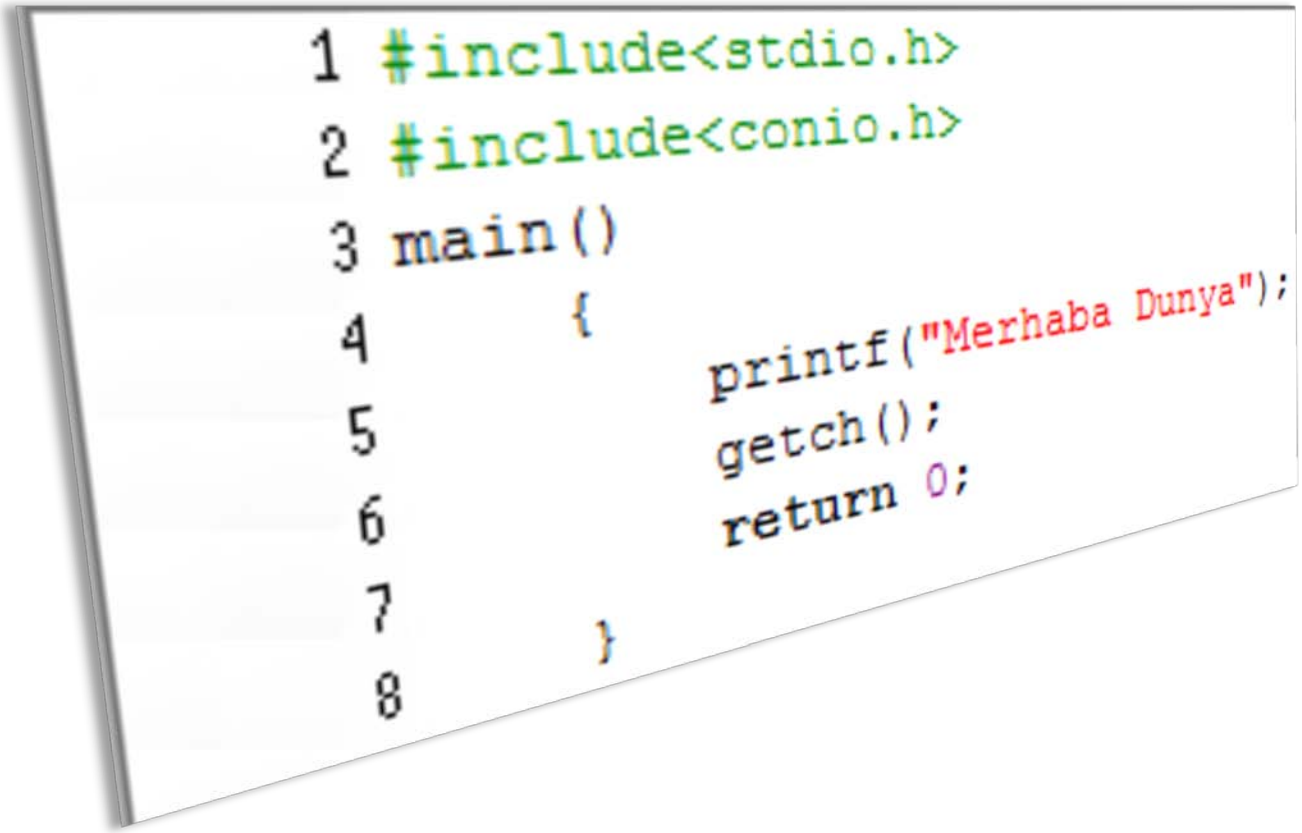
Bilgisayar Programlama



Öğr. Gör. Levent TERLEMEZ

- ✓Program, belli bir komut ve söz dizimi yapısını uygun olarak, sadece belirtilen işlemlerin yerine getirilmesini belirten komut dizisidir
- ✓Programda belirtilen işlem dizisi doğrudan işleme konmaz.
- ✓Bilgisayar, verilen programı, makine dilinde işleme almak zorundadır.

Programcı programlama dili biliyor, bilgisayar ise makine dili biliyor. Programcının, bilgisayara, ne demek istediğini kim tercüme edecek (bilgisayar ile programcı arasındaki iletişimi nasıl sağlanacak)?

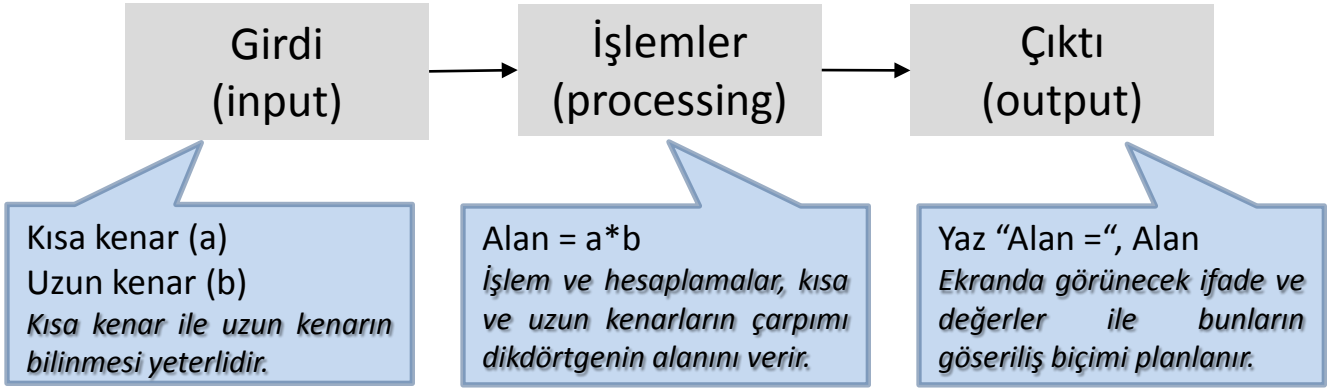
Bu tercümeyi, **derleyici** (compiler) veya **yorumlayıcı** (interpreter) olmak üzere ikiye ayrılan çeviri programları yapar. Bu sayede programlama dilleri ile yazılmış, komutlar dizisinden oluşan kaynak program, makine diline *derlenir* (compile) ve/veya *yorumlanır* (interpret).

Örneğin, makine dilinde iki sayının toplanması, **10001100111010001111010100001001010110100010** biçiminde ifade edilir.

Bir bilgisayar programı üç kısımdan oluşur:

1. Bilgi girişi
2. İşlemler
3. Bilgi çıkışı

Problem: Bir dikdörtgenin alanının hesaplanması



Programın doğru sonuç üretip üretmediğinin test edilebilmesi için, program çıktısında neyin elde edileceğinin bilinmesi gereklidir. Çıkış işleminde çıkış içeriği (elde edilen veya edilecek olan bilgi) ve formatı (elde edilen veya edilecek bilginin gösteriliş şekli) önemlidir.

