. **Extern** değişkenler
4. **Static** değişkenler
5. **Auto** değişkenler
6. **Register** değişkenler

Yerel Değişkenler

Değişken veri türü bildirimleri bir fonksiyonun içinde yada dışında yapılabilir.

Veri türünün fonksiyon içindeki veya dışındaki bildirimi farklı sonuçlara neden olacaktır. Çünkü, fonksiyon içinde bildirimi yapılan bir değişken, sadece o fonksiyon için geçerlidir. Yerel değişkenler, program yürütüldüğünde aktif hale geçerek kendisi için ayrılan bellek alanlarını kullanır. Ancak yer aldıkları blok sona erdiğinde bu bellek alanları iptal olur ve değişken içeriği tamamen yok olur. Aynı blok daha sonra tekrar başlasa bile, yerel değişkenler eski değerlerini alamaz.

Program içinde birden fazla fonksiyon varsa, sadece tanımlandığı fonksiyonda geçerli olabilecek değişkenlere **yerel değişken** (local variable) adı verilir.

```
#include <stdio.h>
```

```
fonk1()
{
    int i;
    printf ("i=%d",i);
}

main()
{
    int i=2;           //yerel değişken
    printf ("i=%d\n",i);
    fonk1();
    getch();
}
```

Evrensel Değişkenler

Eğer bir değişkenin program içindeki tüm fonksiyonlar için geçerli olması söz konusu ise, bu kez fonksiyonların dışında bir yerde bildirim yapılır. Aşağıdaki örnek üzerinde gösterildiği biçimde **i** değişkeni tanımlanacak olursa, bu tanım sadece **main()** fonksiyonunda değil, program içindeki tüm fonksiyonlar için geçerli olacaktır. Bu tür değişkenlere **evrensel değişken** (global variable) adı verilir.

```
#include <stdio.h>
```

```
int i=2; //evrensel değişken
```

```
fonk1()
{
    printf ("i=%d",i);
}

main()
{
    printf ("i=%d\n",i);
    fonk1();
    getch();
}
```

Örnek :

```
#include <stdio.h>
```

```
int i=2; //evrensel değişken
```

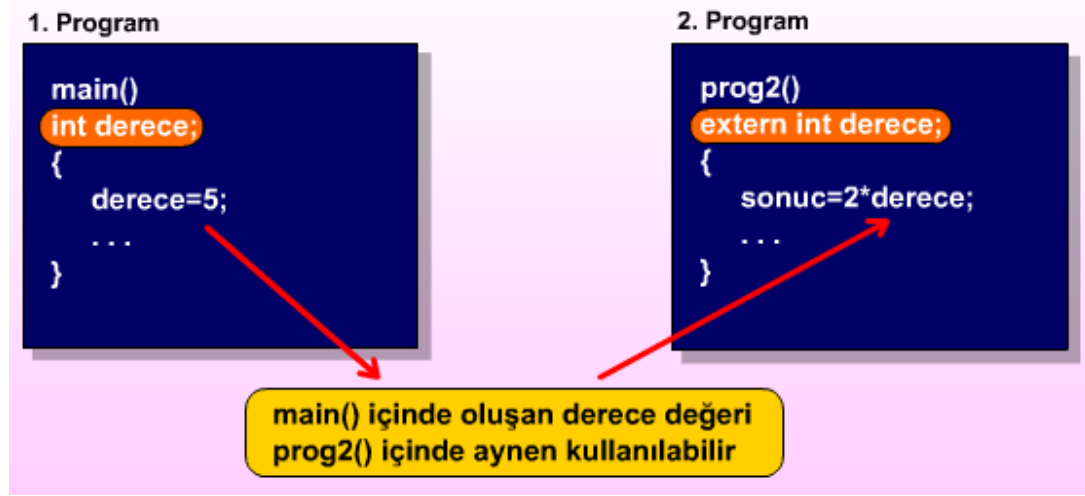
```
fonk1()
{
    int i;
    printf ("i=%d",i);
}

main()
{
    printf ("i=%d\n",i);
    fonk1();
    getch();
}
```

extern Değişkenler

Küresel değişkenlerin içerdiği değerlerin, programın sonuna kadar tüm fonksiyonları kapsayacak biçimde geçerli olduğunu biliyoruz. C dilinde uzun programlar dosyalar halinde bölünür. Dosyalar sonradan birleştirilirken, bir dosya için geçerli değişkenin diğer dosyalar içinde geçerli olabilmesi için küresel değişkenler **extern değişkenler** olarak tanımlanır.

Aşağıdaki şekil üzerinde görüldüğü gibi, iki farklı C programında **derece** isimli değişken birlikte kullanılabilir.



static Değişkenler

Bir değişken, yerel değişken olarak tanımlanmış ise, bu değişkenin yer aldığı fonksiyonun yürütülmesi sona erdiğinde değeri yok oluyordu. Bu durumu önleyerek, değişken değerinin sadece ilgili fonksiyon içinde değil, tüm program boyunca geçerli olması sağlanabilir. Böyle bir amaca ulaşmak için **static** değişkenler tanımlanır.

Örneğin, **derece** isimli değişken değerinin, fonksiyonun çalışması ardından yok olmasını istemediğimizi varsayalım. Bu durumda aşağıda belirtilen tanım yapılır.

Aşağıda yapılan tanımlar, **static** ile küresel değişkenlerin birbirine çok benzediğini göstermektedir. Burada şu farka dikkat etmek gerekir: **static** değişkenler, fonksiyonun yürütülmesi sonunda değerini kaybetmez; ancak bu değer diğer fonksiyonlar tarafından da kullanılamaz. Aynı fonksiyon yeniden yürütüldüğünde, daha önceki değer kullanılmamasını sağlar. Yani, **derece** isimli değişkenin değeri, fonksiyon yeniden yürütüldüğünde, önceki değer göz önüne alınarak işlemlere devam edilir.

```
main ()
{
    static int derece;
    ...
}
```

auto Değişkenler

C ile B dili arasında uyum sağlamak amacıyla otomatik değişken bildiriminde kullanılır. Yerel değişkendir. Otomatik değişkenler gerektiğinde oluşturulurlar ve işleri bitince silinirler.

register Değişkenler

Anlatılan değişkenlerin tümü bellek üzerindeki alanları kullanır. Bazı uygulamalarda, değişkenlere çok daha hızlı biçimde erişmek söz konusu olabilir. Böyle durumlarda, **register** değişkenleri tanımlanarak bellek alanları yerine, mikroişlemcinin hızlı bellek yazmaçları kullanılabilir. Ancak bu alanların miktarı sınırlı olduğundan (genellikle 2 tane), fazla sayıda **register** değişken tanımlanırsa amacına ulaşmaz. Derleyici fazla bulduklarını normal bellek alanlarına yerleştirir.

```
main ()
{
    register int k;
    {
        register int i;
        ...
    }
}
```

Tip Dönüşümleri

Bir formül içerisinde bir çok değişken veya sabit olabilir. Bu değişken ve sabitler birbirinden farklı tipte olursa, hesap sonucunun hangi tipte olacağı önemlidir. Bir bağıntıda, içeriği dönüşüme uğrayan değişkenler eski içeriklerini korurlar. Dönüştürme işlemi için geçici bellek alanı kullanılır; dönüştürülen değer kullanıldıktan sonra o alan serbest bırakılır.

NOT

Tamsayılar arası bölme kesme hatalarına (truncation error) neden olur.

Bunun anlamı iki tamsayının oranı yine bir tamsayıdır.

örneğin: $4/2=2$; ama $3/2=1$ (1.5 değil).

Cast (Belirgin dönüşüm)

Bir değişkenin, sabit değer veya bağıntının önüne tür veya takı (cast) yazılarak sonucun hangi tip çıkması istendiği söylenebilir. Genel yazım biçimi:

(tip) bağıntı;

Örneğin:

```
int x=9;
float a,b,c;
.
.
.
.
.
a = x/4;
b = x/4.0;
c = (float) x/4;
```

işleminin sonucunda **a** değişkenine 2.0, **b** ve **c** değişkenlerine 2.25 değeri aktarılır. Yani $9/4$ ile $9/4.0$ farklı anlamdadır.

Veri tipi büyükten küçüğe çevriliyorsa veri kaybı olur.

Örn : (int) (x + y)

x = 7 ve y = 8.5 ise dönüşüm sonucu 15 olur.

Temel Giriş/Çıkış Fonksiyonları

Temel giriş/çıkış fonksiyonları, bütün programla dillerinde mevcuttur. Bu tür fonksiyonlar, kullanıcıya ekrana veya yazıcıya bilgi yazdırmasına, ve bilgisayara klavyeden veri girişi yapmasına izin verir.

Girdi ve çıktı deyimleri gerçekte C dilinin bir parçası değildir. Yani, diğer programlama dillerinin tersine, C dilinin içine konmuş girdi/çıkı deyimleri yoktur. Girdi/çıkı işlemleri, her zaman, fonksiyonlar çağrılarak yapılır. Tabii ki, girdi/çıkı yapmak için kullanılan fonksiyonların programcı tarafından yazılmasına gerek yoktur. Hemen hemen bütün C ortamlarında girdi/çıkı fonksiyonları içeren standart kütüphaneler bulunmaktadır. Bu kütüphanelerde tanımlanmış bulunan fonksiyonlar (ile alabilecekleri argümanlar) ve ilgili birtakım değişkenlerin bildirisi ise bir başlık dosyasına konur. Herhangi bir standart girdi/çıkı fonksiyonu çağrılmadan veya değişkenleri kullanılmadan önce,

#include <stdio.h>

yazılarak kaynak programın içine kopyalanması gerekir.

Kullanıcının girdi/çıkı yapması için, üç girdi/çıkı ara dosyasının tanımı önceden yapılmıştır. Bunlar şunlardır:

stdin standart girdi; normalde bilgisayar klavyesidir, fakat daha farklı giriş elemanlarından da giriş alınabilir.

stdout standart çıktı; çoğunlukla bilgisayar ekranı olarak düşünülür, fakat daha farklı dış birimlere de çıkış gönderilebilir. Flashdisk veya yazıcılara gönderilen çıkış da **stdout** kanalları üzerinden gerçekleştirilir.

stderr standart hata çıktısı; hata mesajlarını ekrana bastırmak için kullanılan bir kanaldır.

printf() Fonksiyonu

Standart C kütüphanesinde bulunan printf() fonksiyonu, deęişkenlerin tuttuęu deęerleri, onların adreslerini veya bir mesajı ekrana belli bir düzenle (format) standart çıkışa (stdout), yani ekrana, yazdırmak için kullanılan fonksiyondur.

Kullanımı :

int printf(const char *format [, argument]);

Eđer format stringini takip eden argument'lar mevcut ise bu argument deęerleri format'ta verilen forma uygun olarak çıkışa yazar. Geri dönüş deęeri olarak, yazdığı karakter sayısını döndürür. Hata oluştuęunda negatif deęer döndürür.

*format üç kısımdan oluşmaktadır:

- I. **Düz metin (literal string):** yazdırılmak istenen ileti.
Örneęin:
printf("C Programlama Dili...");
- II. **Kontrol karakterleri (escape sequence):** deęişkenlerin ve sabitlerin nasıl yazılacağını belirtmek veya imlecin alt satıra geçirilmesi gibi bazı işlemlerin gerçekleştirilmesi için kullanılır. Bu karakterler alttaki Tablo 'da listelenmiştir.

Kontrol karakterleri

Karakter	Anlamı
\a	Ses üretir (alert)
\b	imleci bir sola kaydır (backspace)
\f	Sayfa atla. Bir sonraki sayfanın başına geç (formfeed)
\n	Bir alt satıra geç (newline)
\r	Satır başı yap (carriage return)
\t	Yatay TAB (horizontal TAB)
\v	Dikey TAB (vertical TAB)
\"	Çift tırnak karakterini ekrana yaz
'	Tek tırnak karakterini ekrana yaz
\\	\ karakterini ekrana yaz
%%	% karakterini ekrana yaz

III. **Tip belirleyici (conversion specifier):** % işareti ile başlar ve bir veya iki karakterden oluşur (%d gibi). Ekran yazdırılmak istenen değişkenin tipi, % işaretinden sonra belirtilir. Örneğin: printf("x in değeri %d dir"); gibi.

Tip karakterleri (Tamsayılar)

Tip Karakteri	Anlamı
%d	işaretsiz tamsayı (onluk sistem)
%i	işaretsiz tamsayı (onluk sistem)
%o	işaretsiz tamsayı (sekizlik sistem)
%u	işaretsiz tamsayı (onluk sistem)
%x yada %X	işaretsiz tamsayı (onaltılık sistem) x → 0..9,a,b,c,d,e,f X → 0..9,A,B,C,D,E,F
H	Sort tamsayı
L	Long tamsayı

*not : i ve d belirleyicileri scanf kullanımında farklılık gösterir.

