

C PROGRAMLAMA DİLİ

* Dersler

- * Giriş
- * Veri Tipleri, Değişkenler
- * Operatörler
- * Temel G/Ç Fonksiyonları
- * Temel Kütüphane Fonksiyonları
- * Karşılaştırma Deyimleri
- * Döngüler
- * Fonksiyonlar
- * Diziler
- * Gösterici (Pointer) Kavramı
- * Katarlar (Stringler)
- * Dinamik Bellek Yönetimi
- * Gösterici Uygulamaları
- * Yapılar ve Birlikler
- * C Makroları

* Yararlanılan Kaynaklar

Rifat Çölkesen ,”İşte C”, Sistem Yayıncılık

* Kullanılan Programlar

Dev-C++, <http://www.bloodshed.net/devcpp.html>

C4Droid, <https://play.google.com>

Ders 1: Giriş

- [1.1 Tarihçe](#)
- [1.2 Neden C?](#)
- [1.3 İlk C Programı](#)
- [1.4 Başlık Dosyaları](#)
- [1.5 Kaynak Kodunun Derlenmesi](#)
- [1.6 C Kodlarının Temel Özellikleri](#)
- [1.7 Kod Yazımı için Bazı Tavsiyeler](#)

1.1 Tarihçe

C Programlama Dili genel amaçlı orta seviyeli ve yapısal bir programlama dilidir. 1972 yılında Dennis Ritchie tarafından Bell Telefon Laboratuvarında Unix işletim sistemi ile kullanılmak için tasarlanmıştır. C, özellikle sistem programlamada sembolik makine dili (Asembler) ile tercih edilmektedir. İşletim sistemleri, derleyiciler ve debug gibi aşağı seviyeli sistem programlarının yazılımında yoğun olarak C programlama dili kullanılır.

C'nin yayılması ve gelişmesi, büyük bir bölümü C dili ile yazılan UNIX işletim sisteminin popüler olmasıyla başlamıştır. C Programlama Dili, hemen her alanda kullanılmaktadır. Günümüzde nesneye yönelik programlama dilleri (C++, Java) ve script dilleri (JavaScript, JavaApplet, PHP) gibi programlama dilleri C Programlama Dili'nden esinlenmiştir.

C taşınabilir (portable) bir dildir. Yani herhangi bir C programı hiçbir değişikliğe uğramadan, veya çok az bir değişimle, başka bir derleyicide ve/veya işletim sisteminde derlenebilir. Örneğin, Windows işletim sistemlerinde yazılan bir C kodu, Linux, UNIX veya VAX gibi işletim sistemlerinde de derlenebilir. Taşınabilirlik, herkesin kabul ettiği bir standart ile gerçekleştirilebilir. Bugün, C Programla Dili için **American National Standards Institute (ANSI)** kurumunun Mart 2000'de belirlediği **C99: ISO/IEC 9899:1999** standartı **Standart C** olarak kabul edilmiştir. ANSI C ile, dilin temel yapısı, operatörler, veri tipleri ve bir C derleyicisinde en azından olması gereken fonsiyonları belirlenmiştir.

Başlangıçta sadece UNIX işletim sistemini yazmak için geliştirilen C dilinin,

1. Assembly dili ile yüksek seviyeli dillerin yeteneklerine sahip olması,
2. Geniş operatör yetenekleri,
3. Yapısal programlama için gerekli kontrol ve döngü deyimlerine sahip olması nedeniyle bir çok uygulamada kullanılır.

Kullanım alanları;

1. Bilimsel çalışmalarda,
2. Mühendislik uygulamalarında,
3. Mikroişlemci-Mikrodenetleyici uygulamalarında.

1.2 Neden C?

C Programlama Dili'ni popüler kılan önemli nedenler aşağıda listelenmiştir:

- C, güçlü ve esnek bir dildir. C ile işletim sistemi veya derleyici yazabilir, kelime işlemciler oluşturabilir veya grafik çizebilirsiniz.
- C, iyi bir yazılım geliştirme ortamına sahiptir.
- C, özel komut ve veri tipi tanımlamasına izin verir.
- C, taşınabilir bir dildir.
- C, gelişimini tamamlamış ve standardı oluşmuş bir dildir.
- C, yapısal bir dildir. C kodları *fonksiyon* olarak adlandırılan alt programlardan oluşmuştur.
- C++, Java, JavaScript, JavaApplet, PHP, C#, ... gibi diller C dilinden esinlenmiştir.
- C, diğer dillere kıyasla çok hızlı çalışır.
- C dili ile Assembly dilindeki donanıma bağımlılık ortadan kalkar.

1.3 İlk C Programı

Program 1.1 de verilen C programı derlendikten sonra, ekrana **C Programlama Dili!** yazısını basan yalın bir C programıdır. Satır başlarına yerleştirilen 1:, 2: 3: ... rakamlarının yazılmasına gerek yoktur. Bu rakamlar sadece daha sonra program ile ilgili açıklama yapılırken, ilgili satırda bulunan kodlar izah edilirken kullanılacaktır. Bu programın bilgisayarda ilk.c adı ile kaydedilmiştir.

*Derlendikten sonra ekrana **C Programlama Dili!** yazar*

```
/* ilk.c: ilk C programı */
#include <stdio.h>

main()
{
    printf("C Programlama Dili!\n");
}
```

`/* ... */`

Programda, 1. satırda `/* ... */` sembolleri görülmektedir. Bu ifadeler arasında yazılan herhangi bir metin, işlem vb. satırlar, derleyici tarafından işlenmez (değerlendirilmez). Yani `/* */` ifadeleri açıklama operatörüdür.

NOT

Açıklama operatörü olarak C++ tarzı iki-bölü (`//`) de kullanılmaktadır. Günümüzde birçok C derleyicisi `//` operatörünü desteklemektedir. Bu operatörü kullanmadan önce derleyicinizin bu operatörü desteklediğinden emin olun.

```
/*
  Bu satırlar derleyici tarafından
  değerlendirilmez. Ayrıca programın      C tarzı
  çalışma hızını da değiştirmez.
*/

// Bu satırlar derleyici tarafından
// değerlendirilmez. Ayrıca programın      C++ tarzı
// çalışma hızını da değiştirmez.
```

`#include <stdio.h>`

2. satırdaki `#include` deyimi, programda eklenecek olan başlık dosyanını işaret eder. Bu örnekte verilen başlık dosyası (header file) `stdio.h` dir. `#include <stdio.h>` ifadesi `stdio.h` dosyasının derleme işlemine dahil edileceğini anlatır.

`main()`

4. satırdaki `main()` özel bir fonksiyondur. Ana program bu dosyada saklanıyor anlamındadır. Programın yürütülmesine bu fonksiyondan başlanır. Dolayısıyla her C programında bir tane `main()` adlı fonksiyon olmalıdır.

`printf()`

6. satırdaki `printf()` standart kütüphane bulunan ekrana formatlı bilgi yazdırma fonksiyondur. `stdio.h` dosyası bu fonksiyonu kullanmak için program başına ilave edilmiştir. Aşağıda `printf()` fonksiyonunun basit kullanımı gösterilmiştir.

Örnek kullanım şekli

```
printf("Element: Aluminyum");
printf("Atom numarası = %d",13);
printf("Yoğunluk = %f g/cm3",2.7);
printf("Erime noktası = %f derece",660.32);
```

Ekranda yazılacak ifade

```
Element: Aluminyum
Atom numarası = 13
Yoğunluk = 2.7 g/cm3
Erime noktası = 660.32 derece
```

1.4 Başlık Dosyaları

C dilinde bir program yazılırken, başlık dosyası (header file) olarak adlandırılan bir takım dosyalar `#include` önişlemcisi kullanılarak program içine dahil edilir. C kütüphanesinde bulunan birçok fonksiyon, başlık dosyaları içindeki bazı bildirimleri kullanır. Bu tür dosyaların uzantısı `.h` dir. ANSI C'deki standart başlık dosyaları şunlardır:

<code>assert.h</code>	<code>locale.h</code>	<code>stddef.h</code>
<code>ctype.h</code>	<code>math.h</code>	<code>stdio.h</code>
<code>errno.h</code>	<code>setjmp.h</code>	<code>stdlib.h</code>
<code>float.h</code>	<code>signal.h</code>	<code>string.h</code>
<code>limits.h</code>	<code>stdarg.h</code>	<code>time.h</code>

Bir çok C derleyicisinde yukarıdakilere ek olarak tanımlanmış başlık dosyaları da vardır.

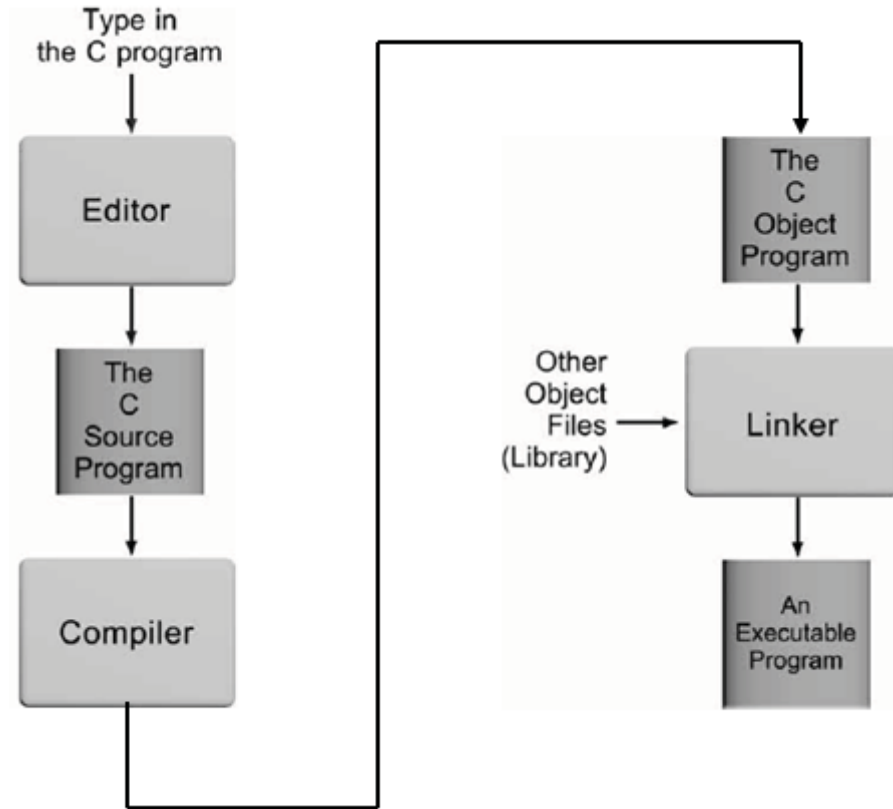
ilk.c programında kullanılan başlık dosyası `stdio.h`, `#include <stdio.h>` ifadesi ile derleme işlemine dahil edilmiştir. `stdio.h` standard giriş/çıkış (STandarD-Input-Output) kütüphane fonksiyonları için bazı bildirimleri barındıran bir dosyadır. Programda kullanılan `printf()` fonksiyonunu kullanmadan önce bu başlık dosyası programın başına mutlaka ilave edilmelidir. Aksi halde derleme esnasında

```
undefined reference to _printf
```

şeklinde bir hata mesajı ile karşılaşılır.

1.5 Kaynak Kodunun Derlenmesi

C programları veya kaynak kodları (source code) uzantısı .c olan dosyalarda saklanır. Kaynak kod, bir C derleyicisi (C compiler) ile nesne koduna (object code) daha sonra uygun bir bağlayıcı (linker) programı ile işletim sisteminde çalıştırılabilen (executable) bir koda dönüştürülür.



1.6 C Kodlarının Temel Özellikleri

Bir C programı aşağıda verilen özellikleri mutlaka taşınmalıdır.

- Yazılımda kullanılacak olan her fonksiyon için ilgili başlık dosyası programın başına ilave edilmedlidir.
- Her C programı main() fonksiyonunu içermelidir.
- Program içinde kullanılacak olan değişkenler ve sabitler mutlaka tanımlanmalıdır.
- Satırın sonuna ; işareti konmalıdır.
- Her bloğun ve fonksiyonun başlangıcı ve bitişi sırasıyla { ve } sembolleridir.
- C dilinde yazılan kodlarda küçük-büyük harf ayrımı vardır (case sensitive).
Örneğin A ile a derleyici tarafından farklı değerlendirilir.
- Açıklama operatörü /* */ sembolleridir.