Problemin Belirlenmesi

Pozitif A ve B tamsayılarının sayılarının çarpılması

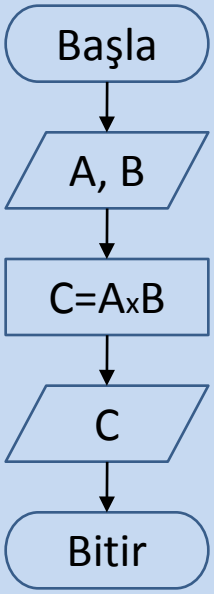
Problemin Analizi

C = A x B *veya* A kadar B topla

Çözüm Algoritması

1.Adım: Başla
2.Adım: A'yı gir.
3.Adım: B'yi gir.
4.Adım: A çarpı B'yi C'ye ata.
5.Adım: C'yi yaz.
6.Adım: Bitir.

1.Metinsel
2.Sözde Kodlar (Pseudo-Codes)
3.Akış Şemaları (Flow Chart)



Prog. Geliştirme Aşamaları

```
...
scanf("%d, %d",&a,b);
c = a * b;
printf("c = %d",c);
...
```

Bilgisayar Programı

Program fonksiyonlarının belirlenmesi, uygun görülen programlama dili ile kodlanması, programın test edilmesi, (yazımsal ve/veya mantıksal) hataların giderilmesi, kullanım kılavuzunun hazırlanması ve benzeri işlemler.

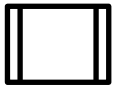
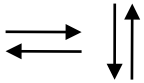
Bir problemin çözümüne yönelik hazırlanacak olan programa ilişkin sonlu sayıda ve anlaşılır adım yada işlemin ardı ardına tanımlanması ve böylece izlenecek yolun veya yöntemin ortaya konulması için hazırlanan plana **algoritma** denir.

- ✓**Algoritma** sıralı olmalıdır. Bir başlangıç noktasından başlayarak, sonlu sayıda, belirsizlik içermeyen adımdan sonra sona erebilmelidir.
- ✓**Algoritma** problemle ilgili olarak ortaya çıkabilecek her türlü durumu içermelidir.
- ✓**Algoritma** aynı tür problemler içinde geçerli olmalıdır.

Algoritma sözcüğü adını Ebu Abdullah Muhammed İbn Musa El Harezmi adındaki Müslüman – Türk aliminden almıştır. Adı Latinceye **Alkhorizma**, Fransızcaya **Algorithme**, İngilizceye ise **Augrim** şeklinde geçmiştir.

Alimin ismini telaffuz edemeyen Avrupalılar algorizm sözcüğünü "*sayıları kullanarak aritmetik problemler çözme kuralları*" anlamında kullanırlar. Bu sözcük daha sonra algoritmaya dönüşür ve yaygın olarak kullanılır. Latince çevirisinin Avrupa'da çok büyük ilgi gördüğü, **Hisab el-cebir** ve **el-mukabala** kitabı dünyanın ilk cebir kitabı ve aynı zamanda ilk algoritma koleksiyonunu oluşturur. Alimin 780 (H.164) senesinde Hazerm'de doğduğu kabul edilir. 850 (H.236) senesinde ise Bağdat'ta vefat etmiştir.

Bir problemin çözümüne yönelik hazırlanacak algoritmanın, görsel simge ve/veya sembollerle ifade edilmiş şekline **akış şeması** denir.

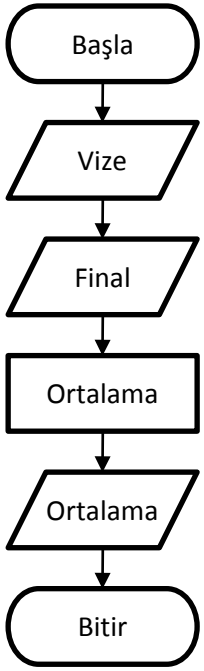
Simge	İşlev	Simge	İşlev
	Başla/Bitir		Döngü
	Genel Girdi/Çıktı		Manyetik Disk
	Genel İşlem		Yordam Çağırma
	Denetim (Karar)		Akış Yönü
	Yazıcı Çıktısı		Bağlaç
	Görüntü Çıktısı		Sayfa Bağlacı
	Ele ile Girdi (Klavye)		

Örnek: Girilen vize (ortalamaya etkisi %40) ve final notu (ortalamaya etkisi %60) göre, ders ortalaması hesaplayan bilgisayar program için algoritma ve akış diyagramı.

Algoritma

1. Adım: Başla
2. Adım: Vize notunu gir
3. Adım: Final notunu gir
4. Adım: Ortalamayı hesapla
(ortalama=vize*.40+final*.60)
5. Adım: Ortalamayı yaz
6. Adım: Bitir

Akış Diyagramı

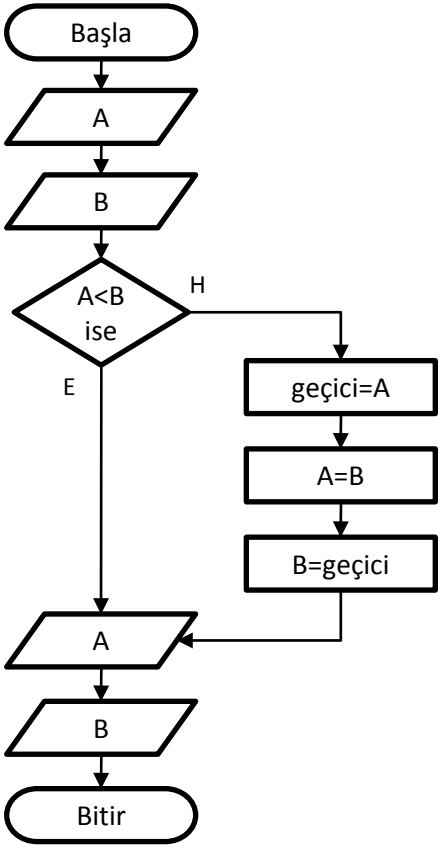


Örnek: Rastgele girilen A ve B sayılarından küçük olanı önce yazan bir program için algoritma ve akış diyagramı.

Algoritma

1. Adım: Başla.
2. Adım: A sayısını gir.
3. Adım: B sayısını gir.
4. Adım: Eğer A<B ise . 6. Adıma git.
5. Adım: A ve B'yi yer değiştir.
6. Adım: A'yı ve B'yi yaz.
7. Adım: Bitir.