Ornek :

```
#include <stdio.h>
```

```
main()
```

```
{
```

```
    int a=12, b=2;
```

```
    printf("\n");
```

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

```
    printf("\n%i\n",a);
```

```
    printf("%o\n",a);
```

```
    printf("%u\n",a);
```

```
    printf("%x\n",a);
```

```
    printf("%X\n",a+1);
```

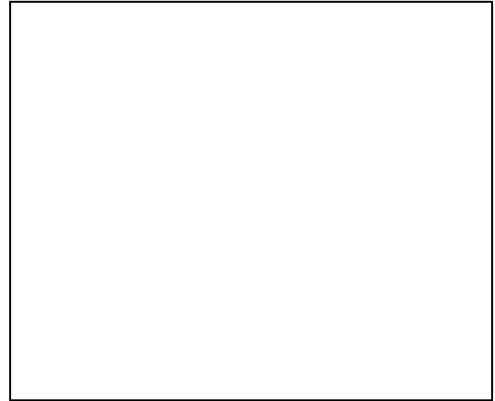
```
    printf("\n****\n");
```

```
    printf("\n%d+%d=%d 'dir\n\n",a,b,a+b);
```

```
    printf("( %d)10-( %d)10=( %x)16 'dir\n",a,b,a-b);
```

```
    getchar();
```

```
}
```



Tip karakterleri (Ondalikli sayilar)

%e yada %E	Ustel yazdirma
%f	Dogrudan yazdirma
%g yada %G	Ondalik degerleri f yada e bicimde yazmak (basamak sayisi 6) (Sayinin büyüklüğüne bağlı)

Ornek:

```
#include <stdio.h>
```

```
main()
```

```
{
```

```
    float sayi=12.3456;  
    printf("\n%f\n",sayi);  
    printf("%e\n",sayi);  
    printf("%E\n",sayi);  
    printf("%g\n",sayi);  
    printf("%G\n",sayi);
```

```
    getch();
```

```
}
```

```
12.345600  
1.234560e+01  
1.234560E+01  
12.3456  
12.3456
```

Tip karakterleri (Karakter veya karakter dizisi)

<code>%c</code>	tek bir karakter
<code>%s</code>	karakter dizisi (string)

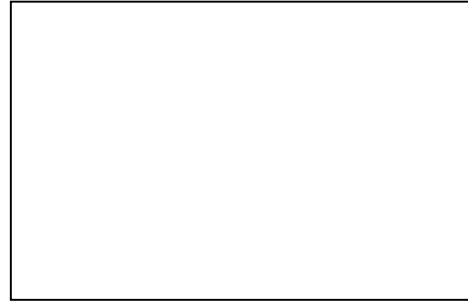
Ornek :

```
#include <stdio.h>
```

```
main()
{
    char a='z';
    char b[]="ALI";
    printf("\n%c\n",a);
    printf("%s\n",b);
    printf("\n\n");
    printf("%s %s %c", "Merhaba",b,a);

    getchar();

}
```



printf fonksiyonunun geri dönüş değeri **int** tipindedir. Bu geri dönüş değeri çıktının kaç karakter olduğunu gösterir. Yani, **printf** fonksiyonu, **format* ile tanımlanmış karakter topluluğunun kaç bayt olduğu hesaplar.

Ornek:

```
#include <stdio.h>
```

```
int main()
{
    int karSay;
    int sayi = 1234;

    karSay = printf("Sayi= %d\n",sayi);

    printf("Ust satirda toplam\nkarakter sayisi: %d dir\n", karSay);

    getchar();
}
```

```
Sayi= 1234
Ust satirda toplam
karakter sayisi: 11 dir
```

Formatlı Yazdırma

Bundan önceki programlardaki değişkenler serbest biçimde (free format), yani derleyicinin belirlediği biçimde ekrana yazdırılmıştı. Bazen giriş ve çıkışın biçimi kullanıcı tarafından belirlenmesi gerekebilir.

Alan Genisliği ve Duyarlilik

1. Verinin yazdırılacağı alanın kesin boyutları alan genişliği ile belirlenir.
2. Eğer alan genişliği yazdırılacak sayının genişliğinden büyük ise veri alan içinde otomatik sağa dayanır.
3. Tamsayılarda duyarlılık :
 - 3.1. Eğer yazdırılan değer belirtilen duyarlılıktan daha az basamağa sahip ise sayının önüne fark kadar sıfır konur.
 - 3.2. Tamsayılar için default duyarlılık değeri 1 dir.
4. Ondalık sayılarda duyarlılık :
 - 4.1. Ondalık kısımda yazdırılacak basamak sayısıdır (e, E, f için).
 - 4.2. Yazdırılacak önemli basamakların sayısıdır (g ve G için).
 - 4.3. Duyarlilik orginal değerdeki ondalık basamak sayısından küçük ise yumarlama olur.
5. String'lerde duyarlılık yazdırılacak karakter sayısıdır.

Kullanımı :

%[alan_genisligi][.duyarlilik]tip_karakteri

Ondalikli sayilar icin alan_genisligi > duyarlilik + 2 olmalıdır.

Ornekler :

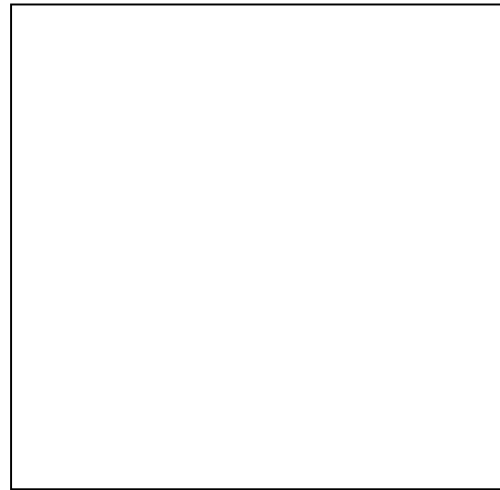
```
#include <stdio.h>
```

```
int main()
{
    int s1=1;
    int s2=12;
    int s3=123;
    int s4=1234;
    int s5=12345;

    printf("Alan Genisligi\n\n");
    printf("%3d\n",s1);
    printf("%3d\n",s2);
    printf("%3d\n",s3);
    printf("%3d\n",s4);
    printf("%3d\n",s5);

    printf("Duyarlilik\n\n");
    printf("%.3d\n",s1);
    printf("%.3d\n",s2);
    printf("%.3d\n",s3);
    printf("%.3d\n",s4);
    printf("%.6d\n",s5);

    getchar();
}
```



```
#include <stdio.h>
```

```
int main()
{
    char x[]="Kocaeli";

    printf("1234567890\n");
    printf("*****\n");
    printf("%10s\n",x);
    printf("%8s\n",x);
    printf("%5s\n",x);

    printf("%.2.1s\n",x);
    printf("%.10.4s\n",x);
    getchar();
}
```



```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    float d;
```

```
    d=123.456;
```

```
    printf("1234567890\n");
```

```
    printf("*****\n");
```

```
    printf("%f\n",d);
```

```
    printf("%8.0f\n",d);
```

```
    printf("%5.1f\n",d);
```

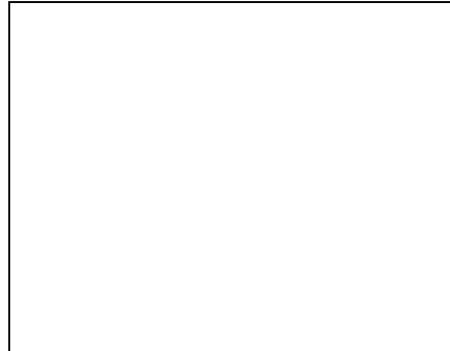
```
    printf("%8.2f\n",d);
```

```
    printf("%.3f\n",d);
```

```
    printf("%2.1f\n",d);
```

```
    getchar();
```

```
}
```



```
#include <stdio.h>
```

```
#define PI 3.14
```

```
main()
```

```
{
```

```
    int not=12,x,y;
```

```
    float a,c;
```

```
    char kr='A';
```

```
    a=(int) PI/2;
```

```
    x=PI/2;
```

```
    y=(float) PI/3;
```

```
    c=PI/2;
```

```
    printf("a=%.2f x=%d\nty=%d c=%.3f\n",a,x,y,c);
```

```
    printf("%5c=%.2f\n",kr,PI);
```

```
    printf("%2d+ %1d=%c\n",5,not,kr);
```

```
    getchar();
```

```
}
```



puts() Fonksiyonu

Ekrana yazdırılacak ifade bir karakter topluluğu ise, *printf()*'e alternatif olarak *puts()* fonksiyonu kullanılabilir.

Ancak *puts()*, ekrana bu karakter topluluğu yazdıktan sonra, imleci alt satıra geçirir. Buna göre:

```
printf("C programlama Dili.\n");  
ile  
puts("C programlama Dili.");
```

kullanımları eşdeğerdir.

puts() fonksiyonu kontrol karakterleri ile kullanılabilir.

```
puts("Bu birinci satır...\nBu ikinci satır.");
```

```
Bu birinci satır...
```

```
Bu ikinci satır.
```

```
—
```

putchar() Fonksiyonu

stdout çıkış kanalının geçerli pozisyonuna tek karakter yazmak için kullanılır.

Kullanımı :

```
int putchar(int c);
```

Örnek:

```
Char ch='a';  
putchar(ch);
```

veya

```
putchar('a');
```

```
#include <stdio.h>
```

```
main()
```

```
{
```

```
    char kr='b';
```

```
    putchar(kr);
```

```
    puts("");
```

```
    putchar('b');
```

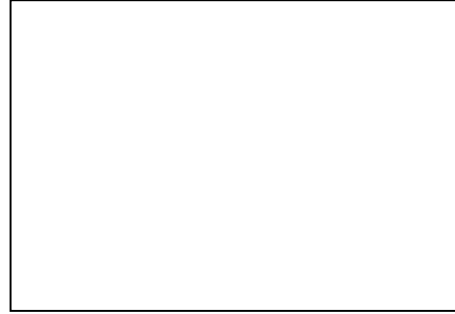
```
    printf("\n");
```

```
    putchar(kr+1);
```

```
    printf("\n%d %x %c",kr,'c'-1,kr);
```

```
    getchar();
```

```
}
```



Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	\$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(72	48	H	104	68	h
9	09	Horizontal tab	41	29)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	
29	1D	Group separator	61	3D	=	93	5D]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	á	192	C0	Ł	224	E0	α
129	81	ù	161	A1	í	193	C1	ł	225	E1	β
130	82	é	162	A2	ó	194	C2	ŧ	226	E2	Γ
131	83	â	163	A3	ú	195	C3	†	227	E3	π
132	84	ä	164	A4	ñ	196	C4	—	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	†	229	E5	σ
134	86	ã	166	A6	ª	198	C6	‡	230	E6	μ
135	87	ç	167	A7	º	199	C7	‡	231	E7	τ
136	88	ê	168	A8	¿	200	C8	Ł	232	E8	Φ
137	89	ë	169	A9	ƒ	201	C9	Ŧ	233	E9	Θ
138	8A	è	170	AA	ƒ	202	CA	Ł	234	EA	Ω
139	8B	ì	171	AB	½	203	CB	Ŧ	235	EB	δ
140	8C	í	172	AC	¾	204	CC	‡	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	=	237	ED	∞
142	8E	Ä	174	AE	«	206	CE	‡	238	EE	ε
143	8F	Å	175	AF	»	207	CF	Ł	239	EF	∩
144	90	É	176	B0	⋯	208	DO	Ł	240	FO	≡
145	91	æ	177	B1	⋮	209	D1	Ŧ	241	F1	±
146	92	Æ	178	B2	■	210	D2	π	242	F2	≥
147	93	ó	179	B3		211	D3	Ł	243	F3	≤
148	94	ö	180	B4	†	212	D4	Ł	244	F4	[
149	95	ò	181	B5	‡	213	D5	Ŧ	245	F5]
150	96	û	182	B6	‡	214	D6	π	246	F6	÷
151	97	ù	183	B7	π	215	D7	‡	247	F7	≈
152	98	ÿ	184	B8	ƒ	216	D8	‡	248	F8	°
153	99	ÿ	185	B9	‡	217	D9	ƒ	249	F9	•
154	9A	Û	186	BA		218	DA	ƒ	250	FA	·
155	9B	¢	187	BB	π	219	DB	■	251	FB	√
156	9C	£	188	BC	Ł	220	DC	■	252	FC	²
157	9D	¥	189	BD	Ł	221	DD	■	253	FD	³
158	9E	€	190	BE	Ł	222	DE	■	254	FE	■
159	9F	f	191	BF	ƒ	223	DF	■	255	FF	□

Temel Giriş/Çıkış Fonksiyonları (Devam)

scanf() Fonksiyonu

Birçok programda ekrana verilerin yazdırılması yanısıra klavyeden veri okunması gerekebilir. `scanf()` fonksiyonu klavyeden veri okumak için kullanılan fonksiyondur.

1. Değişkenlerin içerisine klavyeden değer atamak için kullanılır.
2. Fonksiyon ismi ve parametrelerden oluşur.
3. Parametre olarak, girilecek değerlerin hangi formatta olacağını bildiren girdi formatını ve bu formata göre girilecek değişkenler listesini alır.
4. `scanf` fonksiyonunda dışarıdan değer girilecek bütün değişkenlerin başına `&` işareti konur. (*Karakter dizilerinde bu işaret kullanılmaz*).
5. Bu işaret bellek operatörüdür, değişkenlerin tutulduğu bellek hücrelerinin adresini okur.
6. Değişkenlere uygun değerler girildikten sonra ENTER tuşuna basılır, imlecekte bir alt satıra geçer.

Kullanımı => `scanf("Tip karakteri",*arguman);`

Klavyeden girilen değerler okunurken (karakterler haric), verilerin önündeki boşluk, tab, yeni satır vb. karakterler önemsenmez, ancak aralarda olursa önemlidir.

`printf()` gibi `scanf()` fonksiyonunda tablolarda verilen tip karakterlerini kullanır. Örneğin klavyeden bir `x` tamsayısı okumak için:

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

satırını yazmak yeterli olacaktır. Burada `&` işareti *adres operatörü* olarak adlandırılır. Klavyeden iki farklı sayı okunmak istendiğinde `scanf()` fonksiyonu şöyle kullanılabilir:

```
scanf("%d %f",&x,&y);
```

veriler klavyeden

```
16 1.56 (enter)
```

yada

```
16 1.56 (enter)
```

veya

```
16 (enter)
```

```
1.56 (enter)
```

şekilinde girilebilir.