1.7 Kod Yazımı için Bazı Tavsiyeler

- Kodların açıklamalarını program yazıldıkça yapılmalıdır. Bu unutulmaması gereken çok önemli husustur.
- Değişken, sabit ve fonksiyon adları anlamlı kelimelerden seçilip yeterince uzun olmalıdır. Eğer bu isimler bir kaç kelimeden oluşacak ise, kelimeler alt çizgi (_) ile ayrılmalıdır veya her kelime büyük harfle başlamalıdır. Örneğin:

- `int son_alinan_bit;`
- `void KesmeSayisi();`
- `float OrtalamaDeger = 12.7786;`

- Sabitlerin bütün harflerini büyük harfle yazın. Örneğin:

- `#define PI 3.14;`
- `const int STATUS=0x0379;`

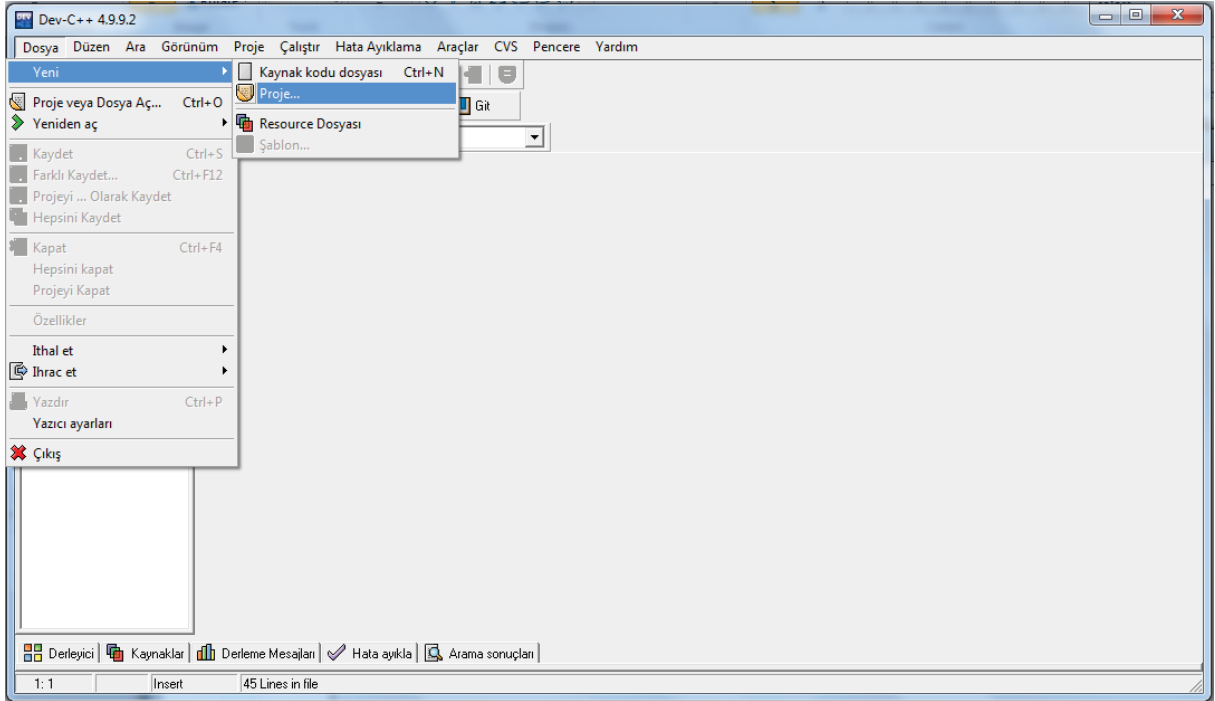
Her alt yapıya girerken birkaç boşluk veya TAB tuşunu kullanın. Bu okunabilirliği arttıracaktır. Örneğin:

- `k = 0;`
- `for(i=0; i<10; i++)`
- `{`
- `for(j=0; j<i; j+=2)`
- `{`
- `do{`
- `if(j>1) k = i+j;`
- `x[k] = 1.0/k;`
- `}while(k!=0);`
- `}`
- `}`

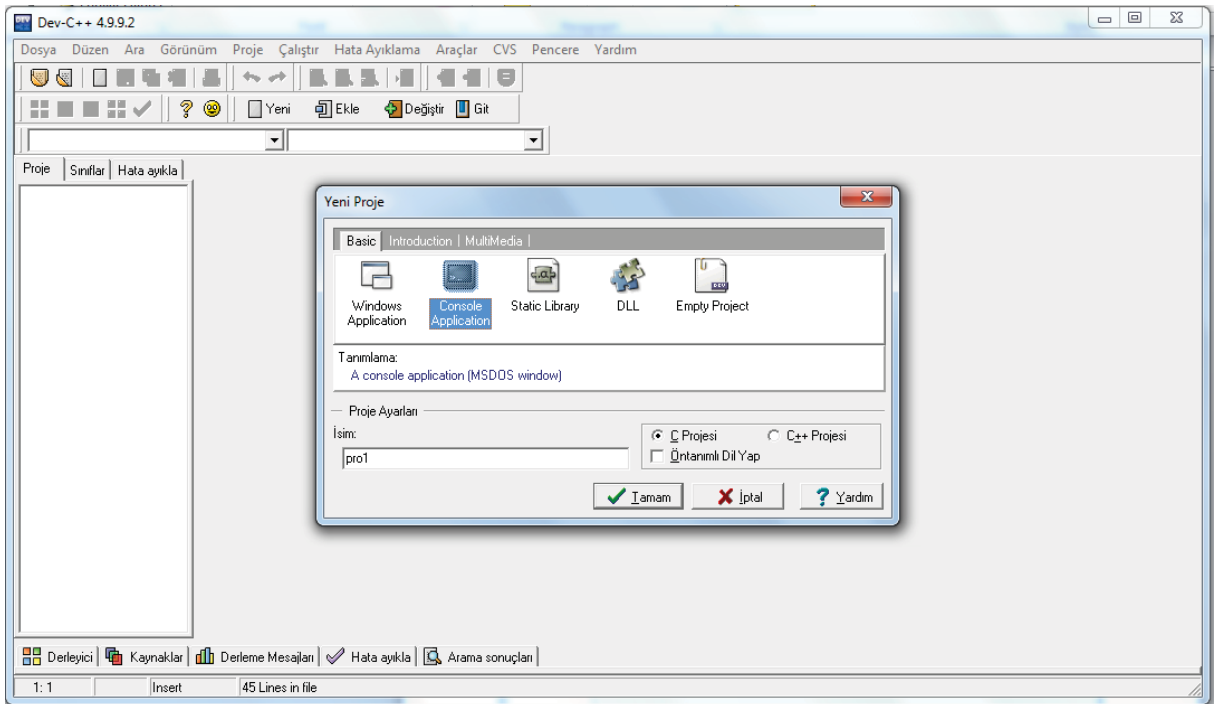
- Aritmetik operatörler ve atama operatörlerinden önce ve sonra boşluk karakteri kullanın. Bu, yazılan matematiksel ifadelerin daha iyi anlaşılmasını sağlayacaktır.Örneğin:
-
- $h_max = \text{pow}(V_o, 2) / (2 * g);$
- $Tf = 2 * V_o / g;$
- $V_y = V_o - g * t;$
- $y = V_o * t - (g * t * t) / 2.0;$
- $z = (a * \cos(x) + b * \sin(x)) * \log(\text{fabs}(y));$
- Program bittikten sonra tekrar tekrar programınızı inceleyerek, programınızı daha iyi şekilde yazma yollarını arayın ve aynı fonksiyonları daha kısa algoritmalarla ve/veya daha modüler şekilde elde etmeye çalışın.

Dev-C++ programının kullanılması ;

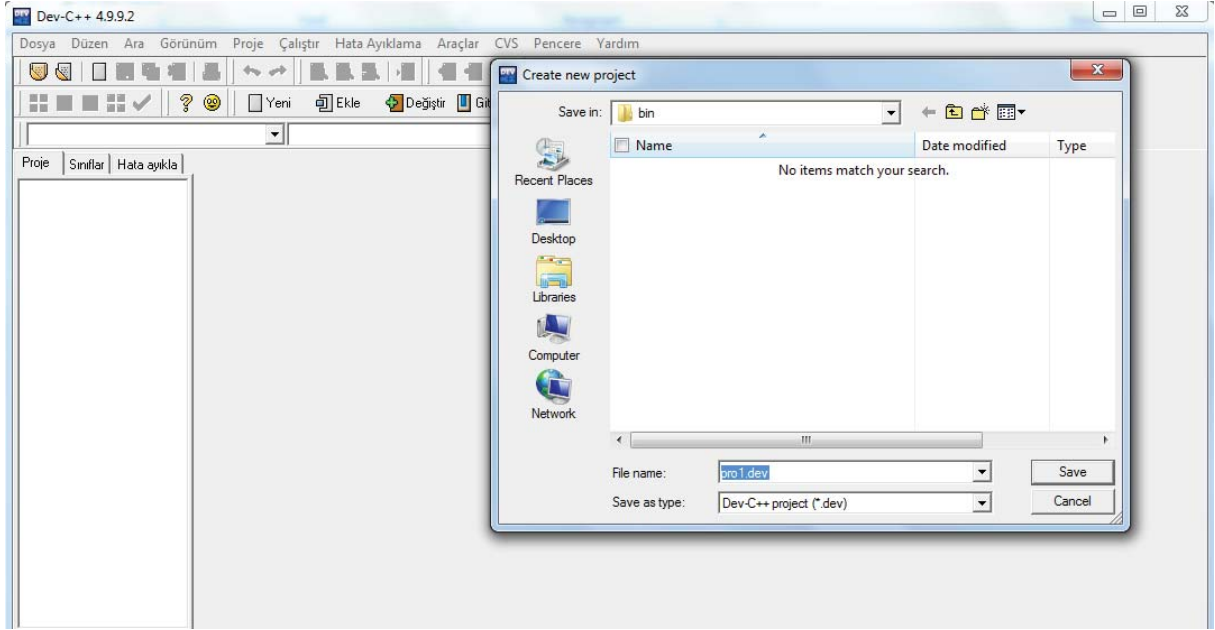
1. Yeni proje seçilir.



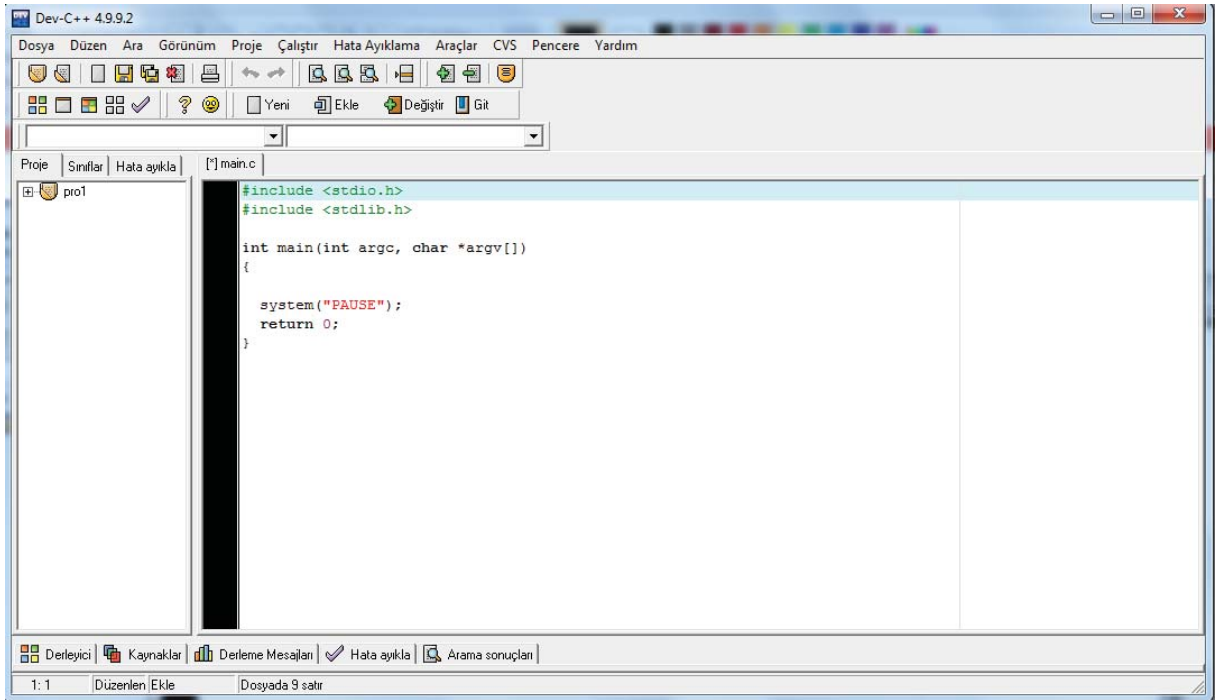
2. Sırasıyla a) console application seçilir, b) Proje ayarlarından projeye isim verilir, c) C projesi seçilir ve tamam 'a' tıklanılır.



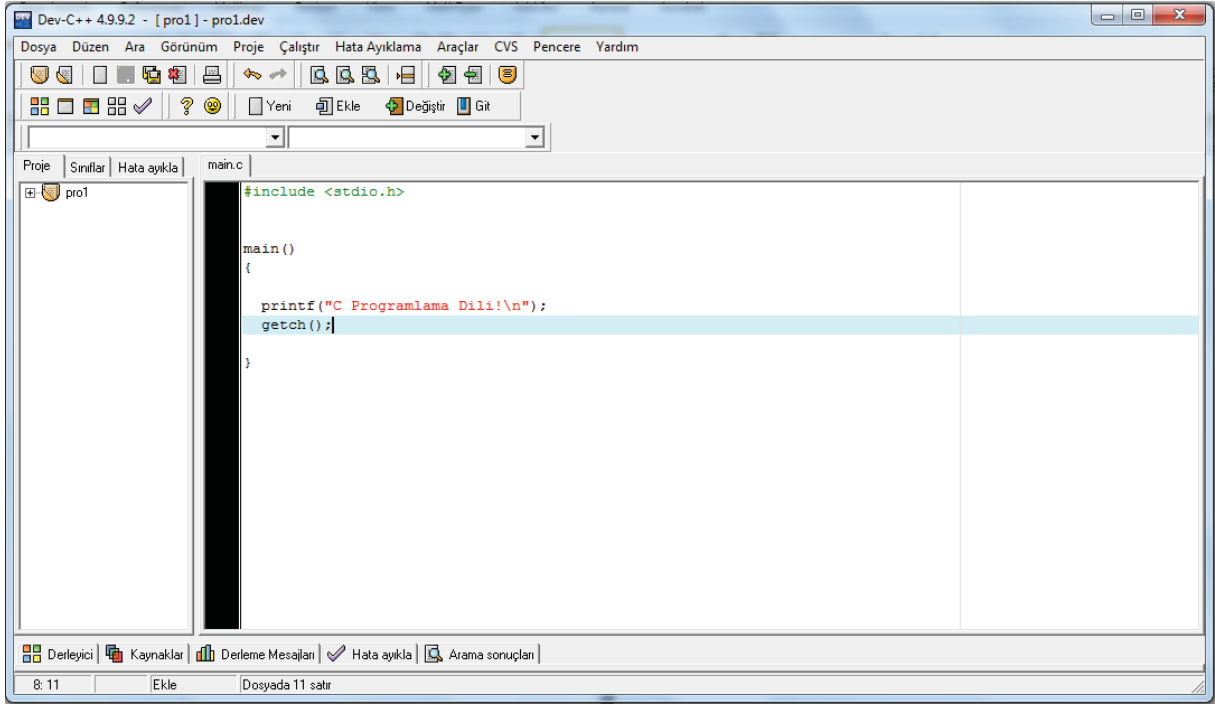
3. Proje bin klasörüne kaydedilir.