Akış Diyagramı

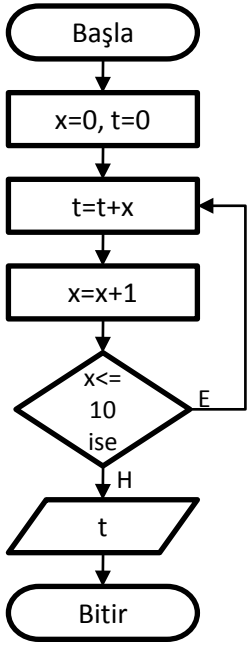


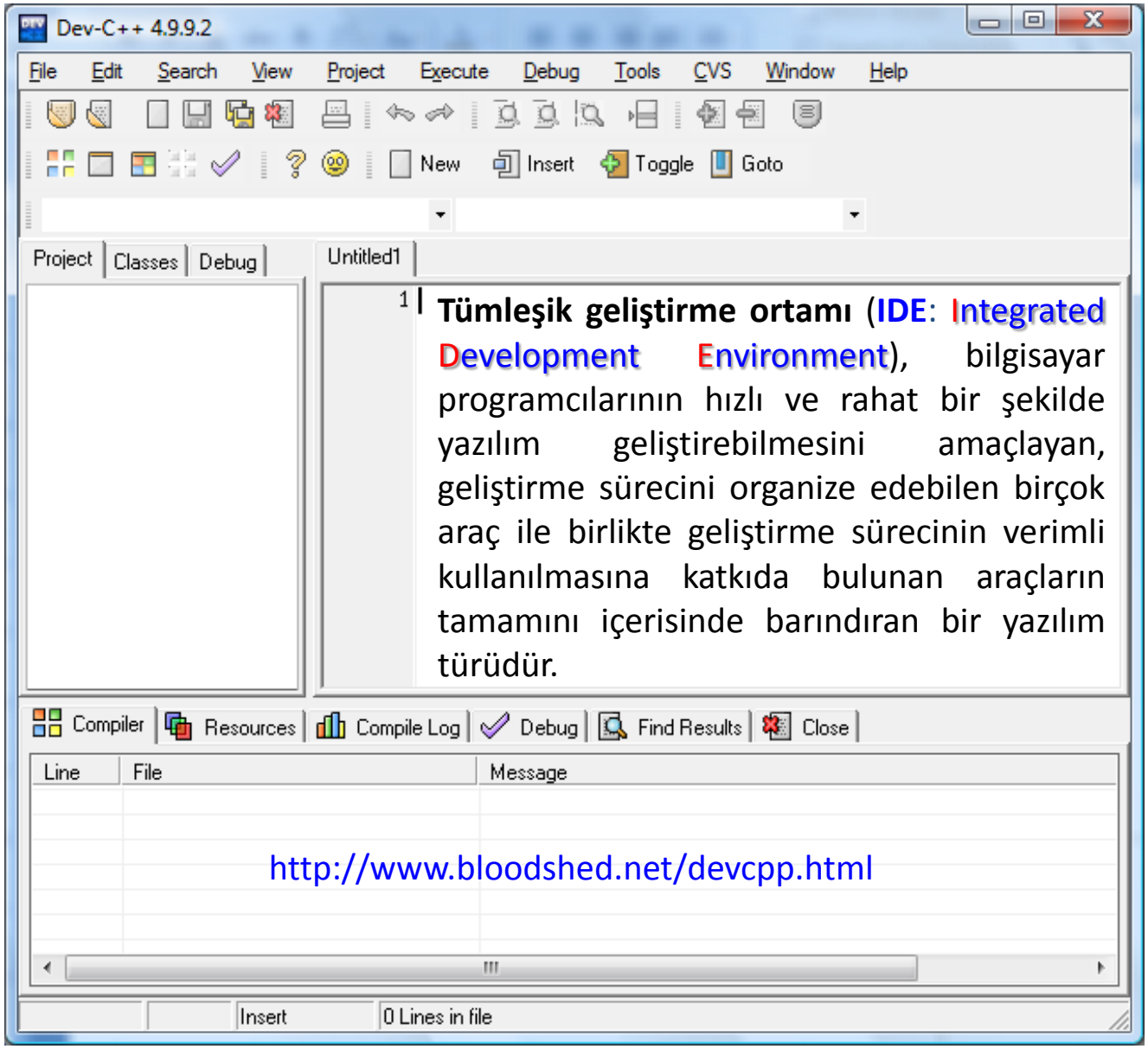
Örnek: 1'den 10'a kadar olan tam sayıların toplamını bulan bilgisayar programı için algoritma ve akış diyagramı.

Algoritma

1. Adım: Başla.
2. Adım: x=1, t=0.
3. Adım: x'leri t'de topla (t=t+x).
4. Adım: x'i bir arttır (x=x+1).
5. Adım: x<=10 ise 3. Adıma git.
6. Adım: t'yi yaz.
7. Adım: Bitir.

Akış Diyagramı





C

AT&T Bell laboratuarlarında, Ken Thompson ve Dennis M. Ritchie tarafından UNIX İşletim Sistemi'ni geliştirebilmek amacıyla B dilinden türetilmiş yapısal bir programlama dilidir.

Günümüzde neredeyse tüm işletim sistemlerinin (Microsoft Windows, GNU/Linux, *BSD, Minix) yapımında %95'lere varan oranda kullanılmış, halen sistem, sürücü yazılımı, işletim sistemi modülleri ve hız gereken her yerde kullanılan oldukça yaygın bir dildir.

[*] ilkprog.c

```
1  /* İlk C Programı*/
2  /* Bilgisayar Programlama
3      Dersinin ilk Uygulama Örneği*/
4  #include<stdio.h>
5  #include<conio.h>
6  int x,y;
7  float z;
8  main()
9      {
10         printf("Merhaba Dunya");
11         getch();
12         return 0;
13     }
14
15
```

"/*" ve "*/" arasında kalan ifade derleyici (compiler) tarafından dikkate alınmaz. Bu bildirim, ifadenin açıklama olduğu anlamına gelmektedir. Kaynak program içinde herhangi bir noktasında yer alabilir.

Kaynak programda ihtiyaç duyulan başlık (header) dosyalarının, önişlemciye (preprocesser) bildirimi

Kaynak programda ihtiyaç duyulan global değişkenlerin bildirimi (declaration). Bu bildirimler ana fonk. Dahil, diğer fonk. İçinde de mümkündür.

Program gövdesi, bu kısım ihtiyaç duyulan yerel değişkenlerin tanımlarını ve işlenecek komut satırlarını içerir. Program gövdesinin kendisi de bir fonksiyondur. Her kaynak program , ana (main) fonksiyonunu içermek zorundadır. Kaynak programın başlatılması ve sonlandırılması bu fonksiyon aracılığı ile gerçekleştirilir.

Komut satırı, ";" ile sonlandırılan her bir ifade bir komut satırını oluşturur.