Tip Karakteri	Anlamı
%d	iřaretli tamsayı (onluk sistem)
%i	iřaretli tamsayı (onluk, sekizlik veya 16 lik sistem)

Örnek:

```
char kr1, kr[10];
```

<i>fonksiyon</i>	<i>Klavye</i>	<i>etkisi</i>
scanf("%c" , &kr1);	ALI(enter)	kr1='A'
scanf("%s" , kr); printf("%s\n" , kr);	ALI(enter)	kr="ALI"

Örnek :

```
#include <stdio.h>

main()
{
  int a,b,c,d,e,f,g;

  printf( "7 tane tamsayi giriniz : \n" );
  scanf( "%d%i%i%i%o%u%x" , &a, &b, &c, &d, &e, &f, &g );

  printf( "girdiginiz sayilar:\n" );
  printf( "%d %d %d %d %d %d %d\n" , a, b, c, d, e, f, g );

  getchar();

}
```

Girdiler

-70 -70 070 0x70 70 70 70

Örnek :

```
#include <stdio.h>

main()
{
    char x;
    char y[ 9 ];

    printf( "String yaziniz: " );
    scanf( "%c%s", &x, y );

    printf( "Birinci karakter : %c\n", x );
    printf( "diger kisim : %s", y );
    getchar();
}

```

Girdiler
ALI

Örnek :

```
#include <stdio.h>

main()
{
    int t;
    float g;
    printf( "Bir real sayi giriniz: " ); scanf( "%f", &g );
    printf( "Bir tamsayi sayi giriniz: " ); scanf( "%d", &t );
    puts("");

    printf( "\t%.1f * %.1f = %.2f\n", g, g, g*g );
    printf( "\t%d * %d = %.d\n", t, t, t*t );
    getchar();
}

```

Girdiler
1.2 3

getchar() Fonksiyonu

Bu fonksiyon ile standart girişten bir karakter okuyup çağırana gönderir.

Kullanımı :

int getchar(void);

Dönüş değeri : Okunan karakteri geri döndürür.

Basılan tusun ekranda gösterir ve enter tusunu bekler.

```
#include <stdio.h>
```

```
main()  
{  
char ch;  
    ch = getchar();  
    printf( "%c %d\n" ,ch, ch );  
    getchar();  
}
```

Girdiler

a

gets() Fonksiyonu

Klavyeden bir karakter topluluğu okumak için kullanılır. puts() - gets() arasındaki ilişki, printf() - scanf() arasındaki gibidir. Yani,

Kullanımı :

char gets(char *);

scanf("%s",str); ile gets(str); aynı işlevlidir.

Operatörler (1)

Operatörler, değişkenler veya sabitler üzerinde matematiksel ve karşılaştırma işlemlerini yapan sembollerdir. Yani bir operatör bir veya daha fazla nesne (değişken) üzerinde işlem yapan sembollerdir.

Aritmetik Operatörler

Değişken veya sabitler üzerinde temel aritmetik işlemleri gerçekleyen operatörlerdir. Bunlar aşağıdaki Tablo 'da listelenmiştir.

Aritmetik Operatörler

Operatör	Açıklama	Örnek	Anlamı
+	toplama	$x + y$	x ve y nin toplamı
-	çıkarma	$x - y$	x ve y nin farkı
*	çarpma	$x * y$	x ve y nin çarpımı
/	bölme	x / y	x ve y nin oranı
%	artık bölme	$x \% y$	x / y den kalan sayı

Aritmetik operatörlerin değerlendirme sırası :



Değerlendirme yapılırken önce ifade hesaplanır sonra atama yapılır. Yani, atama operatörleri sağdan sola birleşime sahiptir.

$$\begin{array}{c} 10 - 6 + 5 \\ \underbrace{\quad} \\ 4 + 5 \\ \underbrace{\quad} \\ 9 \end{array}$$

$$\begin{array}{c} 8 / 2 * 3 \\ \underbrace{\quad} \\ 4 * 3 \\ \underbrace{\quad} \\ 12 \end{array}$$

$$\begin{array}{c} 2 + 3 * 10 / 2 \\ \underbrace{\quad} \\ 2 + 30 / 2 \\ \underbrace{\quad} \\ 2 + 15 \\ \underbrace{\quad} \\ 17 \end{array}$$

Aritmetik operatörlerin değerlendirme sırası (önceliği) parantez kullanılarak değiştirilir. Parantez kullanıldığında operatör önceliği gözetmeksizin parantez içi önce değerlendirilir. İç içe parantezlerde hesaplanma önceliği en içteki parantezin içindeki ifadenindir.

$$\begin{array}{c} 8 * (3 + 2) \\ \underbrace{\hspace{1.5cm}} \\ 8 * 5 \\ \underbrace{\hspace{1.5cm}} \\ 40 \end{array}$$

$$\begin{array}{c} 2 + (3 * (10 / 2)) \\ \underbrace{\hspace{1.5cm}} \\ 2 + (3 * 5) \\ \underbrace{\hspace{1.5cm}} \\ 2 + 15 \\ \underbrace{\hspace{1.5cm}} \\ 17 \end{array}$$

Örnek : Bölme işlemi ve kalan buldurma

```
#include <stdio.h>
```

```
main()
{
```

```
printf(" Bolme islemleri\n");
```

```
int x,y;
```

```
x=10;
```

```
y=3;
```

```
printf("%d / %d isleminin sonucu = %d\n",x, y, x/y);
```

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

```
getchar();
```

```
}
```

```
Bolme islemleri
10 / 3 isleminin sonucu = 3
Kalan = 1
```

Atama Operatörleri

Bu operatörler bir değişkene, bir sabit veya bir aritmetik ifade atamak (eşitlemek) için kullanılır. *Birleşik atama*: bazı ifadelerde işlem operatörü ile atama operatörü birlikte kullanılarak, ifadeler daha kısa yazılabilir. Eğer ifade

değişken = değişken [operatör] aritmetik ifade;

şeklinde ise, daha kısa bir biçimde

değişken [operatör]= aritmetik ifade;

olarak yazılabilir. Bu operatörler Tabloda listelenmiştir.

Atama Operatörleri

Operatör	Açıklama	Örnek	Anlamı
=	Atama	$x = 7;$	$x = 7;$
+=	ekleyerek atama	$x += 3$	$x = x + 3$
-=	eksilterek atama	$x -= 5$	$x = x - 5$
*=	çarparak atama	$x *= 4$	$x = x * 4$
/=	bölerek atama	$x /= 2$	$x = x / 2$
%=	bölüp, kalanını atama	$x \% = 9$	$x = x \% 9$
++	bir arttırma	$x++$ veya $++x$	$x = x + 1$
--	bir azaltma	$x--$ veya $--x$	$x = x - 1$

Bu tanımlamalara göre, aşağıdaki atamaları inceleyiniz:

```
// bir arttırma işlemleri
```

```
i++;  
++i;  
i += 1;  
i = i + 1;
```

```
/* karmaşık atamalar */
```

```
f *= i; // f = f * i; anlamında  
f *= i+1; // f = f * (i+1); anlamında  
z /= 1 + x; // z = z / (1+x); anlamında
```

Bir arttırma veya eksiltme operatörlerini kullanırken dikkatli olunmalıdır. Çünkü aşağıdaki türden atamalar bazen karışıklığa neden olur.

```
a = 5; // a = 5  
b = a++; // a = 6 ve b = 5  
c = ++a; // a = 7 ve c = 7
```

Örnek : Aritmetik ve atama operatörlerinin kullanımı

```
#include <stdio.h>
```

```
main()
{
```

```
printf(" Bolme islemleri\n");
int x,y;
```

```
x=1;
y=3;
```

```
printf("x=%d ve y=%d olarak verilmistir.\n",x, y);
```

```
x = x+y; //x+=y;
```

```
printf("x+y->x atamasi sonucunda x=%d\n",x);
```

```
x=++y;
```

```
printf("x=++y atamasi sonucunda x=%d ve y=%d\n",x,y);
```

```
x=y++;
```

```
printf("x=y++ atamasi sonucunda x=%d ve y=%d\n",x,y);
```

```
x+=y;
```

```
printf("x+=y atamasi sonucunda x=%d \n",x);
```

```
x*=y;
```

```
printf("x*=y atamasi sonucunda x=%d \n",x);
```

```
getchar();
}
```

Örnek : Bir üçgenin taban ve yükseklik değerlerini okuyup alanını hesaplayan programı yazınız.

```
#include <stdio.h>

main()
{
    float taban, yukseklik, alan;

    puts("Ucgenin;");
    printf("Taban Uzunlugu = "); scanf("%f",&taban);
    printf("Yuksekligi = "); scanf("%f",&yukseklik);

    alan = (taban*yukseklik) / 2;

    printf("Ucgenin Alani = %.3f",alan);

    getch();
}
```

Örnek : Yarıçapı klavyeden girilecek bir dairenin alanını ve çevresini hesaplayan program.

```
#include <stdio.h>
#define PI 3.14159

main()
{
    float r, cevre, alan;

    printf("Yaricap = "); scanf("%f",&r);

    //alan hesabi
    alan = PI * r * r ;

    //cevre hesabi
    cevre = 2 * PI * r ;

    //sonuclari ekrana yazdir
    printf("Daire Alani = %.4f\n",alan);
    printf("Daire Cevresi = %.4f\n",cevre);

    getch();
}
```

Matematiksel Fonksiyonlar

Matematiksel fonksiyonlar double parametreleri kullanır ve gönderdikleri değerler de double' dır. Burada anlatılan fonksiyonların kullanılabilmesi **math.h** başlık dosyasının programa dahil edilmesi gerekir.

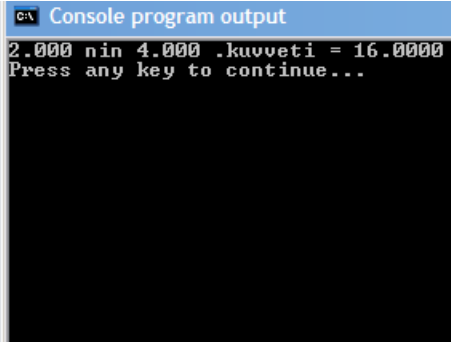
1. sqrt() Fonksiyonu : Kırakök alır.
Kullanımı : **double sqrt(double a);**
2. pow() Fonksiyonu : x^y ifadesinde x' in y 'inci kuvvetini bulur.
Kullanımı : **double pow(double x, double y);**

Örnek : 2 'nin 4. Kuvvetini hesaplayan program

```
#include <stdio.h>
#include <math.h>

main()
{
    double x = 2, y = 4;

    printf("%.3f nin %.3f .kuvveti = %.4f \n", x, y, pow(x, y));
}
```

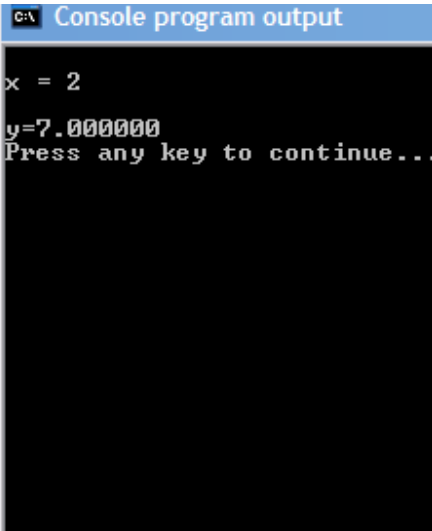


Örnek : $y = x + \sqrt{x + 23}$ ifadesini klavyeden girilen x değeri için hesaplayan program

```
#include <stdio.h>
#include <math.h>

main()
{
    float x, y;

    printf("\nx = ");
    scanf("%f", &x);
    y = pow(x+23, 1/2.0) + x;
    printf("\ny=%f\n", y);
}
```



3. `exp()` Fonksiyonu : e^x ifadesinde e ' in x 'inci kuvvetini bulur.
Kullanımı : ***double exp(double x);***
4. `log()` Fonksiyonu : $\log_e x$ ifadesinde e tabanına göre x ' in doğal logaritmasını (\ln) hesaplar.
 e sayısı 2.71828 dir.
Kullanımı : ***double log(double x);***
5. `log10()` Fonksiyonu : x ' in 10 tabanına göre logaritmasını hesaplar.
Kullanımı : ***double log10(double x);***
6. `sin()` Fonksiyonu : x ' in (radyan) sinusunu hesaplar.
Kullanımı : ***double sin(double x);***
7. `cos()` Fonksiyonu : x ' in (radyan) cosinusunu hesaplar.
Kullanımı : ***double cos(double x);***
8. `tan()` Fonksiyonu : x ' in (radyan) tanjantını hesaplar.
Kullanımı : ***double tan(double x);***

Örnek :

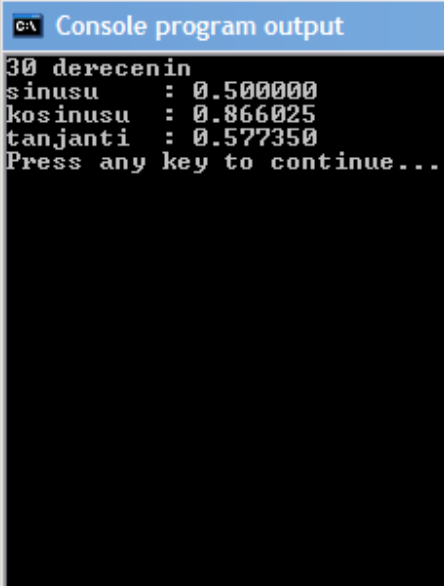
```
/*30 dercelik açının sinüs, kosinüs, tanjant değerleri */
#include <stdio.h>
#include <math.h>

#define PI 3.141593

int main()
{
    double aci = 30.0;

    aci *= PI/180.0; /* radyana çevir */

    puts("30 derecenin");
    printf("sinusu : %f\n", sin(aci));
    printf("kosinusu : %f\n", cos(aci));
    printf("tanjanti : %f\n", tan(aci));
}
```



9. `floor()` Fonksiyonu : x 'den büyük olmayan en yakın tamsayıyı bulur. Örneğin $x=7.3$ ise çıkış 7, $x=-7.3$ ise çıkış -8 olur.
Kullanımı : ***double floor(double x);***

10. `ceil()` Fonksiyonu : x 'den küçük olmayan en yakın tamsayıyı bulur. Örneğin $x=7.3$ ise çıkış 8, $x=-7.3$ ise çıkış -7 olur.

Kullanımı : ***double ceil(double x);***

11. `fabs()` Fonksiyonu : x 'in mutlak değerini bulur.

Kullanımı : ***double fabs(double x);***

12. `abs()` Fonksiyonu : Tamsayı x 'in mutlak değerini bulur.