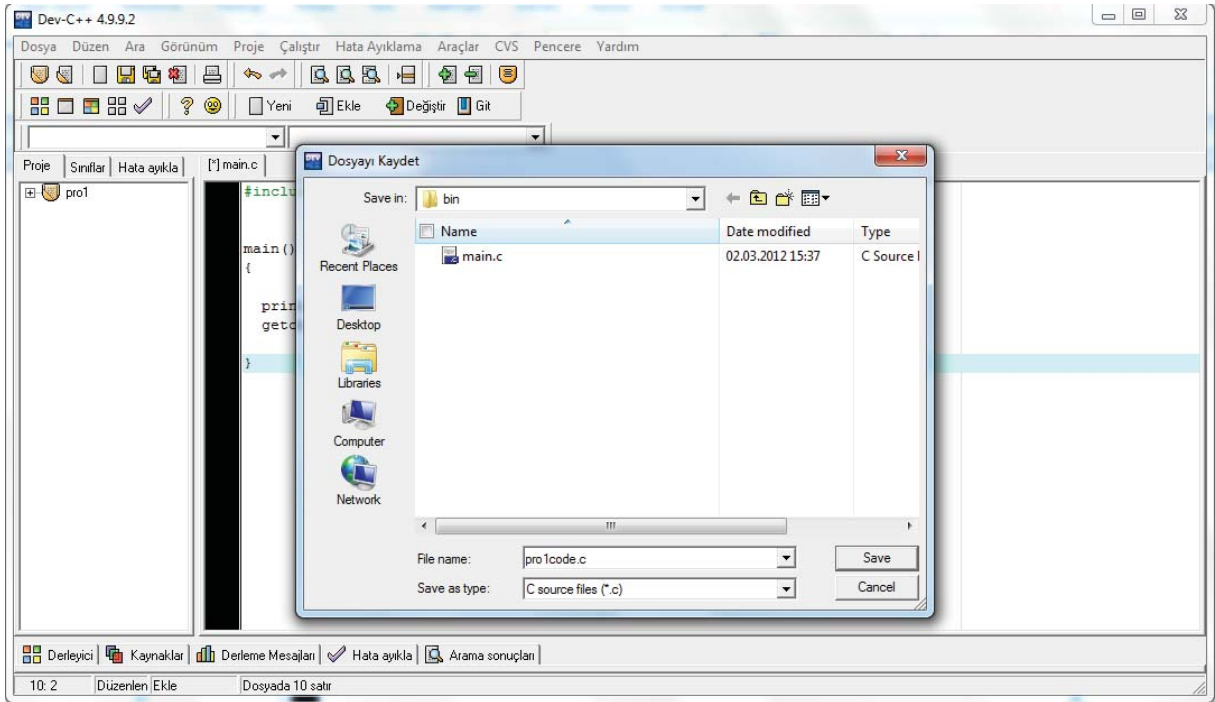
4. Oluşturulan proje için compiler kullanıcuya bir program yapısı oluşturur.



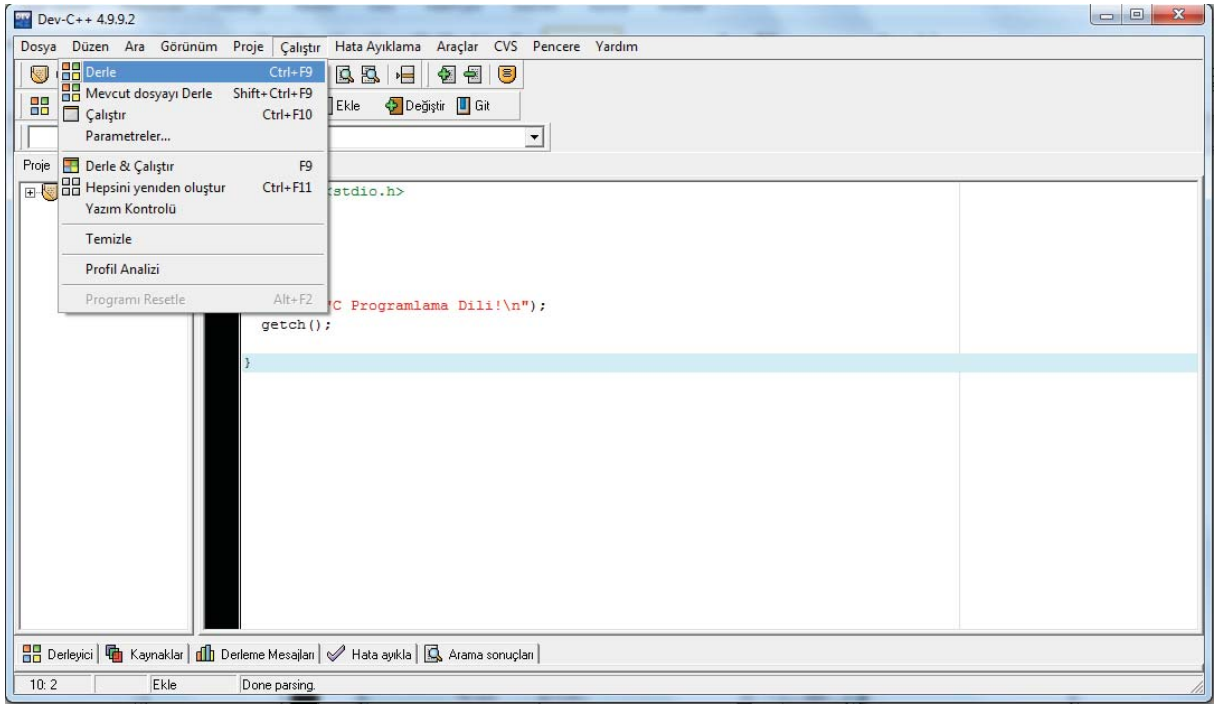
5. Kullanıcı kendi kaynak kodlarını (programını) editore yazar.



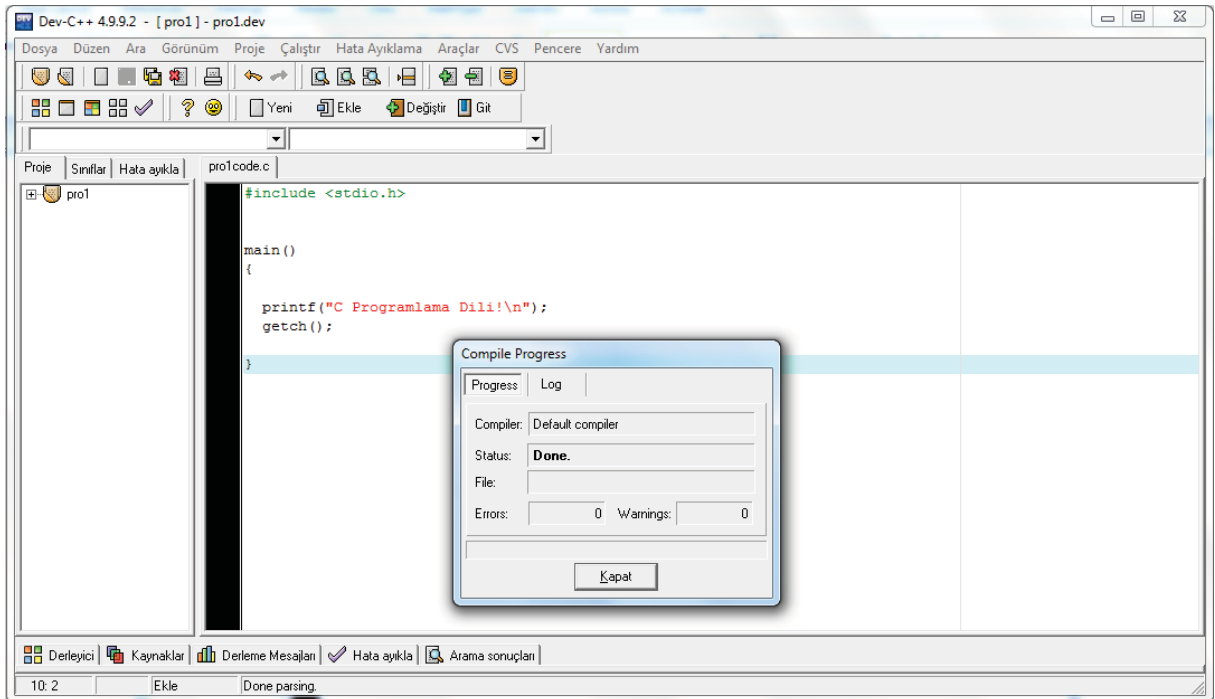
6. Kaynak kod bin klasörüne isim verilerek kaydedilir.



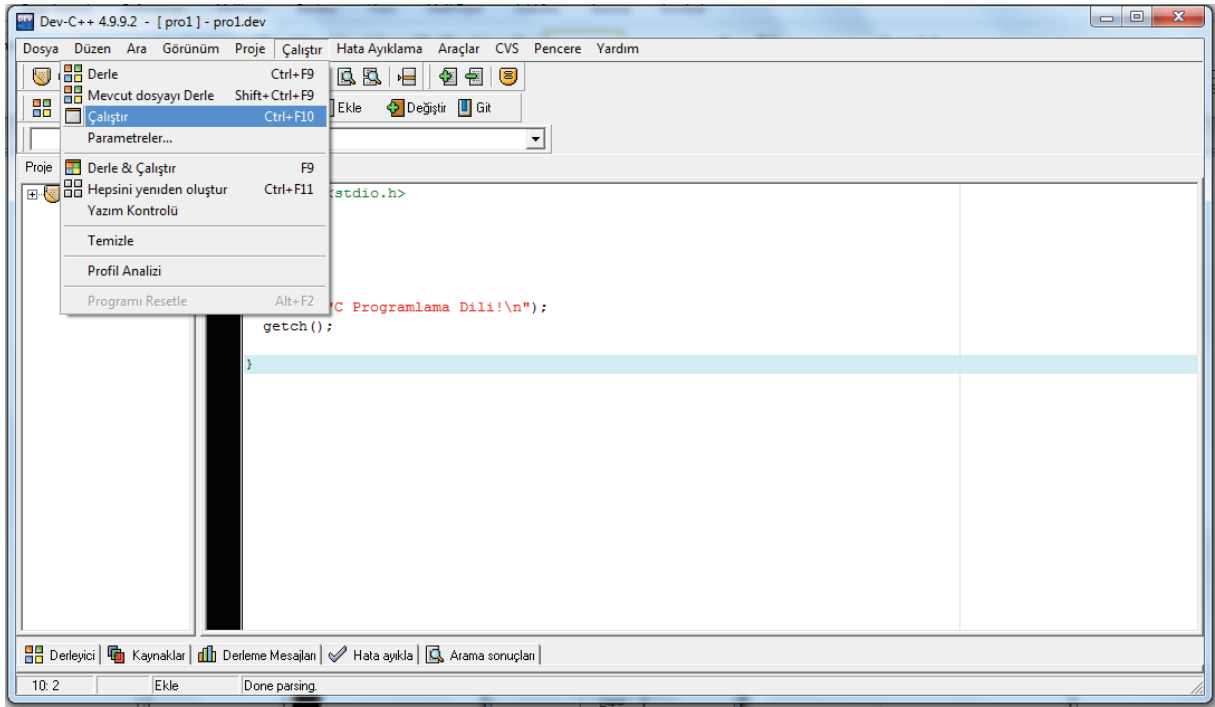
7. Kaynak kod derlenir.



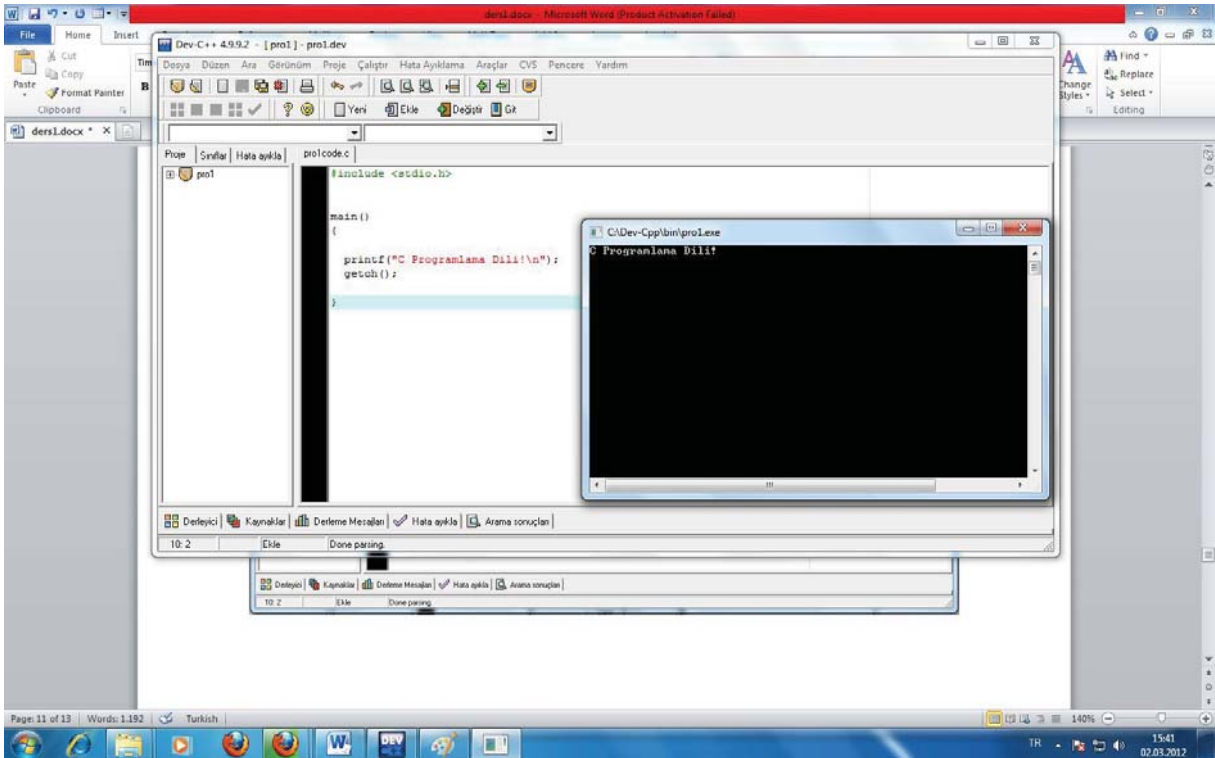
8. Derleme sonucunda Errors : 0 olmalıdır. Eğer sıfır değilse hata bulunup düzeltilir ve tekrar derlenir.