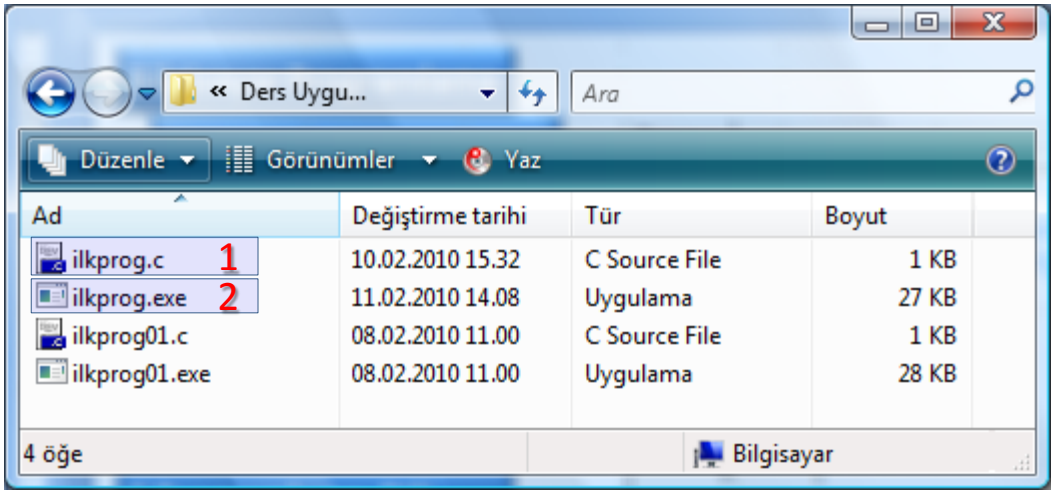
"{" ve "}" arasında kalan komut satırı grubu ise komut bloğu olarak ifade edilir. Örnek programdaki kod bloğu ana (main) fonksiyonuna ait kod bloğudur.)

ilkprog.c

```
1  /* İlk C Programı*/
2  /* Bilgisayar Programlama
3      Dersinin ilk Uygulama Örneği*/
4  #include<stdio.h>
5  #include<conio.h>
6  int x,y;
7  float z;
8  main()
9      {
10         printf("Merhaba Dunya");
11         getch();
12         return 0;
13     }
14
15
```

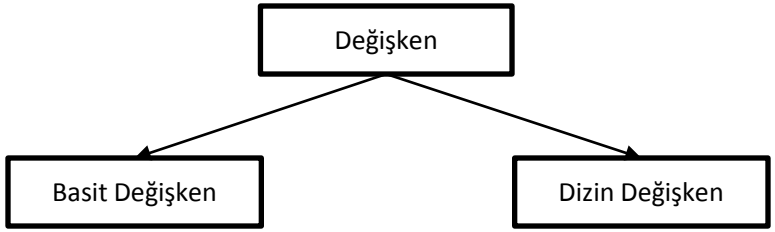
1

ilkproc.c kaynak programı ve derleyici (compiler) ile derlendikten sonra elde edilen heder programın (.exe) çalıştırılması sonucunda elde edilen ekran çıktısı.



Değişken , belirlenmiş bir değeri saklamak için bellekte ayrılan alanların sembolik olarak isimlendirilmesidir.

Böyle bir yol seçilmemiş olsaydı, bellekte saklanmış her bir değer için, 0x0000ffff gibi, bellek adreslerini bilmek gerekecekti.



Bir kaynak program içinde birden fazla değişken kullanılabilir. Dolayısı ile her bir değişkenin bir birinden ayırt edilebilmeleri gerekir ve bu nedenle benzersiz bir tanıtıcıya (identifier) ihtiyaç duyar.

Kullanılacak olan bir değişken için geçerli bir belirleyici, ilk karakteri daima harf olmak ("_" olabilir ancak harfle başlaması tercih edilir) kaydı ile harflerden, rakamlardan ve/veya "_" oluşturulabilir. " " (white space), noktalama işaretleri ve/veya semboller bir belirleyicinin parçası olamazlar.

Dikkat edilmesi gereken diğer bir konu ise belirleyicinin, C diline ve/veya kullanılan derleyiciye has ayrılmış anahtar kelimeler (reserved keywords), standart kütüphanede yer alan veya programcının kendi yazdığı fonksiyon tanıtıcısı (identifier) ile aynı olamayacağıdır.

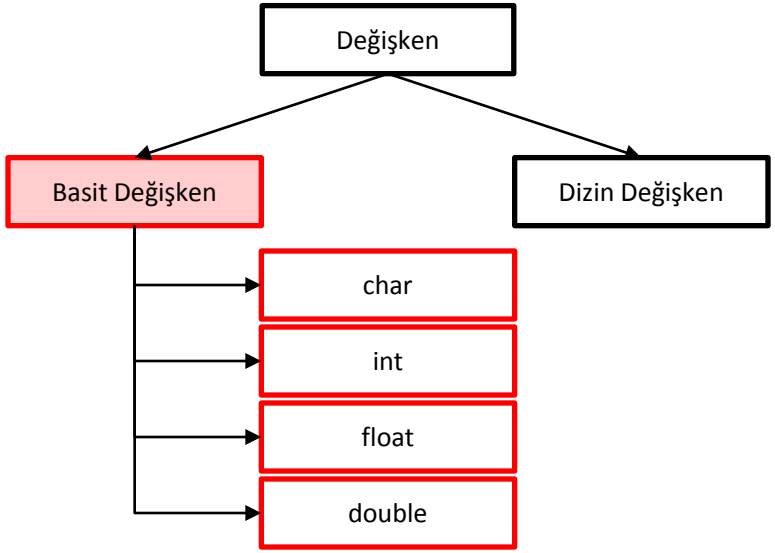
C diline has anahtar kelimelere (reserved keywords) bazı örnekler:

Auto, break, case, return, for, float,…

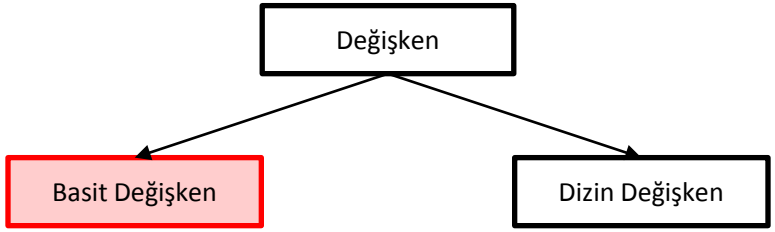
Örnek olarak verilen anahtar kelimeler kesinlikle değişkenin belirleyicisi olarak kullanılamaz.