Kullanımı : ***int abs(int x);***

Şartlı Akış ve Karşılaştırma Yapıları

Program içerisinde bazen iki veya daha fazla değerin karşılaştırılıp program akışının değiştirilmesi gerekebilir. Bunun için, bütün programlama dillerinde karşılaştırma deyimleri mevcuttur. C dili, **if**, **switch** ve **?** olmak üzere üç tip karşılaştırma yapısı bulunur.

Karşılaştırma Operatörleri ve Mantıksal Operatörler

Aşağıdaki Tablo 'da listelenen Karşılaştırma Operatörleri, sayısal değerleri veya karakterleri karşılaştırmak için kullanılır.

Tablo : Karşılaştırma Operatörleri

Operatör	Açıklama	Örnek	Anlamı
>	büyüktür	$x > y$	x, y den büyük mü?
<	küçüktür	$x < y$	x, y den küçük mü?
==	eşittir	$x == y$	x, y ye eşit mi?
>=	büyük-eşittir	$x >= y$	x, y den büyük yada eşit mi?
<=	küçük-eşittir	$x <= y$	x, y den küçük yada eşit mi?
!=	eşit değil	$x != y$	x, y den farklı mı?

Birden çok karşılaştırma işlemi, alttaki Tablo'daki Mantıksal Operatörler'le birleştirilebilir.

Tablo: Mantıksal Operatörler

Operatör	Açıklama	Örnek	Anlamı
&&	mantıksal VE	$x > 2 \ \&\& \ x < y$	x, 2 den büyük VE y den küçük mü?
	mantıksal VEYA	$x > 2 \ \ x < y$	x, 2 den büyük VEYA y den küçük mü?

C dilinde, bir mantıksal işlemin sonucu tamsayı 0 (sıfır) veya başka bir değer olur. 0 *olumsuz*, 0'dan farklı değerler *olumlu* olarak yorumlanır.

```
...
#include <stdio.h>

main()
{
    int x = 1, y = 2, s, u, z;

    s = 2 > 1;
    u = x > 3;
    z = x <= y && y > 0;

    printf("%d %d %d", s, u, z);

    getchar();
}
```

Progmanın çıktısı:

1 0 1

şeklinde olur. Bunun nedeni:

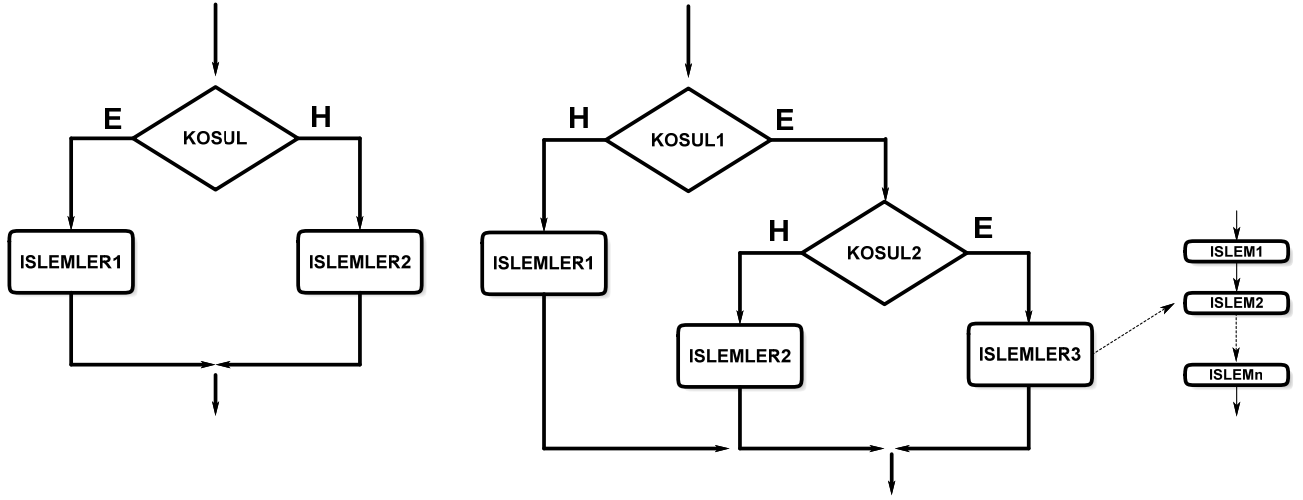
- 2 her zaman 1 den büyük olduğu için s değişkenine 1,
- $x = 1 < 3$ olduğu için x değişkenine 0,

$z = x <= y \ \&\& \ y > 0$; eşitliğin sağ tarafının sonucu olumlu olduğu için z değişkenine 1 atanır.

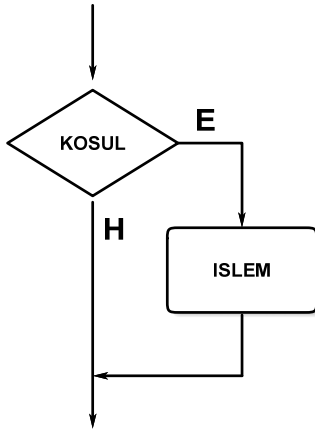
Şartlı Akış

Programın akışını şartlara/koşullara göre değiştirmek için kullanılan komutlardır. Bu yapıda karşılaştırma işlemleri ve birden fazla sıralı yapı kullanılır. Hangi şartlarda hangi sıralı yapının seçileceği karşılaştırma sonucu belirlenir. C programlama dilinde şartlı akış komutları;

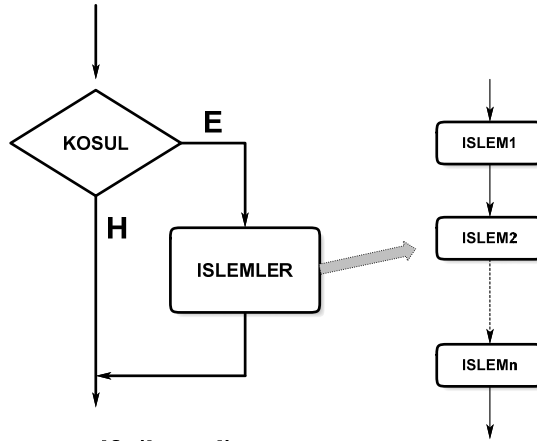
if.... ve *switch....case* 'tir.



if Yapısı



```
if (kosul)  
islem;
```



```
if (kosul)  
{  
. islemler;  
.  
}
```

if deyimi kullanılırken kümenin başlangıcı ve bitişini gösteren, küme parantezleri kullanılmasında kullanıcıya bir esneklik sunulmuştur. Eğer if deyiminden sonra icra edilecek işlemler tek satırdan oluşuyorsa, bu işaretlerin kullanılması zorunlu değildir. Yani, if deyimden sonra { ve } işaretleri kullanılmamışsa, bu işlemi takip eden sadece ilk satır çalışır. Bu durum, else if, else deyimlerinde ve daha sonra işlenecek for ve while gibi döngü deyimlerinde de geçerlidir.

Buna göre aşağıdaki kullanım

```
if(x == y)
{
    puts("x ve y esit");
}
```

ile

```
if(x == y)
    puts("x ve y esit");
```

eşdeğerdir.

Örnek : Klavyeden girilen bir tam sayının pozitif, negatif veya sıfır olduğunu bulan programı yazınız

```
#include <stdio.h>

main()
{
    int a;

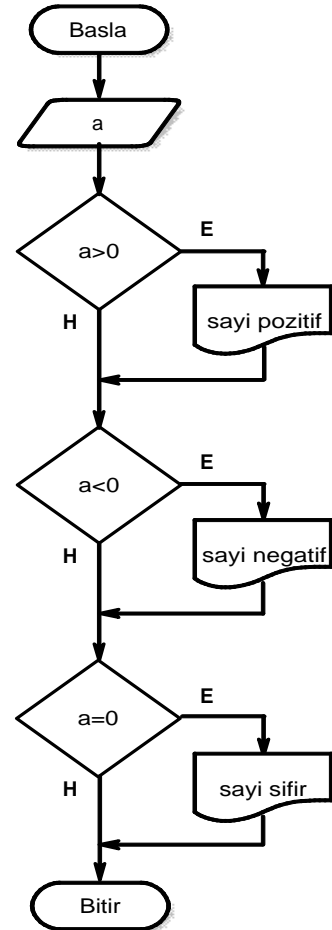
    printf("Bir tamsayi giriniz:");
    scanf("%d",&a);

    if (a>0)
        printf("Girdiginiz %d sayisi pozitif",a);

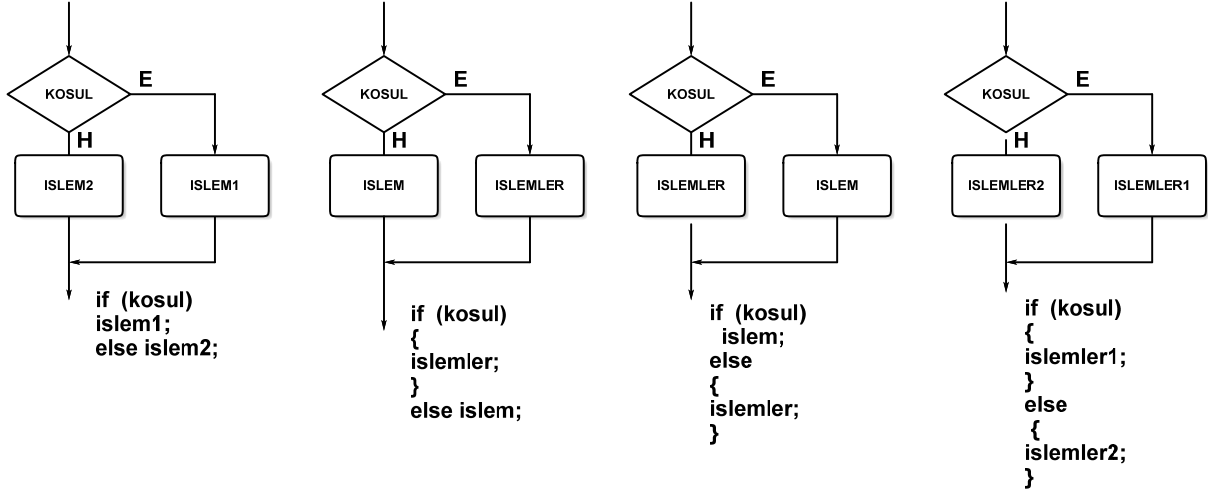
    if (a<0)
        printf("Girdiginiz %d sayisi negatif",a);

    if (a==0)
        printf("Girdiginiz %d sayisi sifir",a);

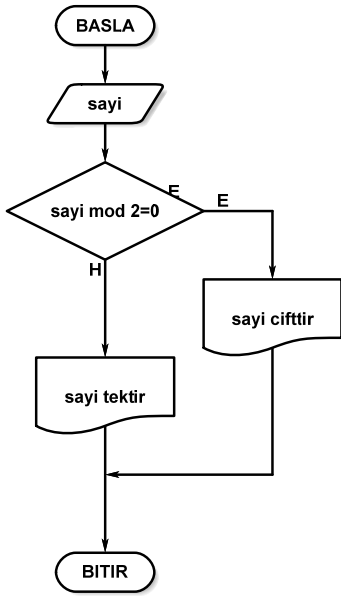
    getchar();
}
```



if else Yapısı



Örnek : Klavyeden girilen bir tamsayının çift olup olmadığını sınırlar. Bilindiği gibi, çift sayılar, 2 ile kalansız bölünebilen sayılardır.



```
#include <stdio.h>

main()
{
    int sayi;

    printf("Bir sayi girin: ");
    scanf("%d",&sayi);

    if (sayi % 2 == 0)
        printf("sayi çifttir.\n");
    else
        printf("sayi tektir.\n");

    getch();
}
```

Mantıksal Operatörler kullanarak birden çok karşılaştırma birleştirilebilir.

&& mantıksal VE
|| mantıksal VEYA

Örnek : 3 kenarının uzunluğu girilecek bir üçgenin eşkenar olup-olmadığını bulan program.

```
#include <stdio.h>
main()
{
    int a, b, c;

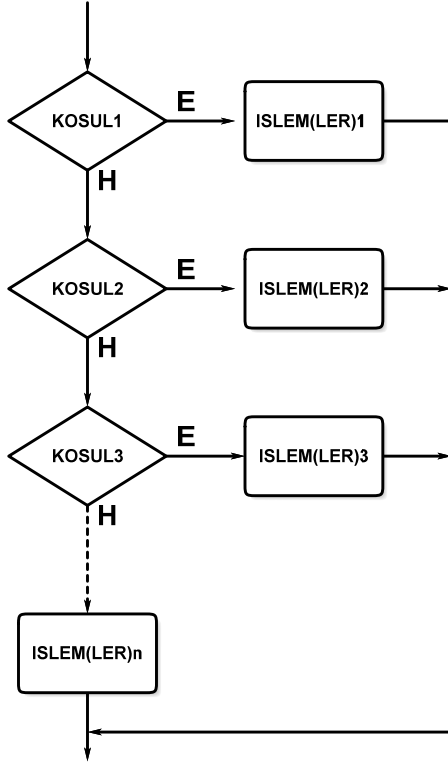
    printf("Ucgenin kenar uzunluklarini giriniz=");
    scanf("%d %d %d", &a , &b , &c);

    if( a == b && a==c )
        printf("Ucgen eskenardir");
    else
        printf("Ucgen eskenar degildir");

    getchar();
}
```

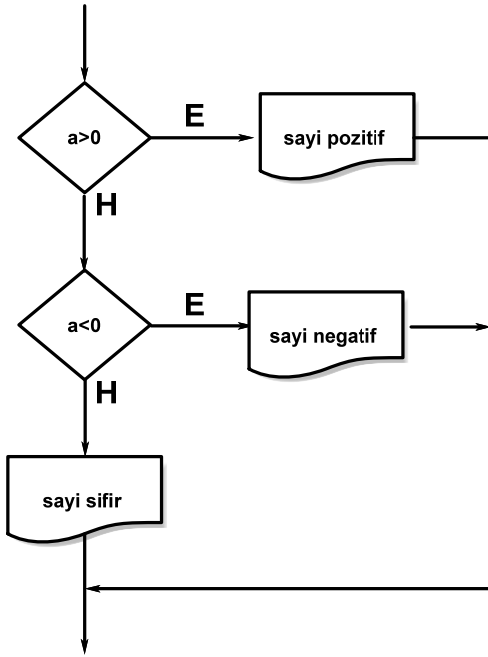
if else if else Yapısı

Eğer program içinde kullanılacak koşulların sayısı ikiden çok ise aşağıdaki yapı kullanılır:



```
if (kosul1)
{
islem(ler)1;
}
else if (kosul2)
{
islem(ler)2;
}
.
.
.
else
{
islem(ler)n;
}
```