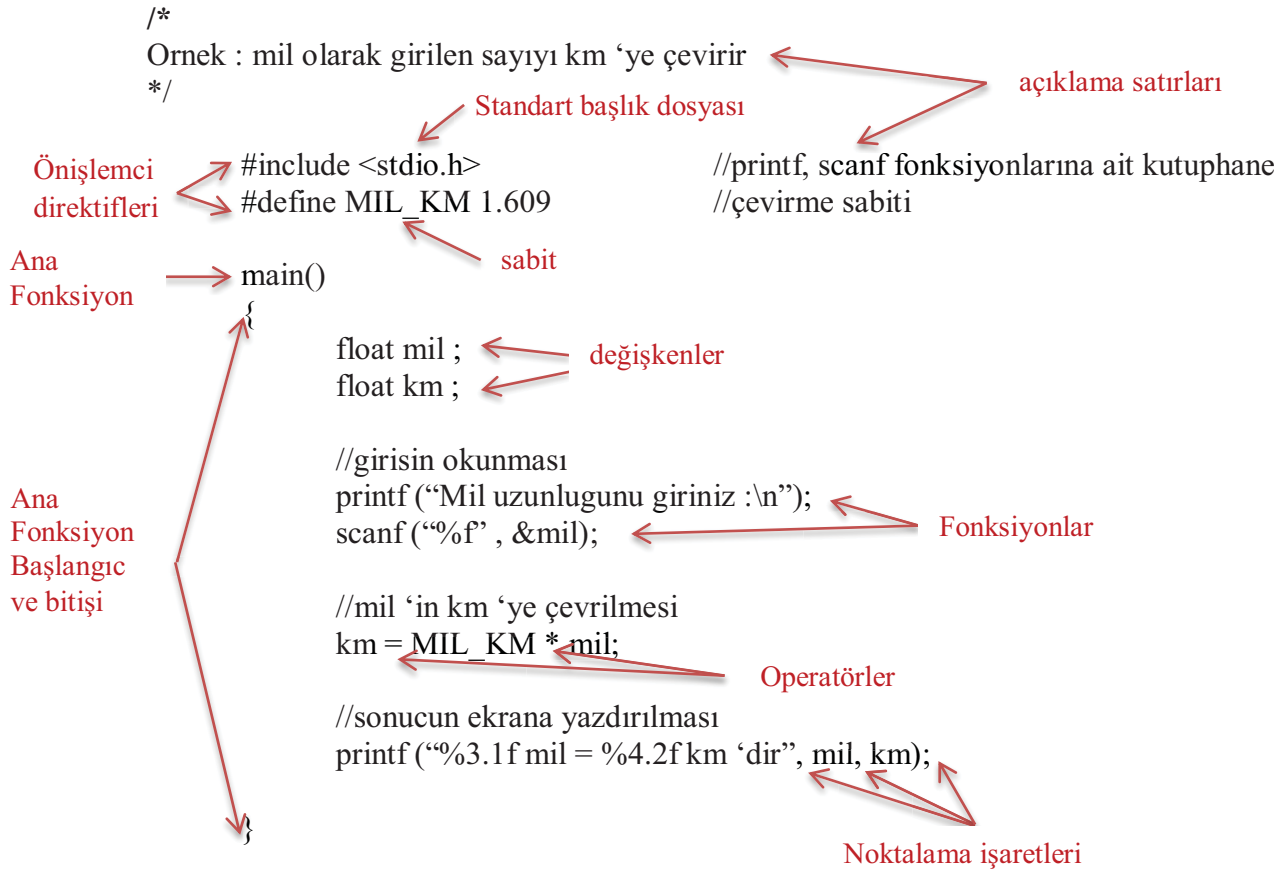
9. Kodlar çalıştırılır.



10. Sonuçlar gözlenir.



C PROGRAM YAPISI



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 her bir 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 **önişlemcisi** yardımıyla sağlanır. Önişlemciler çeşitli emirlerden oluşabilir. Bu emirler C derleyicisinin kaynak kodunu denetlemekte kullanılır. Önişlemci emirleri C programı içinde (#) işareti ile başlar. C'nin en çok kullanılan önişlemci emirleri **#include** ve **#define** ile tanımlanmaktadır. Önişlemci 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 görülmektedir.

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 takdirde, veri tipi ile saklanacak değerın sıfır ve sıfırdan büyük olması sağlanır. İşaretli 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_adı;

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 bildirime 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_adı=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** önilemci komutu ile de tanımlanabilir. Bu komutla sabit bildirimi, 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          //önilemci 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 = 75;      /* i = 75, 10 tabanında */
i = 0113;    /* i = 75, 8 tabanında */
i = 0x4b;    /* i = 75, 16 tabanında */
i = 0x4B;    /* i = 75, 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 */
x = 123.456e+0;       /* üstel gösterim */
x = 1.23456e+2;       /* üstel gösterim */
x = 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 = "Merhaba Dünya"    /* 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 bildirimi 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
```

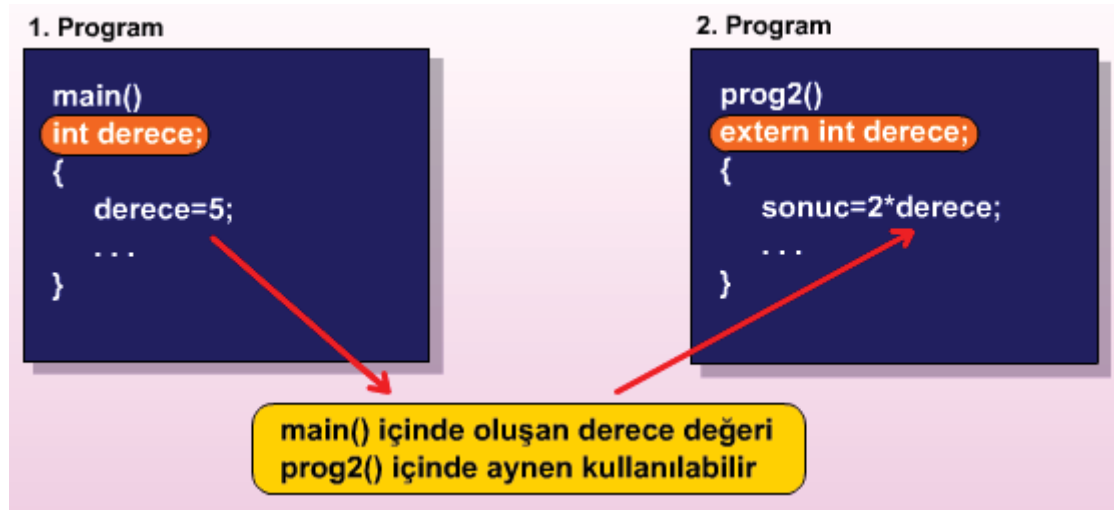
```
fonk1()
{
    int i;
    printf("i=%d",i);
}

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 kullanılamaz. 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 değerin 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/çıkış deyimleri yoktur. Girdi/çıkış işlemleri, her zaman, fonksiyonlar çağrılarak yapılır. Tabii ki, girdi/çıkış yapmak için kullanılan fonksiyonların programcı tarafından yazılmasına gerek yoktur. Hemen hemen bütün C ortamlarında girdi/çıkış 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/çıkış fonksiyonu çağrılmadan veya değişkenleri kullanılmadan önce

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

yazılarak kaynak programın içine kopyalanması gerekir.

Kullanıcının girdi/çıkış yapması için, üç girdi/çıkış 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 çıkışı; hata mesajlarını ekrana bastırmak için kullanılan bir kanaldır.

printf() Fonksiyonu

Standart C kütüphanesinde bulunan printf() fonksiyonu, değişkenlerin tuttuğu değerleri, onların adreslerini veya bir mesajı ekrana belli bir düzene (format) standart çıkışa (stdout), yani ekrana, yazdırmak için kullanılan fonksiyondur.

Kullanımı :

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

Eğer format stringini takip eden argument'lar mevcut ise bu argument değerleri format'ta verilen forma uygun olarak çıkışa yazar. Geri dönüş değeri olarak, yazdığı karakter sayısını döndürür. Hata oluştuğunda negatif değer döndürür.

Basit olarak ekrana Hata oluştu!.. şeklinde bir mesaj yazdırma işlemi:

```
printf("Hata Oluştı!..");
```

*format üç kısımdan oluşmaktadır:

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

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). Ekrana yazdırılmak istenen değişkenin tipi, % işaretinden sonra belirtilir. Örneğin: printf("x in değeri %d dir"); gibi.

Tip karakterleri (Tamsayılar)

Tip Karakteri	Anlamı
%d	işaretli tamsayı (onluk sistem)
%i	işaretli tamsayı (onluk sistem)
%o	işaretli 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 = 15;

    printf("%d\n", a);
    printf("%i\n", a);
    printf("%o\n", a);
    printf("%u\n", a);
    printf("%x\n", a);
}
```

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) (Sayının büyüklüğüne bağlı)

Ornek:

```
#include<stdio.h>

int main()
{
    float sayi = 12.345678;

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

Tip karakterleri (Karakter veya karakter dizisi)

%c	tek bir karakter
%s	karakter dizisi (string)

Ornek :

```
#include<stdio.h>

int main()
{
    char a = 'z';
    char b[]="ALI";

    printf("%c\n", a);
    printf("%s\n", b);

    printf("\n\n%s\n", "Merhaba");
}
```

Karisik Kullanim:

```
#include<stdio.h>

int main()
{
    int not= 12;
    float pi = 3.14;
    char kr = 'A';

    printf(" not = %d , pi = %f ve kr = %c dir", not, pi, kr);
    printf("\n");
}
```

printf() fonksiyonu esnektir. Parametreler herhangi bir C deyimi olabilir. Örneğin x ve y nin toplamı şöyle yazılabilir:

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

Ornek:

```
/*
    Sayısal değerleri ekrana yazdırmak için printf fonksiyonunun kullanımı */

#include <stdio.h>

main()
{
    int a = 2, b = 10, c = 50;
    float f = 1.05, g = 25.5, h = -0.1, yuzde;

    printf("3 tamsayi : %d %d %d\n", a, b, c);
    printf("3 tamsayi [TAB] : %d \t%d \t%d\n", a, b, c);

    printf("\n");

    printf("3 reel sayi (yanyana) : %f %f %f\n", f, g, h);
    printf("3 reel sayi (altalta) : \n%f\n%f\n%f\n", f, g, h);

    yuzde = 220 * 25/100.0;
    printf("220 nin %%25 i %f dir\n", yuzde);
    printf("%f/%f isleminin sonucu = %f\n", g, f, g / f);

    printf("\nprogram sonunda beep sesi cikar...\a");
}
```

ÇIKTI

```
3 tamsayi      : 2 10 50
3 tamsayi [TAB]: 2   10   50

3 reel sayi (yanyana) : 1.050000 25.500000 -0.100000
3 reel sayi (altalta) :
1.050000
25.500000
-0.100000

220 nin %25 i 55.000000 dir
25.500000/1.050000 isleminin sonucu = 24.285715

program sonunda beep sesi cıkar...
```

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:

```
01:  /*
02:    printf fonksiyonunun geri dönüş değerini gösterir */
03:
04:  #include <stdio.h>
05:
06:  int main()
07:  {
08:      int karSay;
09:      int sayi = 1234;
10:
11:      karSay = printf("Ugurlu sayim = %d\n",sayi);
12:
13:      printf("Ust satirda karakter sayisi: %d dir\n", karSay);
14:
15:
16:  }
```

ÇIKTI

```
Ugurlu sayim = 1234
Ust satirda karakter sayisi: 20 dir
```

12. satırdaki işlemle, hem ekrana **Ugurlu sayim = 1234** iletisi bastırılmakta, hem de **karSay** değişkenine bu iletinin uzunluğu atanmaktadır. Ekrana basılan karakterlerin sayısı (\n karakteri dahil) 20 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 Genisligi 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. Duyarlılık orginal değerdeki ondalık basamak sayısından küçük ise yuvarlama olur.
5. String'lerde duyarlılık yazdırılacak karakter sayısıdır.

Kullanımı :

%[alan_genisligi][.duyarlilik]tip_karakteri

Ondalikli sayilar icin $\text{alan_genisligi} > \text{duyarlilik} + 2$ olmalıdır.