Bir değişken belirlerken, dikkat edilmesi gerek diğer bir konuda, C dilinin büyük harf, küçük harfe duyarlı olmasıdır. Örneğin, toplam, Toplam ve TOPLAM belirleyicileri farklı üç değişkeni temsil eder ve farklı veri tiplerinde olabilirler, aynı veri tipinde farklı değerler taşıyabilirler.

```
toplam=1;
Toplam=1.55;
TOPLAM="Tam Sayı";
```



Veri Tipi	Tanımı	Büyüklüğü	Aralığı
[**signed** veya **unsigned**] char	Karakter veya küçük tam sayı	1 Byte	İşaretli: [-128, +127] İşaretsiz: [0, +255]
[**signed** veya **unsigned**] short [**int**]	Kısa tam sayı	2 Byte	İşaretli: [-32 768, +32 767] İşaretsiz: [0, +65 535]
[**signed** veya **unsigned**] int	Tam Sayı	2 Byte	İşaretli: [-32 768, +32 767] İşaretsiz: [0, +65 535]
[**signed** veya **unsigned**] long [**int**]	Uzun Tam Sayı	4 Byte	İşaretli: [-2 147 483 648, +2 .147 483 647] İşaretsiz: [0, +4 294 967 296]
float	Kayan noktalı sayı	4 Byte	[$\pm$ 1.7x10^{-39}, $\pm$ 1.7x10^{38} , $\pm$ 1.7x10^{38}]
double	Çift duyarlıklı sayı	8 Byte	[$\pm$ 1.7x10^{-308}, $\pm$ 1.7x10^{308}]
long double	Uzun çift duyarlıklı sayı	10 Byte	[$\pm$ 1.7x10^{-4933}, $\pm$ 1.7x10^{4933}]



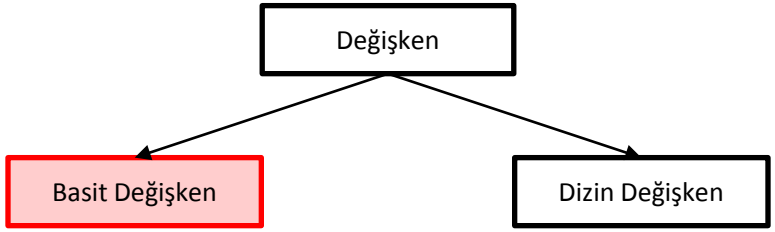
Kaynak programda kullanılacak olan değişkenlerden derleyicinin haberdar olması gerekir. Kullanılmadan önce, kullanılacak olan değişkenlerin özellikleri hakkında derleyiciye bilgi verilmesi işlemine bildirim (declaration) denir.

Genel Bildirim Biçimi:

Veri_Tipi Değişken_Adı_1, Değişken_Adı_2,… ;

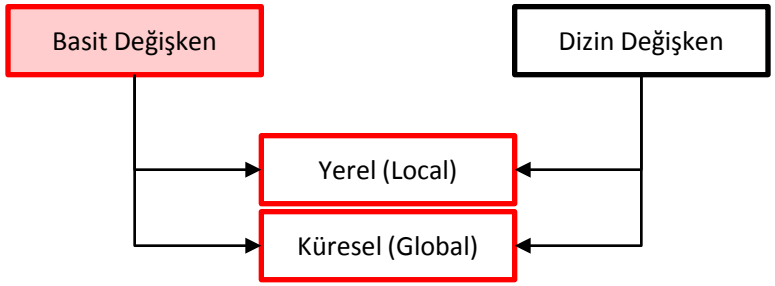
- Komut satırı sonlandırıcısı
- Kullanılan değişkenleri bir birinde ayırt edilmesini sağlayan tanıtıcı (identifier)
- Veri tipi belirleyicisi (specifier)

```
int  x, X;       /*Tam sayı tipinde bildirimi yapılmış x, X değişkenleri*/
float  t, x_1; /*Kayar noktalı sayı tipinde bildirimi yapılmış t, x_1 değişkenleri*/
```



Kaynak programda kullanılacak olan değişkenlerin bildirimi (declaration) yapıldı. Peki bu değişkenlerin kapsama alanı (scope) neresidir? Hangi değişken nerede kullanılabilir veya nerede etkindir?

Bildirimi (declaration) yapılan bir değişken, bildirimin yapıldığı yerde geçerlidir?



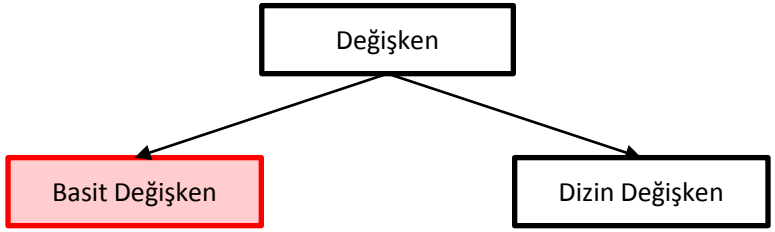
değbildir.c

```
1  /*
2  Değişken bildirimi ve etki alanı
3  */
4  #include<stdio.h>
5  #include<conio.h>
6  /* Küresel Değişken */
7  int x=0, X=1;
8  main()
9      {
10         /* Ana fonksiyonda etkin değişkenler */
11         printf("Ana fonksiyon icinden yazdirilan degisken degerleri\n");
12         printf("Kuresel x=%d\nKuresel X=%d\n",x,X);
13         int x=10, X=10;
14         printf("Ana fonk. Yereli x=%d\nAna fonk.  Yereli X=%d\n",x,X);
15             /* A Kod Bloğu Başlangıcı*/
16             {
17                 /* A Kod bloğunda etkin
18                     değişkenler */
19                 int t=100;
20                 printf("A Kod Blogu icinden yazdirilan degisken degerleri\n");
21                 printf("Kuresel x=%d\nKuresel X=%d\n",x,X);
22                 printf("Ana fonk. Yereli x=%d\nAna fonk.  Yereli X=%d\n",x,X);
23                 printf("A Kod Blogu Yereli t=%d\n",t);
24             } /* A Kod Bloğu Bitişi */
25         getch();
26         return 0;
27     }
```

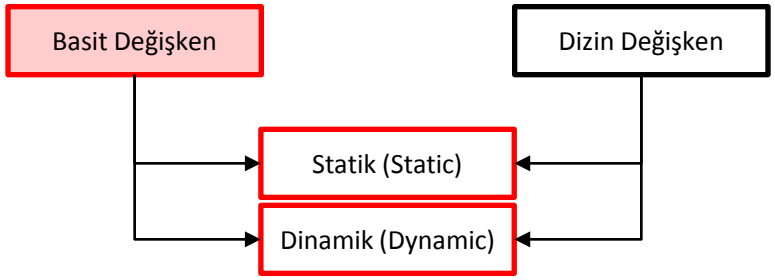
? Neden farklı değerler veriyor

? Neden farklı değer

```
Ana fonksiyon icinden yazdirilan degisken degerleri
Kuresel x=0
Kuresel X=1
Ana fonk. Yereli x=10
Ana fonk.  Yereli X=10
A Kod Blogu icinden yazdirilan degisken degerleri
Kuresel x=10
Kuresel X=10
Ana fonk. Yereli x=10
Ana fonk.  Yereli X=10
A Kod Blogu Yereli t=100
```

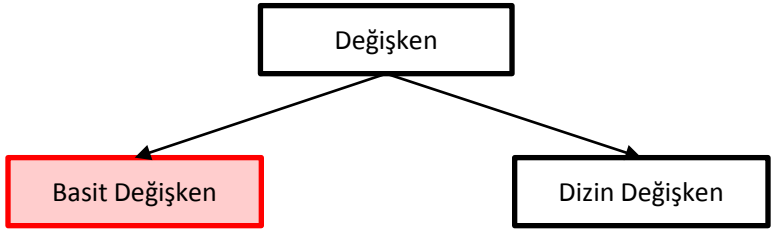


Kaynak programda kullanılacak olan değişkenler bir yaşam süresine de sahiptirler. Bildirim (declaration) esnasında belirtilmediği sürece, değişkenin kapsama alanının icrası bittiğinde otomatik olarak yaşam süresi sona erer.