Örnek : Klavyeden girilen bir tam sayının pozitif, negatif veya sıfır olduğunu bulan programı yazınız.



```
#include <stdio.h>

main()
{
    int a;

    printf("Bir sayi giriniz=");
    scanf("%d", &a);

    if( a > 0 )
        printf("Sayi pozitif");

    else if( a < 0 )
        printf("Sayi negatif");

    else printf("Sayi sıfır");

    getch();
}
```

Örnek : Klavyeden girilen bir tuşun hangi grupta yer aldığını olduğunu bulan programı yazınız.

```
#include<stdio.h>
main()
{
    char ch;
    printf("Bir karakter giriniz : ");
    scanf("%c", &ch);
    if(ch>='0' && ch<='9')
        printf("Rakam Girdiniz ...\n");
    else if(ch>='a' && ch<='z')
        printf("Kucuk harf girdiniz ...\n");
    else if(ch>='A' && ch<='Z')
        printf("Buyuk harf girdiniz ...\n");
    else
        printf("Ozel karakter girdiniz ...\n");

    getch();
}
```

Örnek: $ax^2 + bx + c = 0$ formundaki ikinci dereceden bir polinomun köklerini hesaplamaktadır. Programda delta değerinin sıfırdan küçük olması durumunda köklerin karmaşık sayıya dönüşeceği göz önüne alınmıştır. .

```
#include <stdio.h>
#include <math.h>

main()
{
    float a, b, c, delta, x1, x2, x, kok_delta;

    printf("a, b, c degerlerini girin:\n");
    scanf("%f %f %f",&a,&b,&c);

    delta = b*b - 4.0*a*c;

    if( delta > 0.0 ){
        x1 = ( -b + sqrt(delta) )/( 2.0*a );
        x2 = ( -b - sqrt(delta) )/( 2.0*a );

        printf("\nReel kokler:");
        printf("\nx1 = %f",x1);
        printf("\nx2 = %f\n",x2);
    }
    else if( delta < 0.0 ){
        kok_delta = ( sqrt(-delta) ) / (2.0*a);
        x = -0.5*b/a;

        printf("\nKarmasik kokler:");
        printf("\nx1 = %f + (%f)i", x, kok_delta);
        printf("\nx2 = %f - (%f)i\n", x, kok_delta);
    }
    else{
        x = -0.5*b/a;

        printf("\nKokler esit:");
        printf("\nx1 = x2 = %f\n",x);
    }

    getchar();
}
```

```
a, b, c degerlerini girin:
2 4 -8

Reel kokler:
x1 = 1.236068
x2 = -3.236068
Press any key to continue...
```

```
a, b, c degerlerini girin:
1 1 1

Karmasik kokler:
x1 = -0.500000 + <0.866025>i
x2 = -0.500000 - <0.866025>i
Press any key to continue...
```

switch - case Yapısı

Bir *değişkenin* içeriğine bakarak, programın akışını bir çok seçenektan birine yönlendirir. case (durum) deyiminden sonra *değişkenin* durumu belirlenir ve takip eden gelen satırlar (işlemler) çalışır. Bütün durumların aksi söz konu olduğunda gerçekleştirilmesi istenen deyimler default deyiminden sonraki kısımda bildirilir. Genel yazım biçimi:

```
switch (değişken)
{
    case sabit1:
        ...
        //Islemler1;
        ...
    case sabit2:
        ...
        //Islemler2;
        ...
    .
    .
    .
    case sabitn:
        ...
        //Islemler n;
        ...
    default:
        ...
        //hata deyimleri veya varsayılan deyimler;
        ...
}
```

Örnek :

```
#include <stdio.h>
main(void)
{
    char kr;

    printf("Lutfen bir karakter girin\n");

    kr = getchar(); /* tek bir karakterin okunması */

    switch (kr)
    {
        case 'a':
            printf("a harfine bastiniz\n");
        case 'b':
            printf("b harfine bastiniz\n");
        default:
            printf("a veya b ye basmadiniz\n");
    }

    getchar();
}
```

Lutfen bir karakter girin
a
a harfine bastiniz
b harfine bastiniz
a veya b ye basmadiniz

Lutfen bir karakter girin
b
b harfine bastiniz
a veya b ye basmadiniz

Lutfen bir karakter girin
k
a veya b ye basmadiniz

Lutfen bir karakter girin
c
a veya b ye basmadiniz

break Deyimi

Bir döngü içerisinde break deyimi ile karşılaşıldığında döngü koşula bakılmaksızın sonlanır ve akış döngüden sonraki işleme atlar.

```
#include <stdio.h>

main(void)
{
    char kr;

    printf("Lutfen bir karakter girin\n");

    kr = getchar(); /* tek bir karakterin okunması */

    switch (kr)
    {
        case 'a':
            printf("a harfine bastiniz\n");
            break;
        case 'b':
            printf("b harfine bastiniz\n");
            break;
        default:
            printf("a veya b ye basmadiniz\n");
            break;
    }

    getchar();

}
```

ÇIKTI

```
Lutfen bir karakter girin
a
a harfine bastiniz
```

ÇIKTI

```
Lutfen bir karakter girin
k
a veya b ye basmadiniz
```

Örnek: Klavyeden girilecek 2 tamsayıyı, isteğe bağlı olarak toplayan, çıkaran, çarpan, bölen veya modunu bulan program.

```
#include<stdio.h>
main()
{
    char ch;
    int x, y, sonuc;
    float bolme;
    printf("Islem yapılacak iki adet tamsayi giriniz : \n");
    scanf("%d %d", &x, &y);
    printf("Islemi seciniz (+, -, *, /, %) : ");
    scanf("%c", &ch);
    switch(ch)
    {
        case '+':
            sonuc = x + y;
            printf("%d + %d = %d\n", x, y, sonuc);
            break;

        case '-':
            sonuc = x - y;
            printf("%d - %d = %d\n", x, y, sonuc);
            break;

        case '*':
            sonuc = x * y;
            printf("%d * %d = %d\n", x, y, sonuc);
            break;

        case '/':
            bolme = (float) x / y;
            printf("%d / %d = %f\n", x, y, bolme);
            break;

        case '%':
            sonuc = x % y;
            printf("%d mod %d = %d\n", x, y, sonuc);
            break;

        default :
            printf("Yanlis islem sectiniz, tekrar ediniz ... \n");
    }
    getch();
}
```

Örnek: Klavyeden basılacak harfin sesli veya sessiz olduğunu bulan program.

```
#include<stdio.h>
main()
{
    char ch;
    printf("Bir karakter giriniz : ");
    scanf("%c", &ch);
    switch(ch)
    {
        case 'a' :
        case 'e' :
        case 'i' :
        case 'o' :
        case 'u' :
            printf("Girilen karakter sesli harftir ...\n");
            break;
        default :
            printf("Girilen karakter sesli harf degildir ...\n");
    }

    getch();
}
```

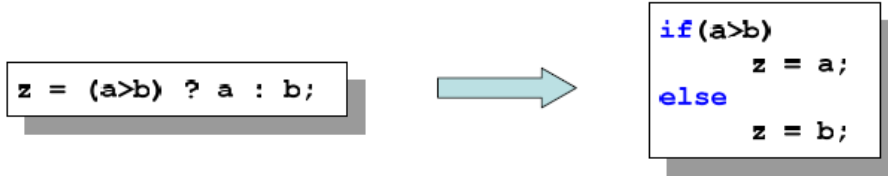
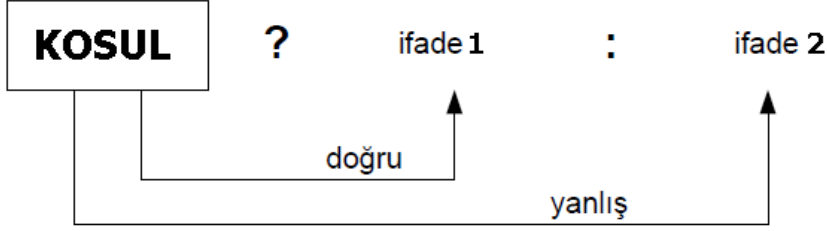
switch-case yapısı *if else if else* yapısının bir alternatifidir.

<pre>switch(secim) { case 1: sonuc = x + y; printf("Toplam = %f\n",sonuc); break; case 2: sonuc = x-y; printf("Fark = %f\n",sonuc); break; case 3: sonuc = x * y; printf("Carpim = %f\n",sonuc); break; case 4: sonuc = x/y; printf("Oran = %f\n",sonuc); break; default: puts("Yanlis secim !\a"); }</pre>	<pre>if(secim == 1) { sonuc = x + y; printf("Toplam = %f\n",sonuc); } else if(secim == 2) { sonuc = x-y; printf("Fark = %f\n",sonuc); } else if(secim == 3) { sonuc = x * y; printf("Carpim = %f\n",sonuc); } else if(secim == 4) { sonuc = x/y; printf("Oran = %f\n",sonuc); } else{ puts("Yanlis secim !\a"); }</pre>
---	--

? Karşılaştırma Operatörü

Bu operatör, *if-else* karşılaştırma deyiminin yaptığı işi sınırlı olarak yapan bir operatördür. Genel yazım biçimi:

(koşul) ? deyim1 : deyim2;



İlk önce koşul sınanır. Eğer koşul olumluysa *ifade1* aksi takdirde *ifade2* değerlendirilir. *ifade1* ve *ifade2* de atama işlemi yapılamaz. Ancak koşul deyiminde atama işlemi yapılabilir. *ifade1* ve *ifade2* yerine fonksiyon da kullanılabilir.

Yukarıdaki ifadede koşul a'nın b'den büyük olmasıdır. Eğer olumluysa z adlı değişkene a, değilse b değeri atanır.

NOT : `if(y)` kullanımı, koşulun 0 dan farklı olup olmadığı sınar.

`if(y) = if(y != 0)`

Örnek :

```
#include <stdio.h>
```

```
main()
```

```
{
```

```
    float x, y, z;
```

```
    printf("x : "); scanf("%f",&x);
```

```
    printf("y : "); scanf("%f",&y);
```

```
        if(y)
```

```
            z = (y > x) ? x/y : x*y;
```

```
        else
```

```
            z = 0.0;
```

```
    printf("z = %f\n",z);
```

```
    getchar();
```

```
}
```

ÇIKTI

```
x : 3  
y : 5  
z = 0.600000
```

ÇIKTI

```
x : 11  
y : 0  
z = 0.000000
```

DÖNGÜLER

İşlem yada işlemlerin bir koşula bağlı olarak yada istenilen sayıda tekrarlanmasını sağlayan yapılardır. Programlama dillerinde birden fazla çalışan işlem yada işlemleri sağlayan yapılara DÖNGÜ (loop) denir.

Programlama dillerinde genellikle 2-tip döngü yapısı bulunur;

1. Sayı-kontrollü döngü : Aşağıdakileri içermelidir;
 - a. Döngü değişkeni,
 - b. Döngü değişkeninin başlangıç ve bitiş değeri,
 - c. Döngü değişkeninin artma veya azalma değeri,
2. Koşul-kontrollü döngü : Aşağıdakiler içermelidir;
 - a. Döngü koşula göre tekrarlanır (döngünün kaç kez tekrarlanacağı bilinmez).

Standart C Programlama dilinde ,

For

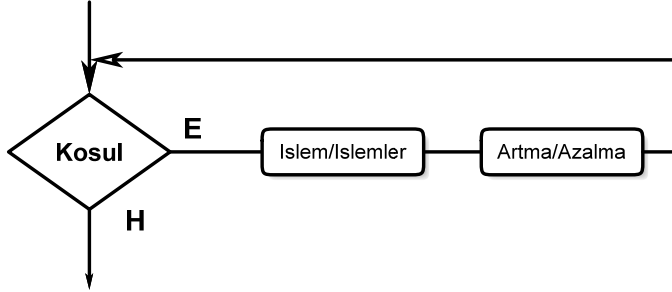
While

do .. while

Olmak üzere 3 farklı döngü yapısı bulunur. Diğer programlama dillerinde olduğu gibi, bu deyimlerle istenildiği kadar iç-içe döngü yapısı kullanılabilir.

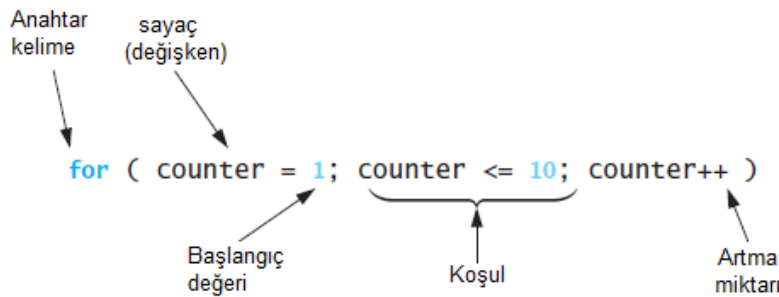
for Döngüsü

Belirlenen işlem yada işlemleri istenilen sayıda tekrarlamak için kullanılır. Bu yapının kullanılabilmesi için döngünün kaç kez tekrar edileceği mutlaka bilinmelidir. Döngü içindeki işlem/işlemler, döngü değişkeni (sayaç) ile başlangıç ve bitiş değerleri arasında, istenildiği miktarda artarak veya azalarak tekrarlanır.