Ornekler :

```
#include<stdio.h>
main()
{
    int sayi1 = 1;
    int sayi2 = 12;
    int sayi3 = 123;
    int sayi4 = 1234;
    int sayi5 = 12345;
    printf("Alan Genisligi :\n");
    printf("%3d\n", sayi1);
    printf("%3d\n", sayi2);
    printf("%3d\n", sayi3);
    printf("%3d\n", sayi4);
    printf("%3d\n", sayi5);
    printf("Duyarlilik :\n");
    printf("%.3d\n", sayi2);
    printf("%.3d\n", sayi3);
    printf("%.3d\n", sayi4);
}
```

```
#include <stdio.h>
main()
{
    char x[]="KOCAELI";

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

    printf("%10s\n",x);
    printf("%8s\n",x);
    printf("%5s\n",x);
    printf("%10.2s\n",x);
}
```

```
#include <stdio.h>
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);
}
```

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);

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

veya

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

Temel Giriş/Çıkış Fonksiyonları (Devam)

Örnek :

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 hücresinin 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 hariç), verilerin önündeki boşluk, tab, yeni satır vb. karakterler önemsizdir, 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	işaretli tamsayı (onluk sistem)
%i	işaretli tamsayı (onluk, sekizlik veya 16 lik sistem)

Örnek:

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

fonksiyon

Klavye

etkisi

```
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%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();
```

```
}
```

```
7 tane tamsayi giriniz :  
-70 -70 070 0x70 70 70 70  
girdiginiz sayilar:  
-70 -70 56 112 56 70 112
```

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();  
  
}
```

```
String yaziniz: ALI  
Birinci karakter : A  
diger kisim : LI
```

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();  
}
```

```
Bir real sayi giriniz: 1.2  
Bir tamsayi sayi giriniz: 3  
  
    1.2 * 1.2 = 1.44  
    3 * 3 = 9
```


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();  
}
```

```
a  
a  97
```

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 simgelerdir. 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.

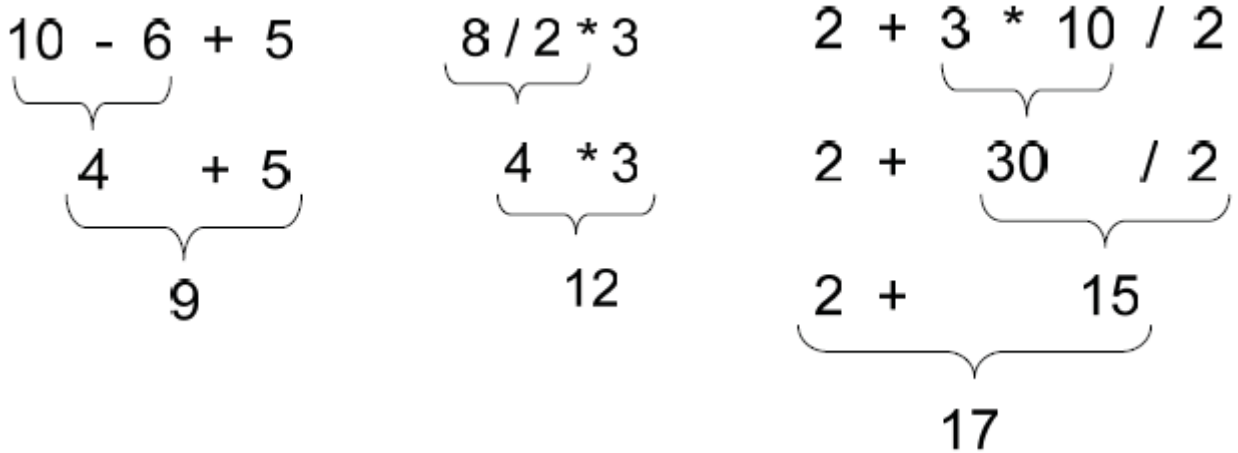
Aritmetik Operatörler

Operatör	Açıklama	Örnek	Anlamı
+	toplama	$x + y$	x ve y nin toplamı
-	çıkarma	$x - y$	x ve y nin farkı
*	carpma	$x * y$	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.



Aritmetik operatörlerin değerlendirme sırası (önceliği) parantez kullanılarak değiştirilir. Parantez kullanıldığında operatör önceliği gözetmeksizin parantez içi önce değerlendirilir. İç içe parantezlerde hesaplama 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(" Bölme işlemleri\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 işlemleri
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;  
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()
```

```
{
```

```
    int taban, yukseklik;
```

```
    float alan;
```

```
    printf("Ucgenin Tabani = ");
```

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

```
    printf("Ucgenin Yuksekligi = ");
```

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

```
    alan = (taban*yukseklik) / 2.0;
```

```
    printf("\n Ucgenin Alani = %.2f dir \n ", alan);
```

```
}
```

C:\ Console program output

Ucgenin Tabani = 10

Ucgenin Yuksekligi = 5

Ucgenin Alani = 25.00 dir

Press any key to continue...

alan = (float) ((taban*yukseklik) / 2); //şeklinde de yazılabilir.

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

```
/*
 * Bir dairenin alanini ve cevresini hesaplar.
 */

#include<stdio.h>
#define PI 3.14159

main()
{
    double yaricap; /*giris : daire yaricapi*/
    double alan;    /*cikis : daire alani*/
    double cevre;   /*cikis : daire cevresi*/

    /*yaricapı al*/
    printf("Daire yarıcapını giriniz :");
    scanf("%lf", &yaricap);

    /*Daire alanını hesapla*/
    alan = PI*yaricap*yaricap;

    /*Daire çevresini hesapla*/
    cevre = 2*PI*yaricap;

    /*Sonuçları görüntüle*/
    printf("Daire Alanı = %.4f\n", alan);
    printf("Daire Çevresi = %.4f\n", cevre);
}
```

```
C:\ Seç c:\documents and settings\ali\belgeler
Daire yarıcapını giriniz :5.0
Daire Alanı = 78.5397
Daire Çevresi = 31.4159
```

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 : Karakök 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));
}
```

Console program output

```
2.000 nin 4.000 .kuvveti = 16.0000
Press any key to continue...
```

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);
}
```

Console program output

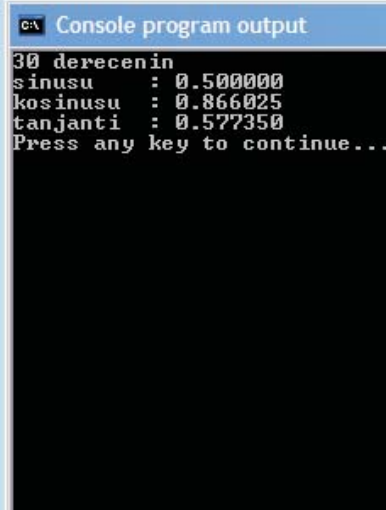
```
x = 2
y=7.000000
Press any key to continue...
```

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. $\log_{10}()$ 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. $\text{floor}()$ 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. $\text{ceil}()$ 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. $\text{fabs}()$ Fonksiyonu : x 'in mutlak değerini bulur.
Kullanımı : ***double fabs(double x);***
12. $\text{abs}()$ Fonksiyonu : Tamsayı x 'in mutlak değerini bulur.
Kullanımı : ***int abs(int x);***

//Temel çıkış fonksiyonları örnek

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

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.

```
...
int x = 1, y = 2, s, u, z;

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

printf("%d\t%d\t%d", s, u, z);
...
```

çıktısı:
1 0 1
şeklinde olur. Bunun nedeni:

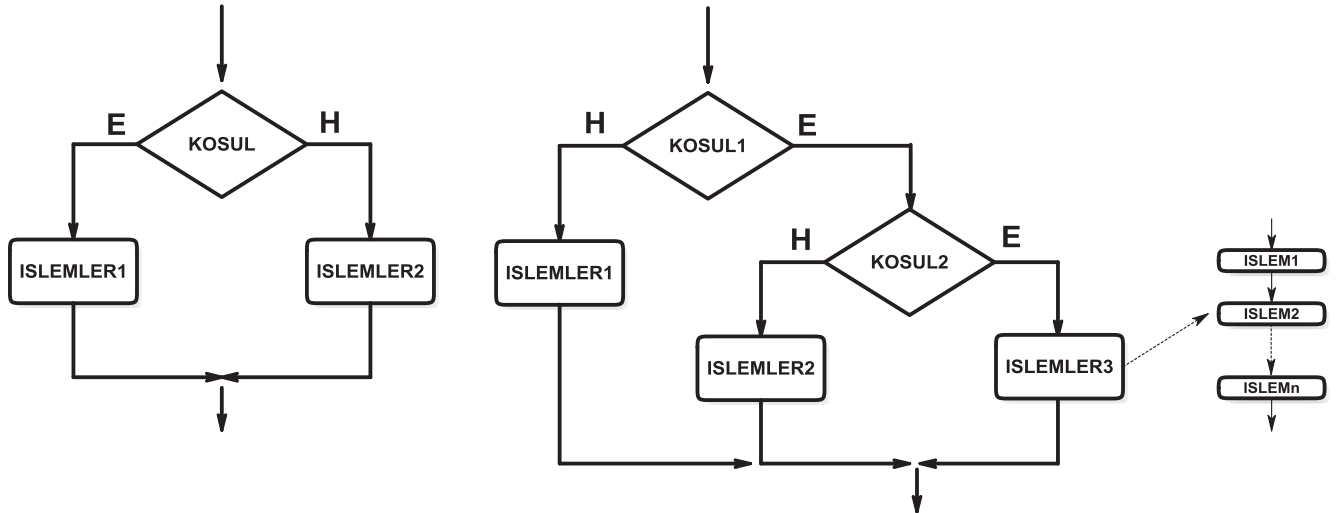
- 2 her zaman 1 den büyük olduğu için s değişkenine 1,
- $x = 1 < 3$ olduğu için x değişkenine 0,

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

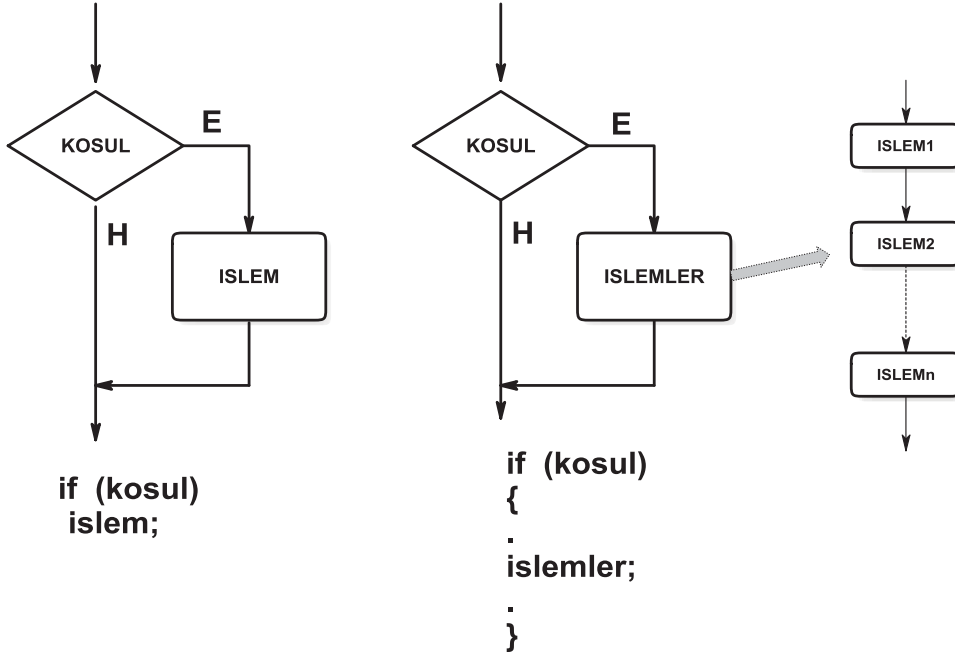
Şartlı Akış

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

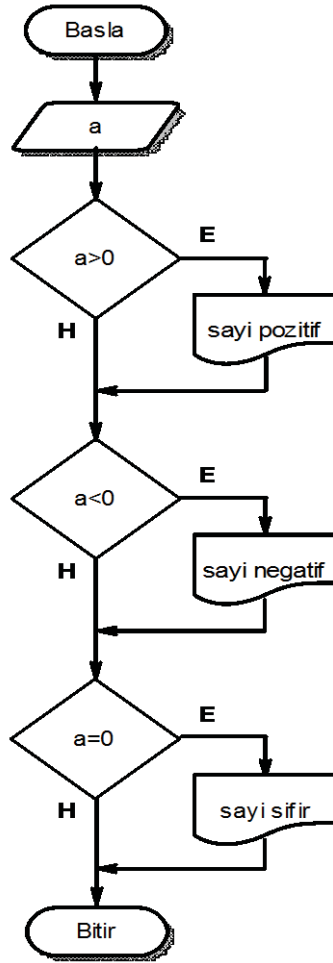


if deyimi kullanılırken kümenin başlangıcı ve bitişini gösteren, küme parantezleri kullanılmasında kullanıcıya bir esneklik sunulmuştur. Eğer if deyiminden sonra icra edilecek işlemler tek satırdan oluşuyorsa, bu işaretlerin kullanılması zorunlu değildir. Yani, if deyimden sonra { ve } işaretleri kullanılmamışsa, 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;
```

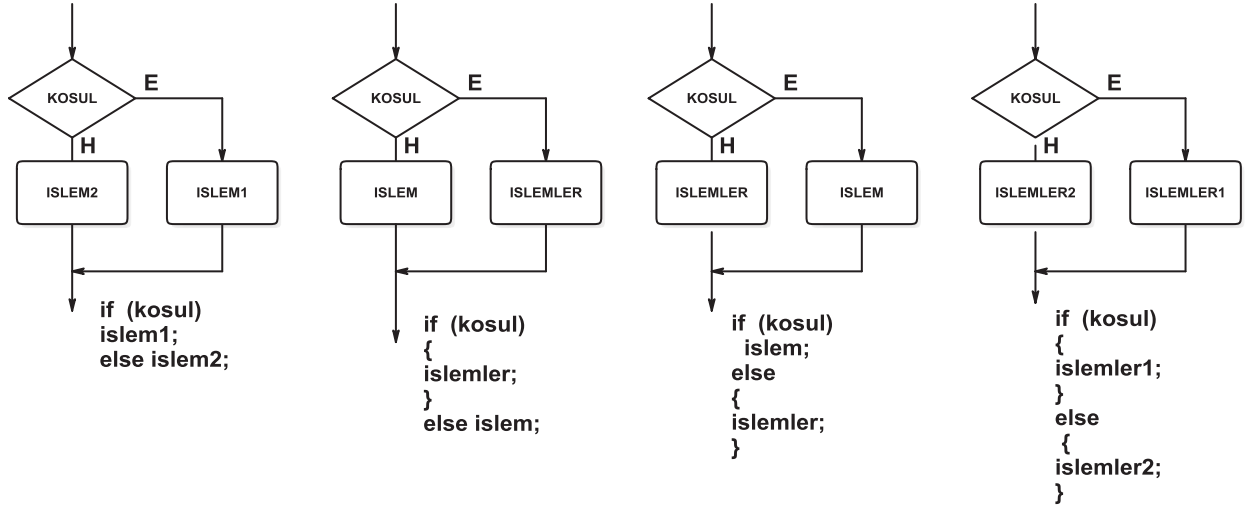
```
printf("Bir tamsayi giriniz:");
```

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

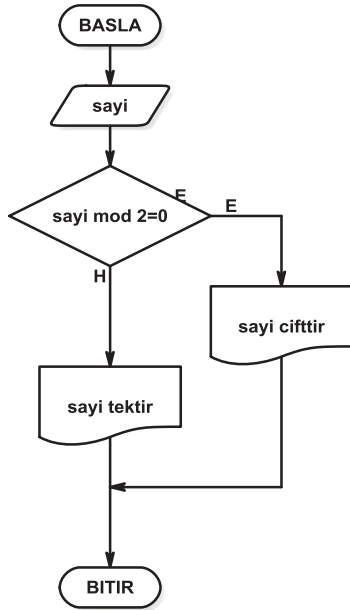
```
if (a>0)
```

```
printf("Girdiginiz %d sayisi pozitif",a);
```

if 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);
```

Mantıksal Operatörler kullanarak birden çok karşılaştırma birleştirilebilir.