Bu bildirim auto belirleyicisi (specifier) ile ifade edilir. Yerel değişkenler otomatik olarak auto'dur. Yerel değişkenin kapsama alanının icrası sona erdiğinde, yerel değişkenin yaşam süresi de sona erer. Bu durum, ihtiyaç duyulması halinde, static belirleyicisi ile hedef programın çalışma süresince saklanır.

```
static int  x, X;          /*Statik, tam sayı tipinde bildirimi yapılmış x, X değişkenleri*/
```



Bir değişken için geçerli bir bildirim yapıldığında, eğer değişken yerel ise rastgele bir değer atanır. Bu durum, küresel ve dizin değişkenler için geçerli değildir. Değişken bildirimi esnasında, eğer gerekiyorsa, ilk değer ataması (initialization) yapılarak bunun önüne geçilebilir.

```
int x=1, X;   /*Tam sayı tipinde bildirimi yapılmış ve ilk değeri 1 olarak atnmış x ve X
                değişkenleri*/
```

değbildir 01.c

```
1  /* Değişken Bildirimleri */
2  #include<stdio.h>
3  #include<conio.h>
4  int kureselx, kuresely;
5  main()
6      {
7          short int x=1, X;
8          float y=.1;
9          double Y;
10         printf("Yerel Degiskenler x = %d, X = %d\n",x,X);
11         printf("Yerel Degiskenler y = %f, Y = %f\n",y,Y);
12         printf("Kuresel Degiskenler kureselx = %d, kuresely = %d\n",
13             kureselx,kuresely);
14         getch();
15         return 0;
16     }
```

Küresel (global) tam sayı değişkenler

Ana (main) fonksiyonu yerel (local) kısa tam sayı değişkenleri, x'e 1 ilk değeri atanmış

Ana (main) fonksiyonu yerel (local) kısa tek duyarlıklı sayı değişkeni, ilk değeri 0.1 olarak atanmış.

Ana (main) fonksiyonu yerel (local) çift duyarlıklı sayı değişkeni

?

değbildir 01.c	13.02.2010 16.23	C Source File	1 KB
değbildir 01.exe	13.02.2010 16.23	Uygulama	28 KB

```
Yerel Degiskenler x = 1, X = 12288
Yerel Degiskenler y = 0.100000, Y = 0.000000
Kuresel Degiskenler kureselx = 0, kuresely = 0
```

4. komut satırında bildirilen küresel tam sayı değişkenlerin ekran çıktısı ilk değer ataması yapılmamasına rağmen 0 "sıfır" olarak belirlenmiştir. 7. komut satırında bildirilen kısa tam sayı yerel değişkenlerden x ilk değeri 1 "bir" olarak belirlenmişken, X değişkenin değeri rastgele bir değer olarak atanmıştır.

Sabitler belirli bir değeri, kaynak program içinde ifade etmek için kullanılır. Örneğin, a=5; kod satırındaki 5 değişmez bir sabittir.

Sabitler; tam sayılardan (integer numerals), kayan noktalı sayılardan (floating-point numerals), çift duyarlıklı sayılardan (double precision floating-point numerals), karakterler (character), karakter dizilerinden (strings) oluşur.

Sabit	Açıklama
1776	Tam Sayı
707	
-273	
0X12 veya 0x12	16 Tabanı (0 ve x veya X'ten sonra)
0113	8 Tabanı (0'dan sonra)
707	Tam veya kısa tam sayı (Varsayılan (default) tip int)
707u (veya U)	İşaretsiz tam veya kısa tam sayı (unsigned int or short int)
707l (veya L)	Uzun tam sayı (long int)
707ul (veya UL)	İşaretsiz uzun tam sayı (unsigned long int)

Tam sayı sabitlerin varsayılan tipi int'tir. Sabitlerin sonuna eklenen u veya U, l veya L ve ul veya UL son ekleri tam sayı sabitlerin nasıl ele alınacağını belirtir.

Kayar noktalı sabitler ondalıklı veya tam bir sayı ile "e" karakteri ve kuvvet değerinden meydana gelir.

"e" → "by ten at the X^{th} height"

Sabit	Açıklama
3.14159	3.14159
6.02E23	$6.02 \cdot 10^{23}$
1.6E-19	$1.6 \cdot 10^{-19}$
3.14159l (veya L)	Uzun çift duyarlıklı sayı (long double)
6.02E23F (veya f)	Kayar noktalı sayı (float)

Nokta içeren ve f ve F soneki almış sabitler, float olarak ele alınırlar.

Sonuna f veya F soneki almamış ve float limitlerini aşmış sabitler double olarak ele alınırlar .

Karakter (char) tipi sabitler ise biraz farklı kullanılır. Bu tipteki sabitler ‘’ ile gösterilirler. Örneğin, char x=‘a’; şeklinde ifade edilirler.

Karakter dizisi (string) tipi ise “” ile ele alınırlar.

Sabit	Açıklama
‘a’	Karakter (char)
‘x’	
“Merhaba Dünya”	Karakter dizisi (veya katarı) (string)
x	?

Nokta içeren ve f ve F soneki almış sabitler, float olarak ele alınırlar.

Sonuna f veya F soneki almamış ve float limitlerini aşmış sabitler double olarak ele alınırlar .

Kontrol karakterleri (Escape Sequence or Escape Codes) yer almaktadır. Bu karakterler önceden belirlenmiştir ve bazı işlemleri yerine getirir.

Kod	İşlev
\a	Alert (Sesli uyarı)
\b	Backspace (imleci bir sola kaydır)
\f	Form Feed (bir sonraki sayfanın başına geç)
\n	New Line (bir sonraki satıra geç)
\r	Carriage Return (satır başına geç)
\tn	Horizontal Tab (yatay sekme, n değiştirilebilir)
\v	Vertical Tab (dikey tab, sütun değiştirmeden alt satıra geç)
\\	Back Slash (\)
\?	Question Mark (?)
\’	Single Quote (’)
\”	Double Quote (”)

```
printf(“Merhaba Dunya\nifadesi iki satirda yazdirilmaktadir ve bipleyecektir.\a” );
```

Kontrol karakterleri (Escape Sequence or Escape Codes) yer almaktadır. Bu karakterler önceden belirlenmiştir ve bazı işlemleri yerine getirir.