İleri sayan döngü;

```
for( Döngü değişkeni = başlangıç değeri ; koşul ; artma değeri )  
{  
  ...  
  İşlemler;  
  ...  
}
```



Yukarıdaki örnekte döngü, artma miktarı 1 olduğu için; Counter değişkeninin 1,2,3,4,5,6,7,8,9,10 değerleri için tekrarlanır.

Geri sayan döngü;

```
for( Döngü değişkeni=başlangıç değeri ; koşul ; azalma değeri )  
{  
  ...  
  İşlemler;  
  ...  
}
```

```
for ( i = 10; i >= 1; i-- )
```

Yukarıdaki örnekte döngü, azalma miktarı 1 olduğu için; i değişkeninin 10,9,8,7,6,5,4,3,2,1 değerleri için tekrarlanır.

Eğer döngü içinde sadece bir satır işlem görecekseniz küme işaretleri kullanılmayabilir.

Örnekler :

```
#include <stdio.h>
```

```
main()
{
    int i;

    for (i=1 ; i<5 ; i++)
        printf("C dersi %d\n",i);
        printf("%d\n",i);

    getchar();
}
```

```
#include <stdio.h>
```

```
main()
{
    int i;
    for (i=1 ; i<5 ; i++)
    {
        printf("C dersi %d\n",i);
        printf("%d\n",i);
    }

    getchar();
}
```

```
#include <stdio.h>
```

```
main()
{
    int i;

    for (i=1 ; i<5 ; i+=2)
        printf("C dersı %d\n",i);
        printf("%d\n",i);

    getchar();
}
```

```
#include <stdio.h>
```

```
main()
{
    int i;

    for (i=5 ; i>1 ; i--)
        printf("C dersı %d\n",i);
        printf("%d\n",i);

    getchar();
}
```

```
#include <stdio.h>
```

```
main()
{
    float i;

    for (i=0 ; i<1 ; i+=0.1)
        printf("%.1f\n",i);

    getchar();
}
```

```

#include <stdio.h>

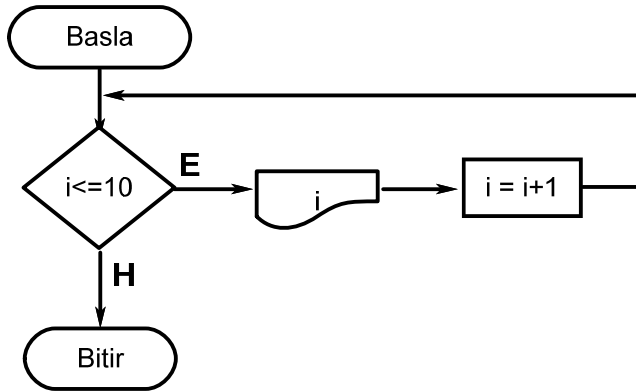
main()
{
    int i;
    char k;

    for (k='a' ; k<'e' ; k+=1)
        printf("%c\n",k);

    getch();
}

```

Örnek : 1..10 'e kadar olan sayıları yanyana bir boşlukla ekrana yazdıran program.



```

#include <stdio.h>

main()
{
    int i;
    for (i=1 ; i<=10 ; i++)
        printf("%d ",i);
    getch();
}

```

Örnek : Klavyeden girilecek N tane tamsayının ortalamasını bulup ekrana yazan program.

```
#include <stdio.h>

main()
{
    int i, n, toplam, sayac, sayi;
    float ort;

    printf("Kac sayinin ortalamasi alinacak :");
    scanf("%d", &n);

    toplam = 0;

    for (sayac=1 ; sayac<=n ; sayac++)
    {
        printf("%d. sayiyi giriniz =",sayac); scanf("%d", &sayi);
        toplam = toplam + sayi ;
    }
    ort = (float) toplam /n;
    printf("Girdiginiz %d adet sayinin ortalamasi = %.3f",n, ort);

    getch();
}
```

```
Kac sayinin ortalamasi alinacak :3
1. sayiyi giriniz =1
2. sayiyi giriniz =2
3. sayiyi giriniz =5
Girdiginiz 3 adet sayinin ortalamasi = 2.667
```

Örnek : Faktöriyel hesabı.

```
#include <stdio.h>

main()
{
    int i, n;
    unsigned long faktor;

    printf("Faktöriyeli hesaplanacak sayi :");
    scanf("%d", &n);

    faktor = 1;

    for (i=1 ; i<=n ; i++)
        faktor *= i; //n! = 1 x 2 x 3 x .. x n

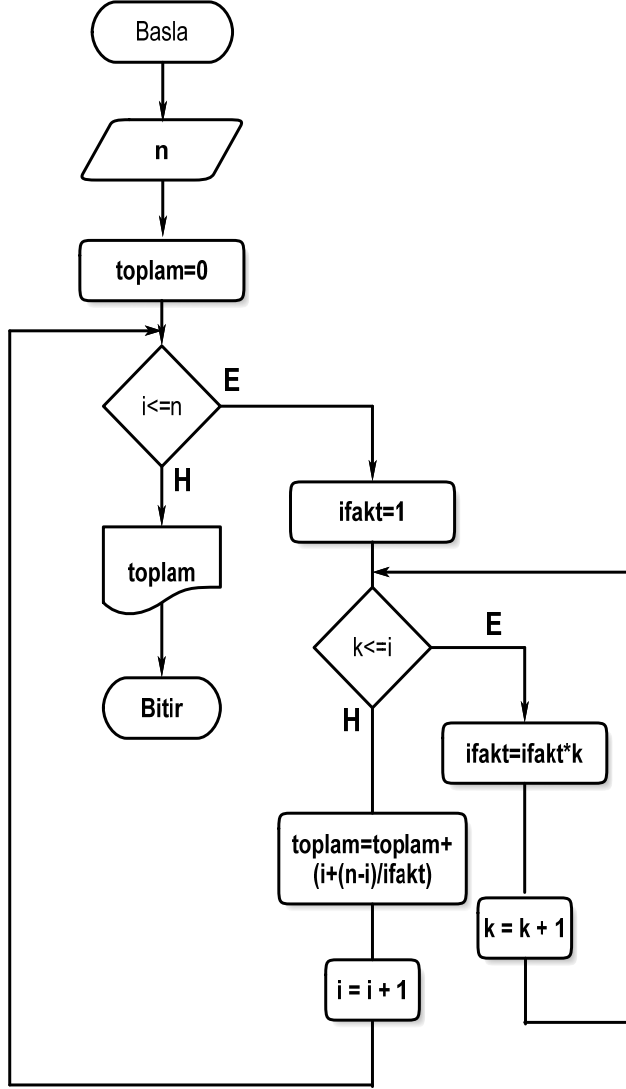
    printf("%d! = %lu\n",n, faktor);

    getch();
}
```

```
Faktöriyeli hesaplanacak sayi :10
10! = 3628800
```

Örnek : Klavyeden girilecek n tam sayısı için aşağıdaki işlemin sonucunu bulan program.

$$\sum_{i=1}^n \left(i + \frac{n-i}{i!} \right) = \left(1 + \frac{n-1}{1!} \right) + \left(2 + \frac{n-2}{2!} \right) + \left(3 + \frac{n-3}{3!} \right) + \dots$$



```

#include <stdio.h>
main()
{
    int i, n, k;
    unsigned long ifakt;
    float toplam;

    printf ("n = "); scanf("%d" , &n);
    toplam = 0.0;

    for (i = 1 ; i <= n ; i++)
    {
        ifakt = 1;
        for (k = 1 ; k <= i ; k++)
            ifakt *= k;
        toplam = toplam + (i+(n-
i)/ifakt);
    }

    printf ("Sonuc = %.2f" , toplam);
    getchar();
}
  
```

for Döngüsünün Farklı Kullanım Şekilleri

1. **koşul** verilmemişse sonsuz döngü oluşur.

```
for (;)
printf ("Sonsuz Dongu);
```

2. Başlangıç değeri döngüye girmeden verilebilir. Döngü değişkeninin başlangıç değeri verilmesse sayaç rastgele değer alır.

```
x = 30;
for (; x<50 ; x+=2)
printf ("%d" , x);
```

3. Döngü değişkeninin artma ve azalma değeri döngü içinde verilebilir.

```
for (x=1; x<100;)
printf ("%f" , sqrt(x++));
```

4. Döngü sayacı olarak birden çok değişken kullanılabilir ve bunlar koşula bağlanabilir. Artma ve azalma değerleri de birden çok olabilir.

```
#include <stdio.h>
main()
{
    int a, b, c;
    for (a=0, b=10, c=100 ; a<10 && b<25 && c>2 ; a++, b+=2, c=c-3)
    printf ("%d %d %d\n", a, b, c);
    getchar();
}
```

İç içe Geçmiş Döngüler

Bir program içinde birbiri içine geçmiş birden çok döngü de kullanılabilir. Bu durumda (bütün programlama dillerinde olduğu gibi) önce içteki döngü, daha sonra dıştaki döngü icra edilir.

```
#include <stdio.h>

main()
{
    int i, j;

    for (i=1 ; i<=5 ; i++)
    {
        for (j=1 ; j<=5 ; j++)
            printf("*");
        printf("\n");
    }
    getchar();
}
```

Üç basamaklı, basamaklarının küpleri toplamı kendisine eşit olan tam sayılara Armstrong sayı denir. Örneğin: 371 bir Armstrong sayıdır çünkü $3^3 + 7^3 + 1^3 = 371$.

```
//Üç basamaklı, basamaklarının küpleri toplamı kendisine eşit olan tam
//sayılara Armstrong sayı denir. Örneğin: 371 = 3^3 + 7^3 + 1^3.
//Bu program İç-içe geçmiş 3 döngü ile bütün Armstrong sayılarını bulur.
```

```
#include <stdio.h>

main()
{
    int a,b,c, kup, sayi, k=1;

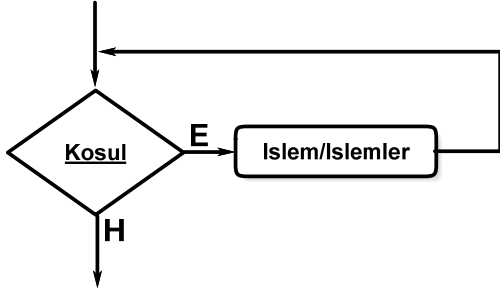
    for(a=1; a<=9; a++)
    for(b=0; b<=9; b++)
    for(c=0; c<=9; c++)
    {
        sayi = 100*a + 10*b + c; // sayi = abc (üç basamaklı)
        kup = a*a*a + b*b*b + c*c*c; // kup = a^3+b^3+c^3
        if( sayi==kup ) printf("%d. %d\n",k++,sayi);
    }

    getchar();
}
```

```
1. 153
2. 370
3. 371
4. 407
```

while Döngüsü:

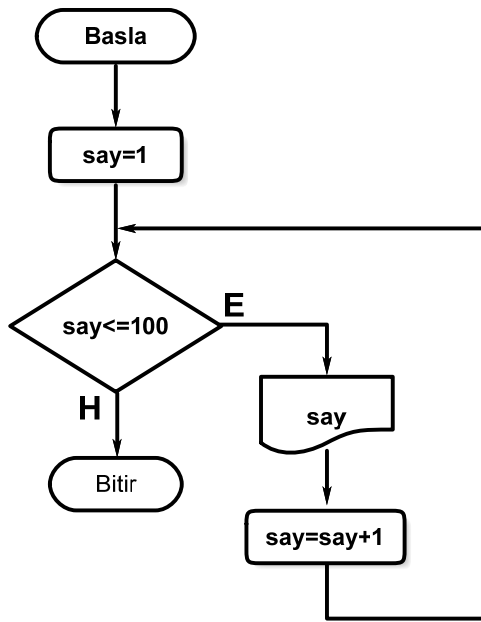
Koşul doğru olduğu sürece istenilen işlem/işlemlerin tekrarlanmasını sağlayan yapıdır.



```
while (kosul)
    islem;
```

```
while (kosul)
{
    islemler;
}
```

Örnek : 1..100 'e kadar olan sayıları yanyana bir boşlukla ekrana yazdıran program.



```
#include<stdio.h>
main()
{
    int say = 1;
    while( say<= 100 )
    {
        printf("%d ",say);
        say++;
    }
    getch();
}
```

Örnek : Ekranı 10 kez "C programlama dili" yazan program.

```
//Ekranı 10 kere "C programlama dili" yazan program

#include<stdio.h>
main()
{
    //i deęişkenine bir başlangıç deęeri atıyoruz.
    //i'ye ilk deęer atanmazsa, döngü yanlış çalışır.
    int i = 0;
    //i'nin deęeri kontrol işleminden
    //sonra 1 artar.
    while( i++ < 10 )

        printf("%d: C programlama dili\n",i);
        printf("%d\n",i);

    getch();
}
```

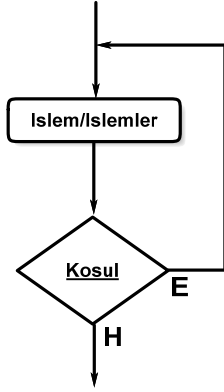
Örnek : Yandaki işlem sonucunu klavyeden girilecek n tam sayısı için bulup sonucu ekrana yazdıran program

$$\sum_{i=0}^n i^2$$

```
#include<stdio.h>
main()
{
    int i = 0;
    int toplam_deger = 0;
    int n;
    printf("n deęerini giriniz : ");
    scanf("%d",&n);
    while( i <= n )
    {
        toplam_deger += i*i;
        i++;
    }
    printf("Sonuç: %d\n",toplam_deger);
    getch();
}
```

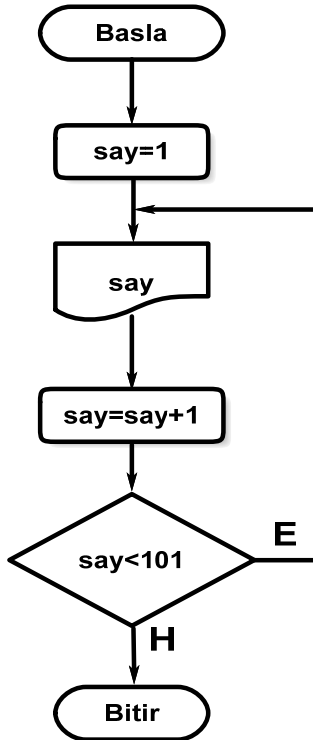
do .. while Döngüsü:

Koşul bozuluncaya kadar istenilen işlem/işlemlerin tekrarlanmasını sağlayan yapıdır. **while** döngüsünde koşul döngüye girmeden, **do .. while** döngüsünde ise koşul döngü sonunda test edilir.



```
do {  
  islem/islemler;  
} while (kosul);
```

Örnek : 1..100 'e kadar olan sayıları yanyana bir boşlukla ekrana yazdıran program.



```
#include<stdio.h>  
main()  
{  
  int say = 1;  
  
  do  
  {  
    printf("%d ",say);  
    say++;  
  } while( say < 101 );  
  
  getch();  
}
```

Örnek : Klavyeden girilecek 2 tam sayıyı toplayan, klavyedeki E veya e tuşuna basıldıkça bu işlemi tekrarlayan, başka bir tuşa basınca programı sonlandıran kod.