(&& mantıksal VE
|| mantıksal VEYA)

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

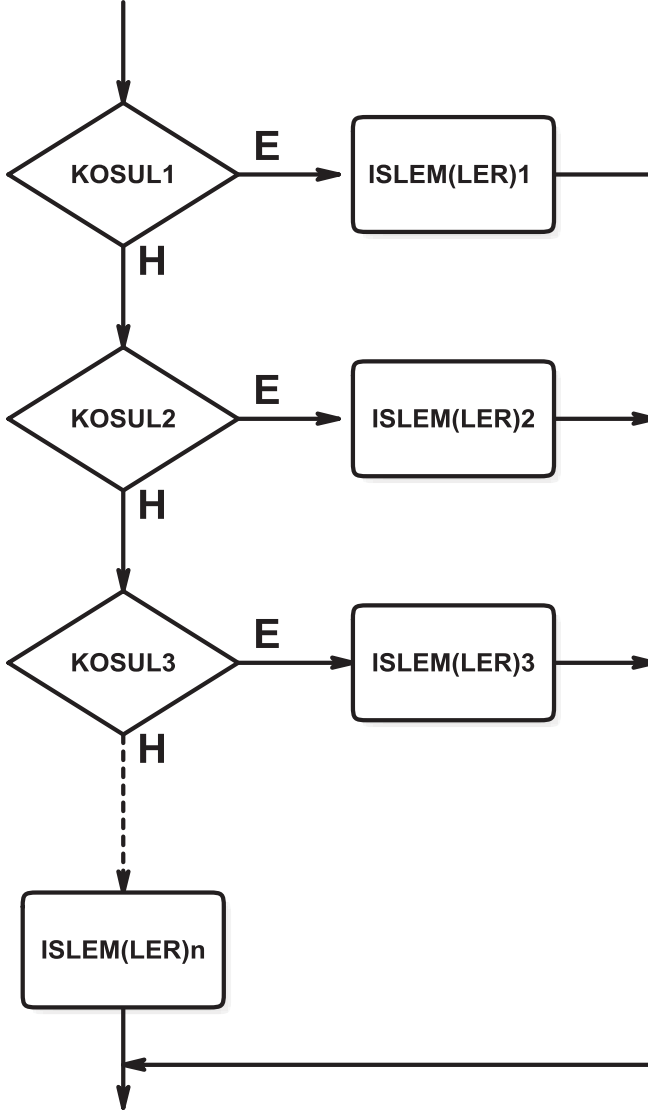
```
#include <stdio.h>
main()
{
    int a, b, c;

    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");
}
```


if else if else Yapısı

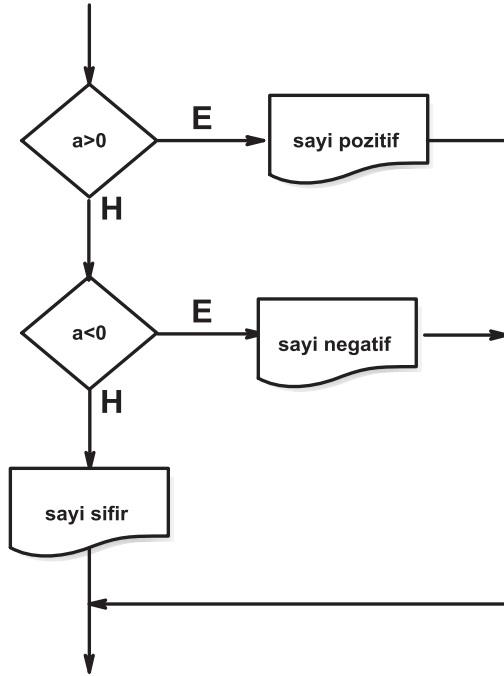
Eğer program içinde kullanılacak koşulların sayısı ikiden çok ise aşağıdaki yapı kullanılır:



```
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");
}
```

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");
}
```

Örnek: $ax^2 + bx + c = 0$ formundaki ikinci dereceden bir polinomun köklerini hesaplamaktadır. Programda delta değerinin sıfırdan küçük olması durumunda köklerin karmaşık sayıya dönüşeceğide 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 eşit:");
        printf("\nx1 = x2 = %f\n",x);
    }
}
```

```
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 *değişken*in içeriğine bakarak, programın akışını bir çok seçenektan birine yönlendirir. case (durum) deyiminden sonra *değişken*in 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:
        ...
        Islemlern;
        ...
    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");
    }
}
```

Lutfen bir karakter girin

a

a harfine bastiniz

b harfine bastiniz

Lutfen bir karakter girin

b

b harfine bastiniz

Lutfen bir karakter girin

k

a veya b ye basmadiniz

Lütfen 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 bastınız\n");
            break;
        case 'b':
            printf("b harfine bastınız\n");
            break;
        default:
            printf("a veya b ye basmadınız\n");
            break;
    }
}
```

ÇIKTI

```
Lutfen bir karakter girin
a
a harfine bastınız
```

ÇIKTI

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

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");
    }
}
```

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");
    }
}
```

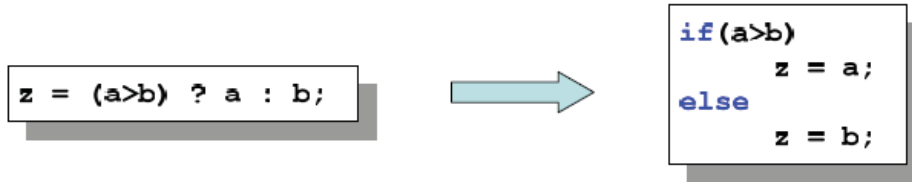
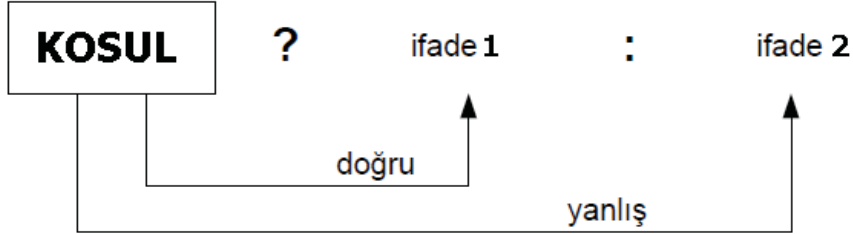
switch-case yapısı if else if else yapısının bir alternatifidir.

<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 sıranır. Eğer koşul olumluysa deyim1 aksi takdirde deyim2 değerlendirilir. deyim1 ve deyim2 de atama işlemi yapılamaz. Ancak koşul deyiminde atama işlemi yapılabilir. deyim1 ve deyim2 yerine fonksiyon da kullanılabilir.

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

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);
}
```

ÇIKTI

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

ÇIKTI

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

12. satırdaki if Yani:

if(y) // deyimindeki koşul biraz farklıdır. Genel olarak koşul bu şekilde bildirilirse, koşulun 0 dan farklı olup olmadığı sınanır.

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

Bu kullanım çok yaygındır. Eğer y, 0 dan farklı ise koşul olumlu olarak değerlendirilecektir. ? operatoru ile bir sınıma yapılmaktadır. Eğer y, x den büyük ise z değişkenine x/y, aksi takdirde x*y değeri atanmaktadır. Eğer y = 0 ise z değişkenine 0 değeri atanmaktadır.

//scanf kullanımı

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

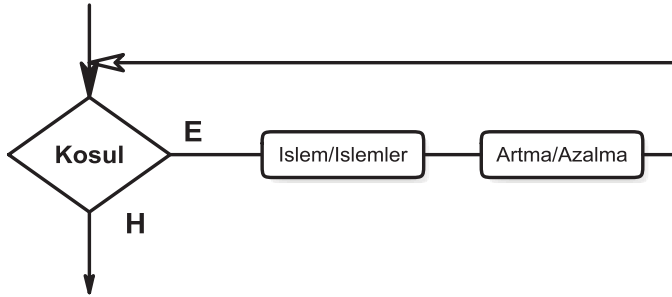
While

do .. while

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

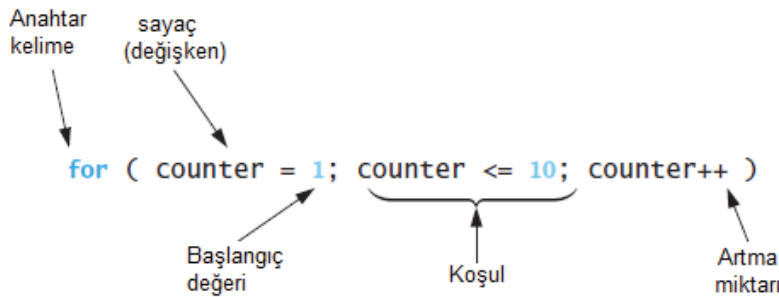
for Döngüsü

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 dersi %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>

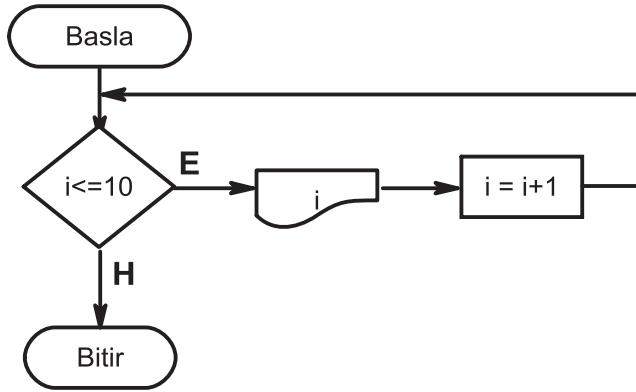
main()
{

    int i;
    char k;

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

    getchar();
}
```

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);

    getchar();
}
```

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);

    getchar();
}
```

```
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("Faktoriyeli hesaplanacak sayi :");
    scanf("%d", &n);

    faktor = 1;

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

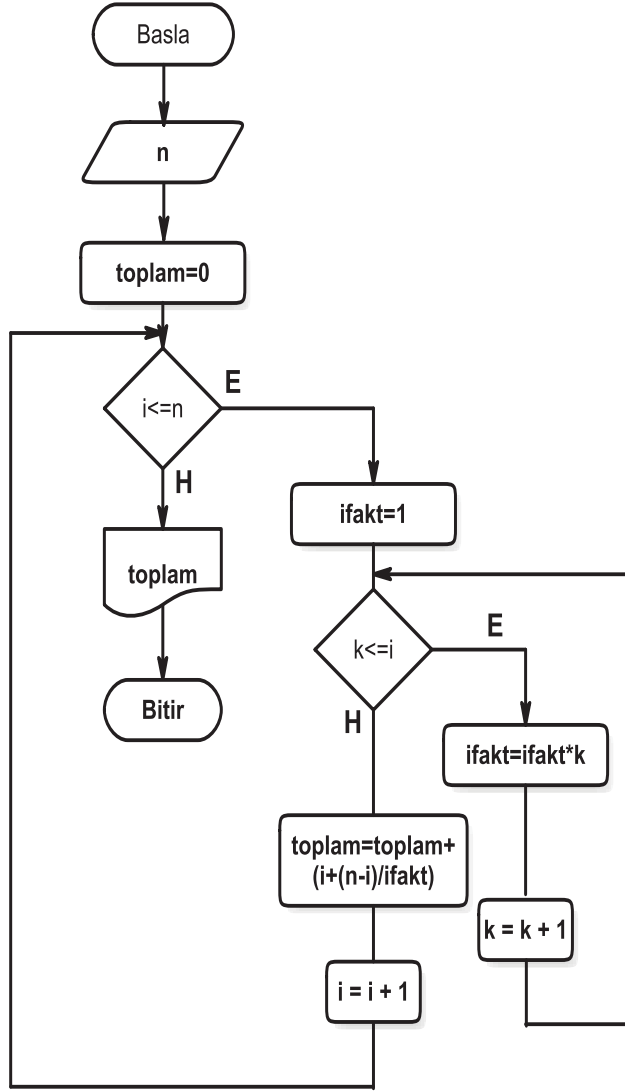
    printf("%d! = %lu\n",n, faktor);

    getchar();
}
```

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

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

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



```
#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);

    getch();
}
```

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.

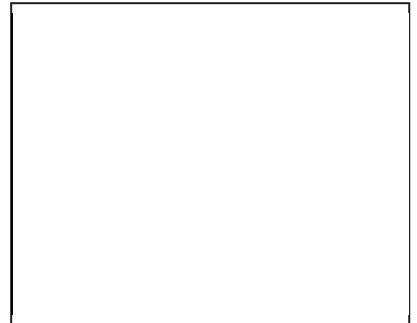
```
x = 30;  
for (; x<50 ; x+=2)  
printf ("%d", x);
```

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>  
int 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();  
    return 0;  
}
```