[*] sabitler.c

```
1  /*
2  Sabit Kullanım Örnekleri
3  */
4  #include<stdio.h>
5  #include<conio.h>
6  main()
7      {
8          int x;
9          float y;
10         double z;
11         double Z;
12         x=1;
13         printf("x=%d\n",x);
14         y=1.0;
15         printf("y=%.3f\n",y);
16         z=1.1F;
17         printf("z=%2.2f\n",z);
18         Z=-3.45E+03L;
19         printf("Z=%2.3e\n\a",Z);
20         printf("Z=%2.3E\n\a",Z);
21         getch();
22         return 0;
23     }
```

Sırasıyla, tam sayı, kayar noktalı ve çift duyarlıklı kayar noktalı olarak bildirilmiş x, y, z ve Z değişkenleri

Tam sayı tipindeki x değişkenin 1 sabiti atanmış ve ekrana çıktı olarak gönderilmiş.

Kayar noktalı sayı tipindeki y değişkenine 1.0 sabiti atanmış ve ekrana çıktı olarak gönderilmiş.

Çift duyarlıklı kayar noktalı sayı tipindeki z değişkenine 1.1 kayar noktalı sabiti atanmış ve ekrana çıktı olarak gönderilmiş.

Çift duyarlıklı kayar noktalı sayı tipindeki Z değişkenine -3.45E+03 uzun çift duyarlıklı sabiti atanmış ve ekrana çıktı olarak gönderilmiş.

```
x=1
y=1.000
z=1.10
Z=-3.450e+003
Z=-3.450E+003
```

Başlık (header) dosyası, C dili bildirimleri ve/veya sabit, makro tanımlarını içeren bir dosyadır. # operatörü ve include komutu ile C önişlemcisi (preprocessor) tarafından ön işleme tabi tutularak kaynak programa eklenmesi sağlanır.

[*] sabitler.c

```
1  /*
2  Sabit Kullanım Örnekleri
3  */
4  #include<stdio.h>
5  #include<conio.h>
6  main()
7      {
8          int x;
9          float y;
10         double z;
11         double Z;
12         x=1;
13         printf("x=%d\n",x);
14         y=1.0;
15         printf("y=%.3f\n",y);
16         z=1.1F;
17         printf("z=%2.2f\n",z);
18         Z=-3.45E+03L;
19         printf("Z=%2.3e\n\a",Z);
20         printf("Z=%2.3E\n\a",Z);
21         getch();
22         return 0;
23     }
```

Başlık (header) dosyası, C dili bildirimleri ve/veya sabit, makro tanımlarını içeren bir dosyadır. # operatörü ve include komutu ile C önişlemcisi (preprocessor) tarafından ön işleme tabi tutularak kaynak programa eklenmesi sağlanır.
Bu dosyalarda, önceden tanımlanmış hazır fonksiyonların bildirimleri, makro fonksiyon ve/veya sabit bildirimler ve derleyiciyi yönlendiren önişlemci tanımlamaları vardır.
Kaynak programda, örneğin x değişkeni değerinin ekrana çıktı olarak verilmesini sağlayan prinf() fonksiyonu stdio.h başlık dosyasında bildirilmiştir.

Örneğin, karekök alan sqrt() fonksiyonu math.h başlık dosyasında bildirilmiştir.

C dilinde matematiksel işlemler için kullanılan aritmetik operatörleri şu şekilde sıralanabilir:

Operatör		İşlev
Aritmetik Operatörler	+	Toplama işlemi
	-	Çıkarma işlemi ve negatif işareti
	*	Çarpma işlemi
	/	Bölme işlemi
	%	Kalan (mod) işlemi
	++	Arttırma işlemi (++x veya x++)
	--	Azaltma işlemi (--x veya x--)
Bileşik Atama Operatörleri	+=	Bileşik atama işlemi y+=x (y=y+x)
	-=	Bileşik atama işlemi y-=x (y=y-x)
	*=	Bileşik atama işlemi y*=x (y=y*x)
	/=	Bileşik atama işlemi y/=x (y=y/x)

C dilinde karşılaştırma ve mantıksal işlemler için kullanılan ilişkisel ve eşitlik operatörleri şu şekilde sıralanabilir:

Operatör		İşlev
İlişkisel ve Eşitlik Operatörleri	==	Eşit (equal to)
	!=	Eşit değil (not equal to)
	>	Büyük (greater than)
	<	Küçük (less than)
	>=	Büyük eşit (greater than or equal to)
	<=	Küçük eşit (less than or equal to)

Kaynak programlar her zaman doğrusal bir komut satırı dizisi şeklinde olmayabilir. İzlediği süreç içerisinde başka komut satırlarına atlaması, bir kod bloğunu tekrar etmesi veya karar vermesi gerekebilir. C dili, kaynak programın neyi, ne zaman ve ne şartlar altında yapması gerektiğini belirlemeye yarayan kontrollere sahiptir. Bu yapılar şu şekilde sıralanabilir:

IF ve **IF…ELSE**

WHILE ve **DO…WHILE**

FOR

SWITCH

C dili, kaynak programın neyi, ne zaman ve ne şartlar altında yapması gerektiğini belirlemeye yarayan kontrollere sahiptir. Koşul yapısı, bir komut satırının veya komut bloğunun belli bir koşula bağlı olarak işletilmesini sağlayan yapıdır.

IF ve ELSE

IF yapısı, bir koşula bağlı olarak komut satırı veya komut bloğu işletilmek istendiğinde kullanılır.

```
if (koşul) komut_satırı;
```

```
if (koşul)
   {komut_satırı_1;
    komut_satırı_2;}
```

```
if (koşul)
     { komut_satırı_1;
       komut_satırı_2;}
else
     { komut_satırı_1;
       komut_satırı_2;}
```

```
if (koşul_1)
     { komut_satırı_1;
       komut_satırı_2;}
else if (koşul_2)
     { komut_satırı_1;
       komut_satırı_2;}
else
     { komut_satırı_1;
       komut_satırı_2;}
```

C dili, kaynak programın neyi, ne zaman ve ne şartlar altında yapması gerektiğini belirlemeye yarayan kontrollere sahiptir. Döngü yapıları, bir komut satırının veya komut bloğunun belli bir koşula bağlı veya belli bir sayıda tekrarla işletilmesini sağlayan yapılardır.

WHILE

WHILE, bir komut satırının veya komut bloğunun, bir koşula bağlı olarak tekrarlı işletilmesini sağlayan kontrol yapısıdır.

```
while (koşul)
   {komut_satırı_1;
    komut_satırı_2; }
```

Koşul sağlanmasından sonra komut bloğu işletilecektir.

```
do
   { komut_satırı_1;
     komut_satırı_2; }
while (koşul)
```

Komut bloğu işletildikten sonra koşul sağlaması yapılacaktır.

C dili, kaynak programın neyi, ne zaman ve ne şartlar altında yapması gerektiğini belirlemeye yarayan kontrollere sahiptir. Döngü yapıları, bir komut satırının veya komut bloğunun belli bir koşula bağlı veya belli bir sayıda tekrarla işletilmesini sağlayan yapılardır.

FOR

FOR, bir komut satırının veya komut bloğunun, belli sayıda tekrarla işletilmesini sağlayan kontrol yapısıdır.

```
for(başlangıç_değeri; koşul; artış_değeri) komut_satırı;
```

```
for(başlangıç_değeri; koşul; artış_değeri)
    { komut_satırı_1;
      komut_satırı_2; }
```

C dili, kaynak programın neyi, ne zaman ve ne şartlar altında yapması gerektiğini belirlemeye yarayan kontrollere sahiptir. Seçim yapısı, **koşul yapısına** benzer şekilde, belli bir komut satırının veya komut bloğunun belli bir koşula bağlı işletilmesini sağlayan bir yapıdır. Koşul yapısından farklı olarak, ikiden fazla seçeneğe sahiptir. Koşul yapısının **doğru** ve **yanlış** olmak üzere iki seçeneğe sahip olduğunu hatırlayınız.

SWITCH

SWITCH, ikiden fazla seçenekli, bir koşula bağlı olarak komut satırı veya komut bloğu işletilmek istendiğinde kullanılır.

```
switch(değişken)
    { case sabit_değer_1:
      komut_bloğu;
      break;
      case sabit_değer_2:
      komut_bloğu;
      break;
      …
      default:
      komut_bloğu; }
```

SWITCH, değişkenin aldığı değere karşılık gelen durumun (case) olup olmadığına bakar. Eğer var ise, o durumun (case) komut bloğunu işletir. Eğer yok ise, varsayılan (default) kod bloğunu işletir. Buradaki sabit_değer'ler, 0, 1, 'a' gibi değerlerdir. Değişken kullanılamaz.

C dili, **fonksiyon** olarak adlandırılan altprogramdan oluşan bir yapıya sahiptir. Bu **fonksiyon**lar (altprogramlar), önceden hazırlanmış ve kütüphanede bulunan hazır fonksiyonlar (**printf()**, **getch()**, **main()**) iken, bazıları programcı tarafından hazırlanan fonksiyonlardır.

Genel Tanımlama Biçimi:

```
Veri_Tipi  Fonksiyon_Adı(parametre_1, parametre_2,…)
   {
       Komut_Bloğu;
       return değişken_adı;
   }
```

Genel tanımlama biçimine göre, **Fonksiyon_adı** isimli fonksiyonumuzun bildirimi esnasında, kurallara uygun şekilde belirlenmiş ismi yanında **veri_tipi** ile döndüreceği değerin tipi, "**()**" ile veri tipleri belirtilerek parametre listesi de belirlenmelidir. **Return** komutu ise, fonksiyonu sonlandırırken, fonksiyonun işletilmesi sonucunda elde edilen değerin döndürmesini sağlar.

C dili, **fonksiyon** olarak adlandırılan altprogramdan oluşan bir yapıya sahiptir. Bu **fonksiyon**lar (altprogramlar), önceden hazırlanmış ve kütüphanede bulunan hazır fonksiyonlar (**printf()**, **getch()**, **main()**) iken, bazıları programcı tarafından hazırlanan fonksiyonlardır.

```
//Fonksiyon Örneği
#include<iostream.h>
int toplama(int a , int b)
    {
      int t;
      t=a+b;
      return t ;
    }
main()
    {
      int z;
      z=toplama(5, 3);
      printf("5+3=%d",z);
      return 0;
    }
```

Toplama fonksiyonu, parametre listesindeki değişkenlerin değerlerini toplayarak, toplam değerini döndüren bir fonksiyondur. Peki bunu nasıl yapmaktadır?

Neden main fonksiyonundan önce kodlanmıştır?

a, b ve r değişkenlerinin etki alanı (scope) neresidir?

C dili, **fonksiyon** olarak adlandırılan altprogramdan oluşan bir yapıya sahiptir. **Fonksiyon**lar (altprogramlar), kaynak programda kullanılmadan önce tanımlanmaları gerekir. Aksi halde derleme hatası ile karşılaşılacaktır. Bu hatayı önlemek için fonksiyonun prototipi (**prototype function**), sadece argüman listesinde yer alacak argümanların veri tipleri bildirilerek, tanımlanabilir.

```
//Fonksiyon Örneği
#include<iostream.h>
int toplama(int, int);
main()
    {
      int z;
      z=toplama(5, 3);
      printf("5+3=%d",z);
      return 0;
    }
int toplama(int a, int b)
    {
      int t;
      t=a+b;
      return  t ;
    }
```

Prototip bildirimi fonksiyonun, fonksiyonun kullanımından önce, tanımlanması zorunluluğunu ortadan kaldırmaktadır.

Ancak dikkat edilmesi gereken konu, fonksiyonun prototip tanımlanması ile gerçek tanımının uyumlu olmasıdır.

C dili, **fonksiyon** olarak adlandırılan altprogramdan oluşan bir yapıya sahiptir. **Fonksiyon**lar (altprogramlar) inline olarak ta tanımlanabilmektedirler. Bu bildirim, fonksiyonun çağrıldığı komut satırına çağırılma kodunun yerine, kendisinin koda eklenmesini sağlar.

```
//Fonksiyon Örneği
#include<iostream.h>
inline int toplama(int a , int b)
    {
      int t;
      t=a+b;
      return  t ;
    }
main()
    {
      int z;
      z=toplama(5, 3);
      printf("5+3=%d",z);
      return 0;
    }
```

inline bildirimi sadece fonksiyonun tanımlanması esnasında bir kez kullanılır ve fonksiyonun çağırılması her zamanki gibidir.

Ancak çoğu derleyici, uygun olması durumunda **normal** tanımlanmış bir fonksiyonu **inline** olarak *optimize* edecektir.

Bunun yanında **inline** olarak tanımlanmış bir fonksiyonu ise, uygun olmaması halinde **normal** bir fonksiyon bildirimine dönüştürecektir.

C dili, **fonksiyon** olarak adlandırılan altprogramdan oluşan bir yapıya sahiptir. **Fonksiyon** (altprogram) daima değer döndürür mü ve/veya çalışabilmesi için bir parametreye ihtiyaç duyar mı? Bazı durumlarda fonksiyon, çalışması için parametreye ve/veya döndüreceği bir değere ihtiyaç duymaya bilir. Böyle durumlarda, döndüreceği bir değer olmaması ve/veya çalışması için parametreye gerek duymaması void komutu ile bildirilir.

```
//Fonksiyon Örneği
#include<iostream.h>
void mesaj(void)
     {
        printf("Bu fonksiyon\n");
        printf("iki sayıyı\n");
        printf("toplamaktadır.")
     }
main()
     {
        int z;
        mesaj();
        z=5+3;
        printf("5+3=%d",z);
        return 0;
     }
```

Kaynak program örneğinde **void** komutu **mesaj** fonksiyonunun bir değer döndürmeyeceğini ve çağırılırken herhangi bir parametreye ihtiyaç duymadığını belirtmektedir.

Fonksiyon, ana fonksiyondan çağırıldığında, kontrolü devralacak ve ekrana çıktı verdikten sonra kontrolü tekrar ana fonksiyona devredecektir.