Dec	Hex	Char	Dec	Hex	Char
64	40	Ø	96	60	`
65	41	A	97	61	a
66	42	B	98	62	b
67	43	C	99	63	c
68	44	D	100	64	d
69	45	E	101	65	e

```
#include<stdio.h>
main()
{
    int sayi1, sayi2;
    char devam_mi;
    do
    {
        printf("Birinci sayi : ");
        scanf("%d",&sayi1);
        printf("ikinci sayi : ");
        scanf("%d",&sayi2);
        printf("%d + %d = %d\n", sayi1, sayi2, sayi1 + sayi2);
        printf("Devam etmek ister misiniz? ");
        //C'de tek karakter okuma işlemi genellikle do..while
        //dongusu icinde kullanılır.
        do
        {
            scanf("%c",&devam_mi);
        }while( devam_mi == '\n' );
        printf("\n");
    } while( devam_mi == 69 || devam_mi == 101 );
    //while( devam_mi == 'E' || devam_mi == 'e' );
    getchar();
}
```

Alt programlar (Fonksiyonlar)

C Programlama Dili fonksiyon olarak adlandırılan alt programların birleştirilmesi kavramına dayanır. Fonksiyonlar, Java veya C# gibi dillerde metod (method) ismini alırlar. Adı ne olursa olsun, görevi aynıdır. Bir işlemi birden çok yaptığımızı düşünün. Her seferinde aynı işlemi yapan kodu yazmak oldukça zahmetli olurdu. Fonksiyonlar, bu soruna yönelik yaratılmıştır. Sadece bir kereye mahsus yapılacak işlem tanımlanır. Ardından dilediğiniz kadar, bu fonksiyonu çağırırsınız. Ayrıca, Fonksiyonlar modülerlik sağlar. Yazılan fonksiyonlara ait kodu, başka programlara taşımanız oldukça basittir.

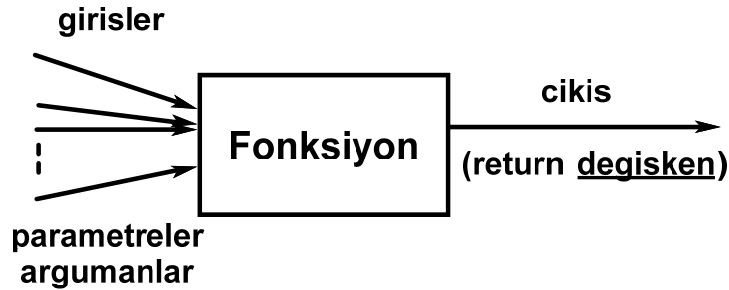
Fonksiyonlar, çalışmayı kolaylaştırır. Diskten veri okuyup, işleyen; ardından kullanıcıya gösterilmek üzere sonuçları grafik hâline dönüştüren; ve işlem sonucunu diske yazan bir programı baştan aşağı yazarsanız, okuması çok güç olur. Yorum koyarak kodun anlaşılabilirliğini, artırabilirsiniz. Ancak yine de yeterli değildir. İzlenecek en iyi yöntem, programı fonksiyon parçalarına bölmektir. Örneğin, diskten okuma işlemi *disten_oku()* isimli bir fonksiyon yaparken; grafik çizdirme işini *grafik_ciz()* fonksiyonu ve diske yazdırma görevini de *diske_yaz()* fonksiyonu yapabilir. Binlerce satır içinde çalışmaktansa, parçalara ayrılmış bir yapı daha mantıklıdır.

Programlama dillerinde alt programların kullanılma amaçları;

1. Programlar kısadır: Tekrarlanan program parçaları alt program ile tanımlanarak sadece bir kez yazılır.
2. Programlar takip etmek kolaylaşır: Benzer işi yapan komutlar bir alt program içinde tanımlandığında programı takip etmek kolaylaşır.
3. Yazılan programlarda hata yapma olasılığı azalır.

Fonksiyon Kavramı

Fonksiyon, belirli sayıda verileri kullanarak bunları işleyen ve bir sonuç üreten komut grubudur. Her fonksiyonun bir adı ve fonksiyona gelen değerleri gösteren parametreleri/argumanları (değişkenleri) vardır.



Fonksiyonların girdilerine parametreler yada argumanlar denir. Bir fonksiyon bu parametreleri alıp bir işleme tabi tutar ve bir değer hesaplar. Bu değer, *çıkıtı* veya *geri dönüş değeri* (return değışken) olarak adlandırılır. Bir fonksiyonun kaç girişı olursa olsun sadece bir çıkışı vardır.

C Programlama Dili, kullanıcıasına bu türden fonksiyon yazmasına izin verir. C dilinde hazırlanan bir fonksiyonun genel yapısı şöyledir:

Fonksiyon Tipi Fonksiyon Adı (parametre tipleri ve isimleri)

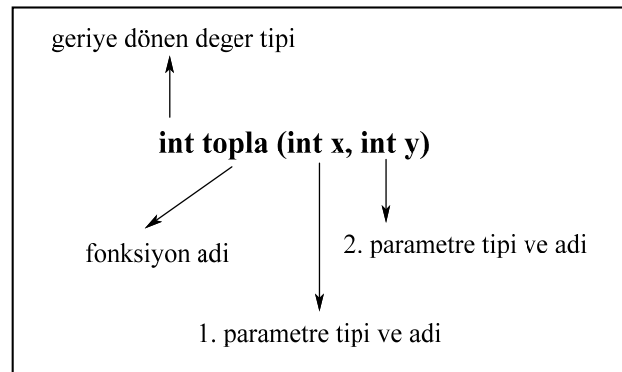
```
{  
  Yerel deęişkenler  
  ...  
  İşlemler  
  ...  
  return geri_dönüş_deęeri;  
}
```

Örneğin iki sayının toplamını hesaplayacak bir fonksiyon şöyle tanımlanabilir:

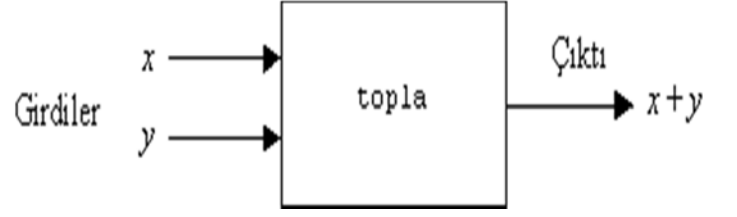
```
int topla(int x,int y)  
{  
  int sonuc;  
  sonuc = x + y;  
  return sonuc;  
}
```

veya

```
int topla(int x,int y)  
{  
  return (x+y);  
}
```



- Fonksiyon tipi: int
- Fonksiyon adı : topla
- parametreler : x ve y
- geri dönüş değeri: x+y



return (geri dönüş) deyimi C programlama dilinin anahtar sözcüklerinden biridir ve fonksiyon içerisinde sonucu, kendisini çağıran yere göndemek için kullanılır. Yani topla fonksiyonu herhangi bir programın içerisinde kullanıldığında, fonksiyonun üreteceği sonuç return deyiminden sonra belirtilen değişken veya işlem olacaktır. Örneğin fonksiyon:

```
...
int t;
...
t = topla(9,6);
...
```

şeklinde kullanılırsa, t değişkenine 9+6=15 değeri atanır.

Fonksiyon Bildirim Yerleri

Bir fonksiyonun bildirimi iki türlü yapılır:

1. Ana programın üstünde

```
int topla(int x,int y) // fonksiyon
{
...
}
.
.
.
main()
{
...
}
```

2. Ana programın altında

Bu durumda fonksiyon prototipi (function prototype) ana programdan önce bildirilmelidir.

```
int topla(int x, int y); // fonksiyon prototipi
.
.
.
main()
{
...
}
.
.
.
int topla(int x, int y) // fonksiyon
{
...
}
```

Bir C programı içinde, yazılmış oldan fonksiyonlar genellikle bu iki tipte kullanılır. İkinci kullanımda fonksiyon prototipi mutlaka bildirilmelidir. Fonksiyon prototipi, fonksiyonun tipi, adı ve parametreleri hakkında bilgi verir. Eğer bir fonksiyon ilk çağrıldığı satırdan önce yazılmışsa, bildirilmesine gerek yoktur. Ancak derleme sırasında henüz tanımlanmamış bir fonksiyon görülürse, tipinin *int* olduğu kabul edilir. Derleme sırasında fonksiyona gelindiğinde tipinin *int* olmadığı görülürse hata oluşur.

Fonksiyon prototipinde parametre isimlerinin yazılması zorunlu değildir. Sadece parametre tiplerini belirtmek de yeterlidir. Yukarıdaki topla fonksiyona ait prototip:

int topla(int x, int y);

şekinde yazılabileği gibi

int topla(int, int);

şeklinde de yazılabilir.

topla fonksiyonunun ana program altında tanımlanması

```
#include <stdio.h>

int topla(int x, int y); // fonksiyon prototipi

main()
{
    int toplam,a,b;

    printf("İki sayı girin : ");
    scanf("%d %d",&a,&b);

    toplam = topla(a,b);

    printf("%d ve %d nin toplamı %d dir.\n", a,b,toplam);

    getch();
}

// fonksiyon tanımlanması
int topla( int x, int y )
{
    int sonuc;
    sonuc = x + y;
    return sonuc;
}
```

ÇIKTI

```
İki sayı girin : 5 12
5 ve 12 nin toplamı 17 dir.
```

Programda, klavyeden okunan a ve b değişkenleri fonksiyonuna parametre olarak aktarılmıştır. Bu değişkenlerin isimleri ile topla fonksiyonunda kullanılan değişkenlerin (x ve y) isimleri aynı olması zorunlu değildir. Burada a ve b değişkenleri sırasıyla x ve y değişkenleri yerine konmuştur. toplam adlı tamsayı değişkenine topla fonksiyonunun dönüş değeri (a + b değeri) atanmıştır.

topla fonksiyonunun ana program üstünde tanımlanması

```
#include <stdio.h>

int topla( int x, int y )
{
    return (x+y);
}

main()
{
    int toplam,a,b;

    printf("İki sayı girin : ");
    scanf("%d %d",&a,&b);

    toplam = topla(a,b);

    printf("%d ve %d nin toplamı %d dir.\n", a,b,toplam);

    getch();
}
```

Geri Dönüş Değerleri

return anahtar sözcüğünün iki önemli işlevi vardır:

1. fonksiyonun geri dönüş değerini oluşturur
2. fonksiyonu sonlandırır

Bu deyimden sonra bir değişken, işlem, sabit veya başka bir fonksiyon yazılabilir. Örneğin:

```
return (a+b/c);    /* parantez kullanmak zorunlu değil */
return 10;        /* değişken kullanmak mecbur değil */
return topla(a,b)/2.0; /* önce topla fonksiyonu çalışır */
```

Bir fonksiyonda birden çok geri dönüş değeri kullanılabilir. Fakat, ilk karşılaşılan return deyiminden sonra fonksiyon sonlanır ve çağrılan yere bu değer gönderilir. Örneğin aşağıdaki harf fonksiyonunda beş tane return deyimini kullanılmıştır.

```
char harf(int not)
{
    if( not >= 0 && not < 50 ) return 'F';
    if( not >= 50 && not < 70 ) return 'D';
    if( not >= 70 && not < 80 ) return 'C';
    if( not >= 80 && not < 90 ) return 'B';
    if( not >= 90 ) return 'A';
}
```

Bu fonksiyon kendisine parametre olarak gelen 0-100 arasındaki bir notun harf karşılığını gönderir. Aslında geri gönderilen değer bir tanedir. Eğer bu fonksiyon aşağıdaki gibi çağrılırsa:

```
char harfim;
...
harfim = harf(78);
...
```

harfim değişkenine 'C' değeri (karakteri) atanır.

Örnek : Girilen sayının asal sayı olup-olmadığını bulan program.

```
#include <stdio.h>

//Altprogram: sayinin asal olup olmadigina bakar.
//Sayi asalsa, geriye 1 aksi hâlde 0 degeri doner.
int sayi_asal_mi( int sayi )
{
    int i;
    for( i = 2; i <= sayi/2; i++ )
    {
        // Sayi asal degilse, i'ye tam olarak bolunur.
        if( sayi%i == 0 ) return 0;
    }
    // Verilen sayi hicbir sayiya tam bolunmediyse,
    //asaldir ve geriye 1 doner.
    return 1;
}

// main fonksiyonu
main()
{
    int girilen_sayi;
    int test_sonucu;

    printf( "Lutfen bir sayi giriniz : " );
    scanf( "%d",&girilen_sayi );
    test_sonucu = sayi_asal_mi( girilen_sayi );
    if( !test_sonucu )
        printf("Girilen sayi asal degildir!\n");
    else
        printf( "Girilen sayi asaldir!\n" );

    getchar();
}
}
```

void Fonksiyonlar

Bir fonksiyonun her zaman geri dönüş deęerinin olması gerekmez. Bu durumda return deyimi kullanılmayabilir. Eęer bu anahtar kelime yoksa, fonksiyon ana bloęu bitince kendilięinden sonlanır. Byle fonksiyonların tipi void (boş/geçersiz) olarak belirtilmelidir. Bu tip fonksiyonlar başka bir yerde kullanılırken, herhangi bir deęişkene atanması söz konusu değildir, çünkü geri dönüş deęeri yoktur. Ancak, void fonksiyonlara parametre aktarımı yapmak mümkündür.

```
#include<stdio.h>
void tek_mi_cift_mi( int sayi )
{
    if( sayi%2 == 0 )
        printf( "%d, cift bir sayidir.\n", sayi );
    else
        printf( "%d, tek bir sayidir.\n", sayi );
}

main()
{
    int girilen_sayi;
    printf( "Lutfen bir sayi giriniz: " );
    scanf( "%d",&girilen_sayi );
    tek_mi_cift_mi( girilen_sayi );

    getchar();
}
```

void anahtar sözcüğü C'ye sonradan dahil edilmiştir. Standart C'de (ANSI C) bu deyimin kullanılması zorunlu değildir. Ancak bu deyim okunabilirliği arttırmaktadır. Örneęin:

<i>void hesapla(int deger)</i>	<i>hesapla(int deger)</i>
{	{
...	...
}	}

şeklindeki kullanımlar geçerli ve aynı anlamdadır.