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 sayıları 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
```

Alt programlar (Fonksiyonlar)

C Programlama Dili fonksiyon olarak adlandırılan alt programların birleştirilmesi kavramına dayanır. Fonksiyonlar, Java veya C# gibi dillerde metot (method) ismini alırlar. Adı ne olursa olsun, görevi aynıdır. Bir işlemi birden çok yaptığınızı 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.

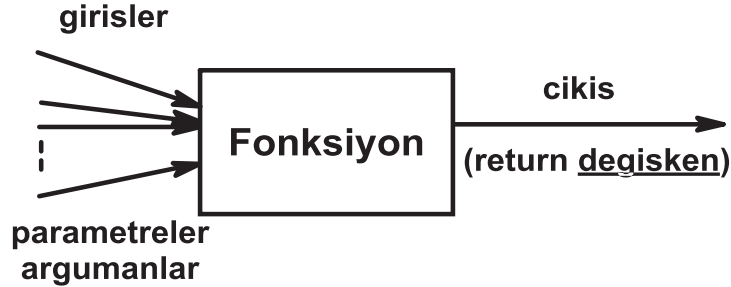
Fonksiyonlar, çalışmayı kolaylaştırır. Diskten veri okuyup, işleyen; ardından kullanıcıya gösterilmek üzere sonuçları grafik hâline dönüştüren; ve işlem sonucunu diske yazan bir programı baştan aşağı yazarsanız, okuması çok güç olur. Yorum koyarak kodun anlaşılabilirliğini, artırabilirsiniz. Ancak yine de yeterli değildir. İzlenecek en iyi yöntem, programı fonksiyon parçalarına bölmektir. Örneğin, diskten okuma işlemini *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, *çıktı* veya *geri dönüş değeri* (return değişken) olarak adlandırılır. Bir fonksiyonun kaç girişi olursa olsun sadece bir çıkışı vardır.

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

FonksiyonTipi FonksiyonAdı(parametre tipleri ve isimleri)

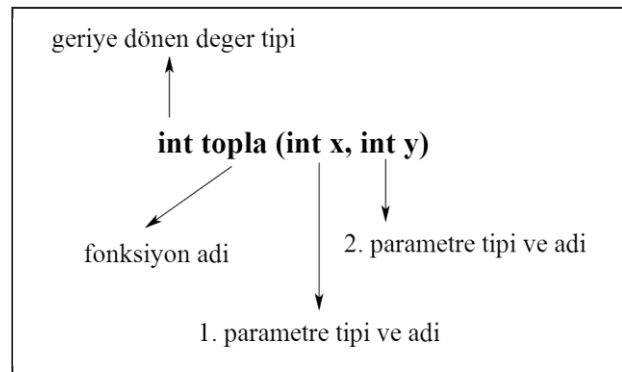
```
{  
  Yerel değişkenler  
  ...  
  İşlemler  
  ...  
  return geri_dönüş_değeri;  
}
```

Örneğin iki sayının toplamını hesaplayacak bir fonksiyon şöyle tanımlanabilir:

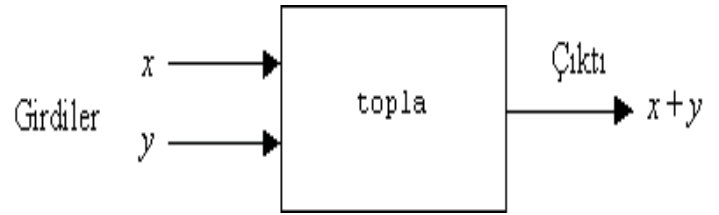
```
int topla(int x,int y)  
{  
    int sonuc;  
    sonuc = x + y;  
    return sonuc;  
}
```

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);

    getchar();
}

/** 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. Burara 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 deyiminden 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 simdiye kadar hicbir sayiya 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 değerinin olması gerekmez. Bu durumda return deyimi kullanılmayabilir. Eğer bu anahtar kelime yoksa, fonksiyon ana bloğu bitince kendiliğinden sonlanır. Böyle fonksiyonların tipi void (boş/geçersiz) olarak belirtilmelidir. Bu tip fonksiyonlar başka bir yerde kullanılırken, herhangi bir değişkene atanması söz konusu değildir, çünkü geri dönüş değeri 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 deyim kullanılması zorunlu değildir. Ancak bu deyim okunabilirliği arttırmaktadır. Örneğin:

<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 tanımladığınız 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 yapacağınız 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 (".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.

Esnek Argümanlı Fonksiyonlar

Aşağıdaki gibi üçüncü dereceden bir polinom düşünelim:

$$P(x) = a + bx + cx^2 + dx^3$$

burada a , b , c , d katsayıları gerçel sayı sabitleridir. Bu polinomu temsil eden en basit fonksiyon şöyle tanımlanabilir.

```
double p(double x, double a, double b, double c, double d)
{
    double p = a + b*x + c*x*x + d*x*x*x;
    return p;
}
```

Buna göre, $x = 1.7$ için, $P(x) = 1 - 2x$ değerini hesaplamak için bu fonksiyon aşağıdaki gibi çağırılmalıdır:

```
sonuc = p(1.7, 1.0, -2.0, 0.0, 0.0);
```

Burada, kullanılmayan katsayılar için 0.0 değeri mutlaka fonksiyona gönderilmelidir. Kullanılmayan parametreler gönderilmeden de fonksiyonu çağırmak mümkündür. C Programlama Dili, kullanıcılarına argümanları *esnek* olarak geçirme imkanı verir. Bunun anlamı, belli kurallar sağlandığında, `p()` fonksiyonu aşağıdaki gibi çağırılabilir:

```
/* x n a b */
sonuc = p(1.7, 2, 1.0, -2.0);
```

Esnek argümanlar için iki temel kural vardır:

- Esnek argümanlar kullanımı isteğe bağlıdır.
- Esnek argümanları oluşturan küme ardışık olarak listeye eklenmelidir.

Bu türden argümanlar, aşağıdaki gibi, fonksiyonun parametre listesi kısmında ... ile belirtilir.

```
double p(double x, int n, ...)
{
    ...
}
```

Esnek Argümanlı Fonksiyon tanımlaması yapabilmek için [stdarg.h](#) kütüphanesinde üç tane makro fonksiyon tanımlanmıştır. Bu fonksiyonlar Tablo 'da listelenmiştir.

stdarg.h'te tanımlı tip ve makro fonksiyonlar

İsim	İşlev	Tanımlama	Örnek
va_list	ardışık esnek argümanlar için tip belirleyici	typedef char* va_list	va_list ap;
va_start	va_list tipinde bildirilmiş gösterici için bellekten n elemanlı yer ayırır.	void va_start(va_list ap, parmN)	va_start(ap, n);
va_arg	Döngü içinde kullanılarak bütün argümanların sırasıyla alınmasını sağlar	va_arg(va_list ap, type)	va_arg(ap, int);
va_end	va_list tipinde bildirilmiş gösterici için bellekten bölgeyi boşaltır.	void va_end(va_list ap)	va_end(ap);

Bu kurallar ışığında, `p()` fonksiyonunun genel kullanımı alttaki Program 'da gösterilmiştir. `p()`, kendisine parametre olarak gelen x , n ve a_i katsayılarına göre

$$\text{Örnek : } P(x, n) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$$

polinomu hesaplar. a_i ($i = 0, 1, 2, \dots, n$) katsayıları esnek argüman olarak bildirilmiştir.

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

```
/* Verilen x, n ve ai katsayıları için,
P(x,n) = a0 + a1*x + a2*x^2 + ... + an*x^n polinomu hesaplar.
a0, a1, ..., an katsayıları esnek arguman olarak bildirilmiştir. */
```

```
double p(double x, int n, ...)
{
    double a, t = 0.0;
    int i;

    /* arguman göstericisi; ag va_list tipinde */
    va_list ag;

    /* ag için bellekten n adet hücre ayır */
    va_start(ag, n);

    for(i=0; i<n; i++)
    {
        /* herbir argumani sırasıyla al */
        a = va_arg(ag, double);

        /* polinomun değerini hesapla */
        t += a*pow(x,i);
    }

    /* belleği boşalt */
    va_end(ag);

    return t;
}
```

```
/*Ana program*/
int main(void){

    double x = 1.7;

    printf("x = %f için:\n", x);

    printf("p(x, 1, 1.0) = %.4f\n",
        p(x, 1, 1.0));

    printf("p(x, 2, 1.0, -2.0) = %.4f\n",
        p(x, 2, 1.0, -2.0));

    printf("p(x, 3, 1.0, -2.0, 0.2) = %.4f\n",
        p(x, 3, 1.0, -2.0, 0.2));

    printf("p(x, 4, 1.0, -2.0, 0.2, 1.1) = %.4f\n",
        p(x, 4, 1.0, -2.0, 0.2, 1.1));

    printf("p(x, 5, 1.0, -2.0, 0.2, 1.1, -0.6) = %.4f\n",
        p(x, 5, 1.0, -2.0, 0.2, 1.1, -0.6));

    return 0;
}
```

ÇIKTI

```
Console program output
x = 1.70 için:
p(x, 1, 1.0) = 1.0000
p(x, 2, 1.0, -2.0) = -2.4000
p(x, 3, 1.0, -2.0, 0.2) = -1.8220
p(x, 4, 1.0, -2.0, 0.2, 1.1) = 3.5823
p(x, 5, 1.0, -2.0, 0.2, 1.1, -0.6) = -1.4290
Press any key to continue...
```

Örnek : Esnek parametrelili fonksiyonla n tane sayının toplamını hesaplayan program.

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

/* Esnek argumanla tanımlanmış n tane tamsayının sayının
   toplamını gönderir */
int topla(int n, ...)
{
    va_list ap;
    int i, top = 0;

    va_start(ap, n);

    for (i=1; i<=n; i++)
        top += va_arg(ap, int);

    va_end(ap);
    return top;
}

main()
{
    printf("topla(2, 11,22)           = %d\n", topla(2, 11,22));
    printf("topla(3, 11,22,33)        = %d\n", topla(3, 11,22,33));
    printf("topla(4, 11,22,33,44)      = %d\n", topla(4, 11,22,33,44));
    printf("topla(5, 11,22,33,44,55)   = %d\n", topla(5, 11,22,33,44,55));
    printf("topla(6, 11,22,33,44,55,66) = %d\n", topla(6, 11,22,33,44,55,66));

    getchar();
}
```

ÇIKTI

```
Console program output
topla(2, 11,22)           = 33
topla(3, 11,22,33)        = 66
topla(4, 11,22,33,44)      = 110
topla(5, 11,22,33,44,55)   = 165
topla(6, 11,22,33,44,55,66) = 242
Press any key to continue...
```

Argüman sayısı bildirilmeden de bir küme üzerinde işlem yapılabilir. Ancak bu durumda kümenin sonunu gösteren bir sayı kümeye eklenir. Alttaki örnekte, argümanları esnek olarak bildirilmiş `argyaz(int arg, ...)` fonksiyonu, son elemanı -1 olan bir kümenin elemanlarını ekrana yazar. Kümenin sonu -1 ile belirlenmiş olur.

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

/* herbiri int tipinde ve sonu -1 ile biten kümeyi ekrana yazar */
void argyaz(int arg, ...)
{
    va_list ap;
    int i;

    va_start(ap, arg);

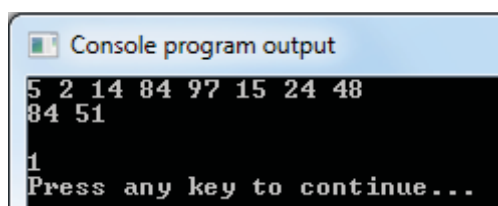
    for (i = arg; i != -1; i = va_arg(ap, int))
        printf("%d ", i);

    va_end(ap);
    putchar('\n');
}

main(void)
{
    argyaz(5, 2, 14, 84, 97, 15, 24, 48, -1);
    argyaz(84, 51, -1);
    argyaz(-1);
    argyaz(1, -1);

    getchar();
}
```

ÇIKTI



```
Console program output
5 2 14 84 97 15 24 48
84 51

Press any key to continue...
```

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 değerlendir. 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** öniflemcisi 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
hipo(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 sayi girin: ");
    scanf("%d %d",&x,&y);

    eb = büyük(x,y);

    printf("buyuk olan %d\n",eb);

    getchar();
}
```

ÇIKTI

iki sayı girin: 8 6

buyuk 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 değerdendirilir. Örneğın 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 göreacaktır. 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 gördeli yavaşlıktan kaynaklanır. Fonkiyon 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:

Diziler

Bir programda, birbirine benzer, eş değişkenlerin kullanımı söz konusu olabilir.

Örnek : Bir dersi alan öğrencilerin arasından aldıkları notları saklamak için kullanılan değişkenler : Her öğrenci notu için bir değişken.

not1, not2, not3,, notN

-Her veri için bir değişken tanımlamak yerine bunların tümünü bir dizi ile tanımlamak mümkündür. Dizi, benzer tipteki değişkenlerin isimlendirilmiş bir topluluğudur.

-Bu toplulukta her değişken bir elemandır.

-Dizi içerisinde elemanlar, tamsayı pozisyon numaraları veya indisler ile belirlenirler.

Dizi Bildirimi

-Normal değişkenlerde olduğu gibi bir dizi de kullanılmadan önce bildirimi yapılmalıdır.

Tanımlama ;

dizi_tipi dizi_ismi [eleman_sayısı] ;

dizi_tipi	Dizinin içerdiği değerlerin veri türü. Aynen değişken türlerinin tanımlandığı biçimde kullanılır.
dizi_ismi	Dizinin mutlaka bir adı olmalıdır. Program içinde dizinin tüm elemanları bu ortak isim ile temsil edilir.
eleman_sayisi	Dizinin elemanları için bellekte ayrılacak yeri belirler. Ayrılan yerin tümüyle dolması gerekmez. Örneğin 10 elemanlık bir boyuta sahip dizinin 3 elemanı olabilir.

Örnek :

```
int notlar[5];
```

notlar					
0	1	2	3	4	

1. Dizi elemanları temsili şekilde olduğu gibi hafızada arka arkaya yer alırlar.
2. Her kutuda bir değer saklanır.
3. Her kutunun içerdiği değer aynı tipe sahiptir. Bu da dizinin tipidir
4. Dizi ismi bütün kutucukları temsil eder.
5. Dizi ismi aynı zamanda dizinin hafızadaki başlangıç adresini de temsil eder.
6. Dizi bildirimindeki eleman_sayısı dizinin kaç kutucuğa sahip olacağını belirtir.
7. eleman_sayısı, sabit bir ifade olmalıdır.

- Değişken kullanılamaz.
- Programın çalışması esnasında değeri değiştirilemez.
- #define direktifi ile değer verilebilir.

```
int n = 100;
int a[n];

veya

#define n 100
...
int a[n];
```

Dizi Elemanlarına Ulaşmak

1. Dizi içindeki her bir kutucuk (eleman) pozitif tamsayı bir numaraya sahiptir (indis). Kutucuklara (elemanlara) bu numaralar vasıtasıyla ulaşılır.
2. Derleyici yanlış indis kontrolü yapmaz.
3. Dizinin her bir elemanı normal bir değişken gibi olduğundan elemanların değerleri okunabilir veya bunlara değer yazılabilir.
3. Bir dizinin elemanına ulaşmak için :

`dizi_adı[indis]`

Örnek : sayılar[3], ch[5], notlar[0]

4. İndis numaraları;
- İlk elemanın indisi 0 dır.
- Son elemanın indisi eleman_sayisi - 1 dir.
5. İndis, hafızada elemanın dizinin başına olan uzaklığını gösterir.

Dizilere Başlangıç Değeri Verme

Bir diziye başlangıç değerleri aşağıdaki gibi kısa formda atanabilir:

```
float kutele[5]= { 8.471, 3.683, 9.107, 4.739, 3.918 };
int maliyet[3] = { 25, 72, 94 };
double a[4] = { 10.0, 5.2, 7.5, 0.0};
```

Küme parantezlerinin sonlandırıcı ; karakteri ile biter.

Bir dizinin uzunluğu belirtilmeden de başlangıç değeri atamak mümkündür.

```
int a[] = { 100, 200, 300, 400 };
float v[] = { 9.8, 11.0, 7.5, 0.0, 12.5};
```

Derleyici bu şekilde bir atama ile karşılaştığında, küme parantezi içindeki eleman sayısını hesaplar ve dizinin o uzunlukta açıldığını varsayar. Yukarıdaki örnekte, a dizisinin 4, v dizisinin 5 elemanlı olduğu varsayılır.

Dizileri Yazdırma/Okuma

`printf` ve `scanf` fonksiyonları bir dizinin okunması ve yazdırılması için de kullanılır. Örneğin bir `A` dizisinin aşağıdaki gibi bildirildiğini varsayalım:

```
int A[10];
```

Bu dizinin elemanlarını klavyeden okumak için `for` döngüsü ile:

```
for(i=0; i<10; i++)  
    scanf("%d", &A[i]);
```

Herhangibir elemanına değer yüklemek için:

```
A[3]=5;  
A[i+j]=10;
```

daha sonra bu değerlerini ekrana yazmak için:

```
for(i=0; i<10; i++)  
    printf("%d\n", A[i]);
```

Örnek : Klavyeden girilen 10 adet sayının ortalamasını hesaplayıp, kaç adet elemanın ortalamanın altında kaldığı ve kaç adet elemanın ortalamanın üstünde olduğu bulan program.

```
#include<stdio.h>
main()
{
    float dizi[ 10 ];
    float ortalama, toplam = 0;
    int ortalama_ustu_adedi = 0;
    int ortalama_alti_adedi = 0;
    int i;

    for( i = 0; i < 10; i++ ) {
        printf( "%d. elemanı giriniz> ", (i+1) );
        scanf( "%f", &dizi[ i ] );
        toplam += dizi[ i ];
    }

    ortalama = toplam / 10.0;

    for( i = 0; i < 10; i++ ) {
        if( dizi[ i ] < ortalama )
            ortalama_alti_adedi++;
        else if( dizi[ i ] > ortalama )
            ortalama_ustu_adedi++;
    }

    printf( "Ortalama: %.2f\n", ortalama );
    printf( "Ortalamadan düşük %d eleman vardır.\n", ortalama_alti_adedi );
    printf( "Ortalamadan yüksek %d eleman vardır.\n", ortalama_ustu_adedi );

}
```

Örnek : Elemanları belli olan 10 elemanlı bir dizinin en büyük ve en küçük elemanını ve bunların dizinin kaçınca elemanı olduğunu bulan program.

```
#include<stdio.h>
main()
{
    // dizi'yi tanitirken, ilk deger
    // atiyoruz
    int dizi[ ] = { 15, 54, 1, 44, 55, 40, 60, 4, 77, 45 };
    int i, max, min, siramin, siramax;

    // Dizinin ilk elemanini, en kucuk ve en buyuk deger kabul ediyoruz.

    min = dizi[ 0 ]; siramin=0;
    max = dizi[ 0 ]; siramax=0;

    for( i = 1; i < 10; i++ ) {

        if( min > dizi[i] )
        {
            min = dizi[i];
            siramin=i;
        }

        if( max < dizi[i] )
        {
            max = dizi[i];
            siramax=i;
        }

    }

    printf( "En kucuk deger %d ve dizinin %d. elemanidir\n", min, siramin );
    printf( "En buyuk deger %d ve dizinin %d. elemanidir\n", max, siramax );

}
```


Dizilerin Fonksiyona Aktarılması

C dilinde fonksiyon kullanırken fonksiyona parametre aktarma işlemini görmüştük. Dizili değişkenlerin fonksiyona aktarılması adres aktarma(by reference) yöntemi kullanılarak yapılmaktadır.

Bu yöntemdeki temel amaç çağırıcı ve çağırılan fonksiyonun aynı veriyi ortak olarak kullanması demektir. Böylece çağırıcı programca fonksiyona gönderilen dizinin bir elemanı verileri fonksiyon içerisinde işlem gördükten sonra çağırıcı programa geri döndüklerinde değişmiş olan değerleri üzerinden işlem görürler. Fonksiyonun çağırılması sırasında dizili değişkenin adı parametre listesinde yer alır. Ayrıca kaç elemanın kullanılacağını da parametre olarak bildirilmesi gerekebilir.

1. Belirli dizi elemanlarının altprograma aktarılması :

Dizi elemanları normal değişkenler gibi olduğundan bir fonksiyona dizinin bir elemanı normal değişkenler gibi geçirilebilir.

```
int toplam(int, int); //Fonksiyon prototipi
void main(void)
{
    int dizi[] = {1,4,2,6,8,4};
    int topla = toplam(dizi[3], dizi[4]);
    .....
}

int toplam(int a, int b) //Fonksiyon tanımı
{
    return a+b;
}
```

2. Dizi elemanlarının hepsinin aktarılması :

2.1. Global sabit kullanarak dizinin aktarılması ;

```
#include<stdio.h>

#define n 10

int bul ( int dizi[] );
//veya int bul ( int []);

main()
{
    int not[n] = { 23, 56, 9, 78, 96, 55, 34, 21, 7, 8 };
    int enkucuk;

    enkucuk=bul(not); //fonksiyon cagriliyor

    printf( "Dizinin en kucuk elemani = %d\n", enkucuk );

}

int bul ( int dizi[] )
{
    int i, ek;
    ek = dizi[0];
    for (i=1; i<n; i++)
        if (dizi[i] < ek)
            ek = dizi[i];
    return ek;
}
```

2.2. Eleman sayısı ile dizinin aktarılması

```
#include<stdio.h>
void elemanlari_goster( int [ 5 ] );
main()
{
    int dizi[ 5 ] = { 55, 414, 7, 210, 15 };
    elemanlari_goster( dizi );

}
void elemanlari_goster( int gosterilecek_dizi[ 5 ] )
{
    int i;
    for( i = 0; i < 5; i++)
        printf( "%d\n", gosterilecek_dizi[ i ] );
}
```

Yukarıdaki altprogram sadece 5 boyutlu bir dizi için çalışır. Farklı boyutlarda bu diziyi çalıştırmamız gerekebilir.

2.3. Değişken boyutlarda eleman sayıları ile dizinin aktarılması

Bu yapıda dizi, çağrılan fonksiyona gönderilirken adı ile birlikte boyutu parametre tanımlama kısmında belirtilir.

```
#include<stdio.h>
void elemanlari_goster( int [ ], int );

main()
{
    int dizi[ 5 ] = { 55, 414, 7, 210, 15 };
    elemanlari_goster( dizi, 5 );
}

void elemanlari_goster( int gosterilecek_dizi[ ], int eleman_sayisi )
{
    int i;
    for( i = 0; i < eleman_sayisi; i++)
        printf( "%d\n", gosterilecek_dizi[ i ] );
}
```

elemanlari_goster isimli altprogram farklı boyutlu diziler içinde çalışır.

Örnek : 8 elemanlı bir dizinin en büyük, en küçük elemanını ve ortalamasını bulan program.

```
#include<stdio.h>
float maksimum_bul( float [ ], int );
float minimum_bul( float [ ], int );
float ortalama_bul( float [ ], int );

main()
{
    float sayilar[ 8 ] = { 12.36, 4.715, 6.41, 13, 1.414, 1.732, 2.236, 2.645 };
    float max, min, ortalama;

    max = maksimum_bul( sayilar, 8 );
    min = minimum_bul( sayilar, 8 );
    ortalama = ortalama_bul( sayilar, 8 );
    printf( "En buyuk: %.2f\n", max );
    printf( "En kucuk: %.2f\n", min );
    printf( "Ortalama: %.2f\n", ortalama );
}

float maksimum_bul( float dizi[ ], int eleman_sayisi )
{
    int i;
    float max;
    max = dizi[0];
    for( i = 1; i < eleman_sayisi; i++ ) {
        if( max < dizi[ i ] )
            max = dizi[ i ];
    }
    return max;
}

float minimum_bul( float dizi[ ], int eleman_sayisi )
{
    int i, min;
    min = dizi[ 0 ];
    for( i = 1; i < eleman_sayisi; i++ ) {
        if( min > dizi[ i ] ) {
            min = dizi[ i ];
        }
    }
    return min;
}

float ortalama_bul( float dizi[ ], int eleman_sayisi )
{
    int i, ortalama = 0;
    for( i = 0; i < eleman_sayisi; i++ )
        ortalama += dizi[ i ];

    return ortalama / 8.0;
}
```

Çok Boyutlu Diziler

Bir dizi aşağıdaki gibi bildirildiğinde bir boyutlu (tek indisli) dizi olarak adlandırılır. Bu tip dizilere *vektör* denir.

```
float a[9];
```

Bir dizi birden çok boyuta sahip olabilir. Örneğin iki boyutlu b dizisi şöyle tanımlanabilir:

```
float b[9][4];
```

İki boyutlu diziler *matris* olarak adlandırılır. ilk boyuta *satır*, ikinci boyuta *sütun* denir. Yukarıda b matrisinin eleman sayısı $9 \times 4 = 36$ dır.

```
int sayi[3][4];
```

sayi[0][0]	sayi[0][1]	sayi[0][2]	sayi[0][3]
sayi[1][0]	sayi[1][1]	sayi[1][2]	sayi[1][3]
sayi[2][0]	sayi[2][1]	sayi[2][2]	sayi[2][3]

genel olarak diziler şöyle gösterilir:

Dizlerin Bildirimi

Dizi Çeşiti	Genel Bildirimi	Örnek
Tek boyutlu diziler (Vektörler)	<i>tip dizi_adı[eleman_sayısı]</i>	int veri[10];
İki boyutlu diziler (Matrisler)	<i>tip dizi_adı[satır_sayısı][sütun_sayısı]</i>	float mat[5][4];
Çok boyutlu diziler	<i>tip dizi_adı[boyut_1][boyut_2]...[boyut_n];</i>	double x[2][4][2];

Çok Boyutlu Dizilere İlk Değer Atama

Çok boyutlu bir diziyi tanımlarken, eleman değerlerini atamak mümkündür. Çok boyutlu dizilerde ilk değer atama, tek boyutlu dizilerdekiyle aynıdır. Girdiğiniz değerler sırasıyla hücrelere atanır. Bilgisayar, çok boyutlu dizi elemanlarını hafızada arka arkaya gelen bellek hücreleri olarak değerlendirir.

```
int tablo[3][4] = { 8, 16, 9, 52, 3, 15, 27, 6, 14, 25, 2, 10 };
```

Diziyi tanımlarken, yukardaki gibi bir ilk değer atama yaparsanız, elemanların değeri aşağıdaki gibi olur:

Satır 0	:	8	16	9	52
Satır 1	:	3	15	27	6
Satır 2	:	14	25	2	10

Çok boyutlu dizilerde ilk değer atama yaparsanız, değerleri kümelendirmek iyi bir yöntemdir; karmaşıklığı önler. Örneğin yukarıda yazmış olduğumuz ilk değer atama kodunu, aşağıdaki gibi de yazabiliriz:

```
int x[3][4] = {8,16,9,52, /* 1. satır elemanları */  
               3,15,27,6, /* 2. satır elemanları */  
               14,25,2,10}; /* 3. satır elemanları */
```

Tek boyutlu dizilerde ilk değer ataması yaparken, eleman sayısından az değer girerseniz, kalan değerler 0 olarak kabul edilir. Aynı şey çok boyutlu diziler için de geçerlidir; olması gerektiği sayıda eleman ya da grup girilmezse, bu değerlerin hepsi 0 olarak kabul edilir.

İki boyutlu dizilerin bütün elemanları ile işlem yapmak için iç-içe 2 for döngüsü kullanılır.

```
for(i=0; i<3; i++)  
{  
    for(j=0; j<4; j++)  
        printf("%4d",x[i][j]);  
  
    printf("\n");  
}
```

çıktısı:

11	34	42	60
72	99	10	50
80	66	21	38

şeklinde olur.

Örnek : İki matrisin toplamı

```
#include <stdio.h>
#define SAT 2
#define SUT 3
main()
{
    int a[SAT][SUT] = {5, 3, 7, 0, 1, 2};

    int b[SAT][SUT] = {1, 2, 3, 4, 5, 6};
    int c[SAT][SUT];
    int i, j;

    puts("A Matrisi:");
    for(i=0; i<SAT; i++)
    {
        for(j=0; j<SUT; j++)
            printf("%4d",a[i][j]);
        printf("\n");
    }

    puts("B Matrisi:");
    for(i=0; i<SAT; i++)
    {
        for(j=0; j<SUT; j++)
            printf("%4d",b[i][j]);
        printf("\n");
    }

    puts("\nC Matrisi:");
    for(i=0; i<SAT; i++)
    {
        for(j=0; j<SUT; j++)
        {
            c[i][j] = a[i][j] + b[i][j];
            printf("%4d",c[i][j]);
        }
        printf("\n");
    }
}
```

ÇIKTI

```
A Matrisi:
 5   3   7
 0   1   2
B Matrisi:
 1   2   3
 4   5   6
C Matrisi:
 6   5  10
 4   6   8
```


Örnek : 3x3 boyutlu 2 matrisin çarpımı

```
#include <stdio.h>

#define N 3

int main()
{
    int a[N][N], b[N][N], c[N][N];
    int i,j,k,toplam;

    puts("A Matrisini girin:");
    for(i=0; i<N; i++)
        for(j=0; j<N; j++)
            scanf("%d",&a[i][j]);

    puts("B Matrisini girin:");
    for(i=0; i<N; i++)
        for(j=0; j<N; j++)
            scanf("%d",&b[i][j]);

    puts("\nC Matrisi:");
    for(i=0; i<N; i++)
    {
        for(j=0; j<N; j++)
        {
            for(toplam=0, k=0; k<N; k++)
                toplam += a[i][k]*b[k][j];
            c[i][j] = toplam;
            printf("%4d",c[i][j]);
        }

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

ÇIKTI

```
A Matrisini girin:
5  3  7
0  1  2
9  0  4
B Matrisini girin:
1  2  3
4  5  6
7  8  9
C Matrisi:
66  81  96
18  21  24
37  50  63
```