Bir fonksiyona parametre aktarım yapılması zorunlu değildir. Parametresiz bir fonksiyon da tanımlamak mümkündür. Bu durumda argümanlar kısmı ya boş bırakılır yada bu kısma void yazılır.

```
void mesaj_yaz()
{
    printf("Hata olustu !..\n");
}
```

yada

```
void mesaj_yaz(void)
{
    printf("Hata olustu !..\n");
}
```

Özetle :

	Parametresiz	Parametrelili
Geri dönüş değeri yok	<pre>void main(void) { TestFunc(); ... } void TestFunc() { // parametresiz // geri dönen // değer yok }</pre>	<pre>void main(void) { TestFunc(123); ... } void TestFunc(int i) { // paramre "i" // geri dönen // değer yok }</pre>
Geri dönüş değeri var	<pre>void main(void) { x = TestFunc(); ... } int TestFunc(void) { // parametresiz // geri dönen // değer var return 123; }</pre>	<pre>void main(void) { x = TestFunc(123); ... } int TestFunc(int x) { // paramre "x" // geri dönen // değer var return (x + x); }</pre>

Yerel (Local) ve Global Değişkenler

Bir fonksiyon içerisinde tanımladığımız değişkenler yerel değişkendir ve sadece o fonksiyon içerisinde tanımlıdır. main() veya diğer fonksiyonlardan bu değişkenlere ulaşamaz. main() içinde tanımlanan a isimli değişkenle, bir fonksiyon içerisinde tanımlanmış a isimli değişken, bellekte farklı adresleri işaret eder. Dolayısıyla değişkenlerin arasında hiçbir ilişki yoktur. Fonksiyon içerisinde geçen a değişkeninde yapılan değişiklik, main() fonksiyonundakini etkilemez. Benzer şekilde, tersi de geçerlidir.

Yerel değişken dışında, bir de global değişken tipi bulunur. Programın herhangi bir noktasından erişebilen ve nerede olursa olsun aynı bellek adresini işaret eden değişkenler, global değişkenlerdir. Hep aynı bellek adresi söz konusu olduğu için, programın herhangi bir noktasında yapacağımız değişiklik, global değişkenin geçtiği bütün yerleri etkiler.

Bir fonksiyon içerisinde, Global değişkenle aynı isimde, yerel bir değişken tanımlanabilir fakat, bu durumda lokal değişkenle işlem yapılır.

Örnek : Girilen sayının karesini ve küpünü bulan program

```
#include<stdio.h>
// Verilen sayinin karesini hesaplar
void kare_hesapla( int sayi )
{
    int a; //local degisken
    a = sayi * sayi;
    printf( "Sayinin karesi\t: %d\n", a );
}

// Verilen sayinin kupunu hesaplar
void kup_hesapla( int sayi )
{
    int a; //local degisken
    a = sayi * sayi * sayi;
    printf( "Sayinin kupu\t: %d\n", a );
}

main()
{
    int a; //local degisken
    printf( "Sayi giriniz: " );
    scanf( "%d",&a );
    printf( "Girdiginiz sayi\t: %d\n", a );
    kare_hesapla( a );
    kup_hesapla( a );
    getchar();
}
```

Örnek : Girilen sayının karesini bulan program

```
#include<stdio.h>
int sonuc = 0; //global degisken

void kare_hesapla( int sayi )
{
    sonuc = sayi * sayi;
}

int main( void )
{
    int a; //local degisken
    printf( "Sayi giriniz: ");
    scanf( "%d",&a );
    printf( "Girdiginiz sayi\t: %d\n", a );
    kare_hesapla( a );
    printf("Sayinin karesi\t: %d\n", sonuc );
    getchar();
}
```

Örnek : Aşağıdaki programın çıktısını bulunuz.

```
#include <stdio.h>

// function prototype
void Funny(int);

void main(void)
{
    int i;
    for(i = 1; i <= 5; i = i + 1)
        Funny(i);
}

// function definition
void Funny(int num)
{
    // local variable, local to Funny()
    int j;
    for(j = 1; j <= num; j = j + 1)
        printf("j = %d\t", j);
    printf("\n");
}
```

Örnek : Aşağıdaki programın çıktısını bulunuz.

```
#include <stdio.h>

// function prototype
void Funny(int, int);

void main(void)
{
    Funny(5, 8);
}

// function definition
void Funny(int num1, int num2)
{
    for( ; num1 <= num2; num1 = num1 + 1)
        printf("num1 = %d\n", num1);
}
```

Örnek : Kombinasyon Hesabı

$$C_n^m = \frac{m!}{n!(m-n)!}$$

```
#include <stdio.h>
```

```
float combin(int a, int b);  
int fact(int x);
```

```
main()  
{  
  int n, m;  
  printf ("m ve n degerlerini yaziniz: ");  
  scanf("%d %d",&m, &n);  
  printf ("%0.2f", combin(m,n));  
  getchar();  
}
```

```
float combin(int a, int b)  
{  
  int f1, f2, f3;  
  float sonuc;  
  f1 = fact(a);  
  f2 = fact(b);  
  f3 = fact(a - b);  
  sonuc = (float) f1 / (f2 * f3);  
  return sonuc;  
}
```

```
int fact(int x)  
{  
  int fx = 1;  
  int i;  
  for (i = 2; i <= x; i++)  
    fx = fx * i;  
  return fx;  
}
```

Rekürsif fonksiyonlar

Bir fonksiyon doğrudan yada dolaylı olarak kendisini çağırıyorsa buna rekürsif fonksiyon denilir. Döngü deyimleri kullanılarak yazılan tekrarlamalı uygulamalar, ikili ağaç (binary tree) üzerinde arama, ekleme, hızlı sıralama (quicksort) algoritmasının yazılmasında rekürsif fonksiyonlarla yapılması çok uygundur.

```
#include <stdio.h>
```

```
long int faktoriyel(int);
```

```
main()
```

```
{
```

```
    int sayi;
```

```
    printf("Faktoriyeli hesaplanacak sayi: ");
```

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

```
    printf("%d sayisinin faktoryeli=%ld",sayi, faktoriyel(sayi));
```

```
    getchar();
```

```
}
```

```
long int faktoriyel(int x)
```

```
{
```

```
    long int s;
```

```
    if (x>1) s=x*faktoriyel(x-1);
```

```
    else s=1;
```

```
    return(s);
```

```
}
```

Rekürsif fonksiyonlarda bir karşılaştırma işleminden sonra return deyimi kullanmak zorunludur. Eğer kullanılmazsa fonksiyon kendisini sonsuz kez çağırır ve sistemin kilitlenmesine neden olabilir.

Rekürsif olarak tanımlanmış bir fonksiyon, döngü deyimleri kullanılarak yazılmışına göre daha yavaştır ve daha çok bellek alanı kullanır. Bu nedenle her çevrim için fazladan bir fonksiyon çağırısı yapılır ve bu da zaman kaybına neden olur ve her çağırılmasında yerel değişkenleri için bellekte yer ayrılacağından bellek daha fazla kullanılır.

main Fonksiyonu

Ana program olan *main* 'de bir fonksiyondur. C programlarının başlangıcı ve sonu bu fonksiyonla belirlenir. Buna göre, bir C (veya C++) programı sadece bir tane *main* içerir.

main fonksiyonu da geri dönüş değeri gönderebilir. *main* fonksiyonunun geri dönüş değerinin görevi, programın çalışması bittikten sonra sonucu işletim sistemine göndermektir. Program içinde *return* deyimi ile iletilen değer 0 olduğunda, bu işletim sistemi tarafından "program başarılı olarak sonlandı" olarak değerlendirir. Başka bir deyişle,

```
return 0;
```

program, kullanıcının talebi doğrultusunda (olumlu anlamda) "yapması gereken işi yaptı" mesajını işletim sistemine bildirilir. 0'dan farklı herhangi bir değer ise programın sorunlu sonlandığı anlamına gelecektir. Bu yüzden bütün C programlarımızın sonuna `return 0;` ilave edilir.

main fonksiyonunun tipi belirtilmesse;

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

bu durumda geri dönüş değeri (veya tipi) tamsayı (`int`) kabul edilir. Bu şekilde kullanımda, derleyici uyarı (warning) mesajı verebilir. Bu yüzden, aşağıdaki kullanımı tavsiye edilir.

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

Eğer ana programdan bir değer döndürülmeyecekse, *main* fonksiyonunun önüne aşağıdaki gibi void deyimi eklenmelidir. Ancak bu bazı derleyiciler tarafından hata olarak yorumlanır. Bu nedenle, aşağıdaki kullanımlar pek tavsiye edilmez.

```
void main()
{
...
}
yada
void main(void)
{
...
}
```

C Makro Fonksiyonları

Makro fonksiyonlar, altprogram benzeri fonksiyonlar olmayıp, belirli bir işi yapan program parçalarına verilen isimlerdir. Makro fonksiyonlar ile çok sık yazılan program parçalarına verilen simgesel isimler ile tekrar tekrar yazma engellenmiş olur. Makro fonksiyonlar, bir başlık dosyasına eklenerek veya programın içine koyularak kullanılabilir. Makro fonksiyonlarda kullanılan değişkenlerin tanımlamalarında tipleri hakkında herhangi bir bilgi yoktur, bütün sayısal tipler için doğrudan kullanılabilirler. Makro fonksiyonlar **#define** önişlemcisi ile tanımlanır.

```
#define kare (x)      ( (x)*(x) )
#define buyuk (a,b)  ( (a) > (b) ) ? (a) : (b)
#define yaz(s)      puts("Merhaba "); puts(s);
```

Örnek :

```
#include <stdio.h>
#include <math.h>

// makro fonksiyonlar
#define kare(x)  (x*x)
#define topl(x,y) (x+y)
#define carp(x,y) (x*y)
#define hipo(x,y) sqrt(x*x+y*y)

main(void)
{
    float a=3.0, b=4.0;

    printf("kare(2) = %f\n",kare(2));
    printf("topl(a,b) = %f\n",topl(a,b));
    printf("carp(a,b) = %f\n",carp(a,b));
    printf("hipo(a,b) = %f\n",hipo(a,b));
}
```

ÇIKTI

```
kare(2) = 4.000000
topl (a, b) = 7.000000
carp(a, b) = 12.000000
hi po(a, b) = 5.000000
```

Bir makro fonksiyon tanımlanırken değişkenlerin parantez içine alınması zorunlu değildir, operatörlerin önceliklerinden dolayı yanlış hesaplamalara neden olmamak için parantezlerin kullanımı daha iyi olur.

Örnek :

```
#include <stdio.h>

#define büyük(a,b) ( (a>b) ? a:b)

main()
{
    int x,y,eb;

    printf("iki sayı girin: ");
    scanf("%d %d",&x,&y);

    eb = büyük(x,y);

    printf("büyük olan %d\n",eb);

    getchar();
}
```

ÇIKTI

iki sayı girin: 8 6

büyük olan 8

Makro fonksiyon ile kütüphane veya kullanıcı tanımlı fonksiyonlar arasındaki farklar:

Makrolar fonksiyonlara alternatif olarak kullanılmalarına karşılık, makrolar ile fonksiyonlar arasında çok önemli farklar bulunmaktadır:

Makrolar kaynak kodu dolayısıyla da çalışabilir (exe) kodu büyütürler. Makrolar önışlemci aşamasında değerlendirilir. Örneğin kare makrosunun #define önışlemci komutuyla tanımlandıktan sonra kaynak kod içinde yüz kere çağırıldığını düşünelim. Önışlemci kaynak kodu ele aldığıında yüz tane çağırılma ifadesinin her biri için makroyu açacaktır. Derleyici modülü kaynak kodu ele aldığıında artık makroları değil makroların açılmış şeklini görecekler. Makro açılımları kaynak kodu büyütecektir. Kaynak kodun büyümesinden dolayı çalışabilen dosyanın boyutu da büyüyecektir. Oysa makro yerine bir fonksiyon tanımlansaydı, Fonksiyon çağırılması durumunda yalnızca çağırılma bilgisi kayanak koda yazılmaktadır.

Makroların en büyük avantajı, fonksiyon çağırma işlemindeki görelî yavaşlıktan kaynaklanır. Fonksiyon çağırması sembolik makine dili (assembler) düzeyinde ele alındığıında, bazı ilave makine komutları da icra edilmekte ve bu makine komutlarının icrası ve yapılan bazı ilave işlemler icra süresini uzatacaktır.

Hazır fonksiyonlar kütüphaneler (*.lib) içinde yada başlık dosyaları (*.h) içinde bulunurlar. Kütüphane içindeki fonksiyonlar programda kullanılırsa, o fonksiyonun kodu derleme sırasında programa eklenir ve gerektiğinde programcı tarafından tanımlanmış fonksiyon gibi çağırılır.

İşte ilginç bir makro daha. Daha önce anlatılan takas(a,b) fonksiyonu gösterici kullanmadan aşağıdaki makro ile yazılabilir: