

C Programlama Dili'ne Giriş

Ders 1: Giriş

Giriş

Bu ilk derste, bir C programının nasıl derlenip çalıştırılacağı ve Internet'te bulabileceğimiz derleyicilerden bahsedilecektir. En basit C programının derleyip çalıştırdıktan sonra, geriye kalan sadece C Programlama Dili'nin kurallarını, yapısını ve deyimlerini öğrenmekten ibarettir.

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** standardı **Standart C** olarak kabul edilmiştir.

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.

1.3 İlk C Programı

Program 1.1 de verilen C programı derlendikten sonra, ekrana 'Merhaba Dünya!' 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.

Program 1.1: *Derlendikten sonra ekrana 'Merhaba Dünya!' yazar*

```
01: /* ilk.c: ilk C programi */
02: #include <stdio.h>
03:
04: main()
05: {
06:     printf("Merhaba Dünya!\n");
07: }
```

`/* ... */`

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 dosyasını 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` önışlemcisi 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. Bunlar derleyicinin yardım kısmından veya derleyicinin kullanım kılavuzundan öğrenilebilir.

`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 dosyasıdır. 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.

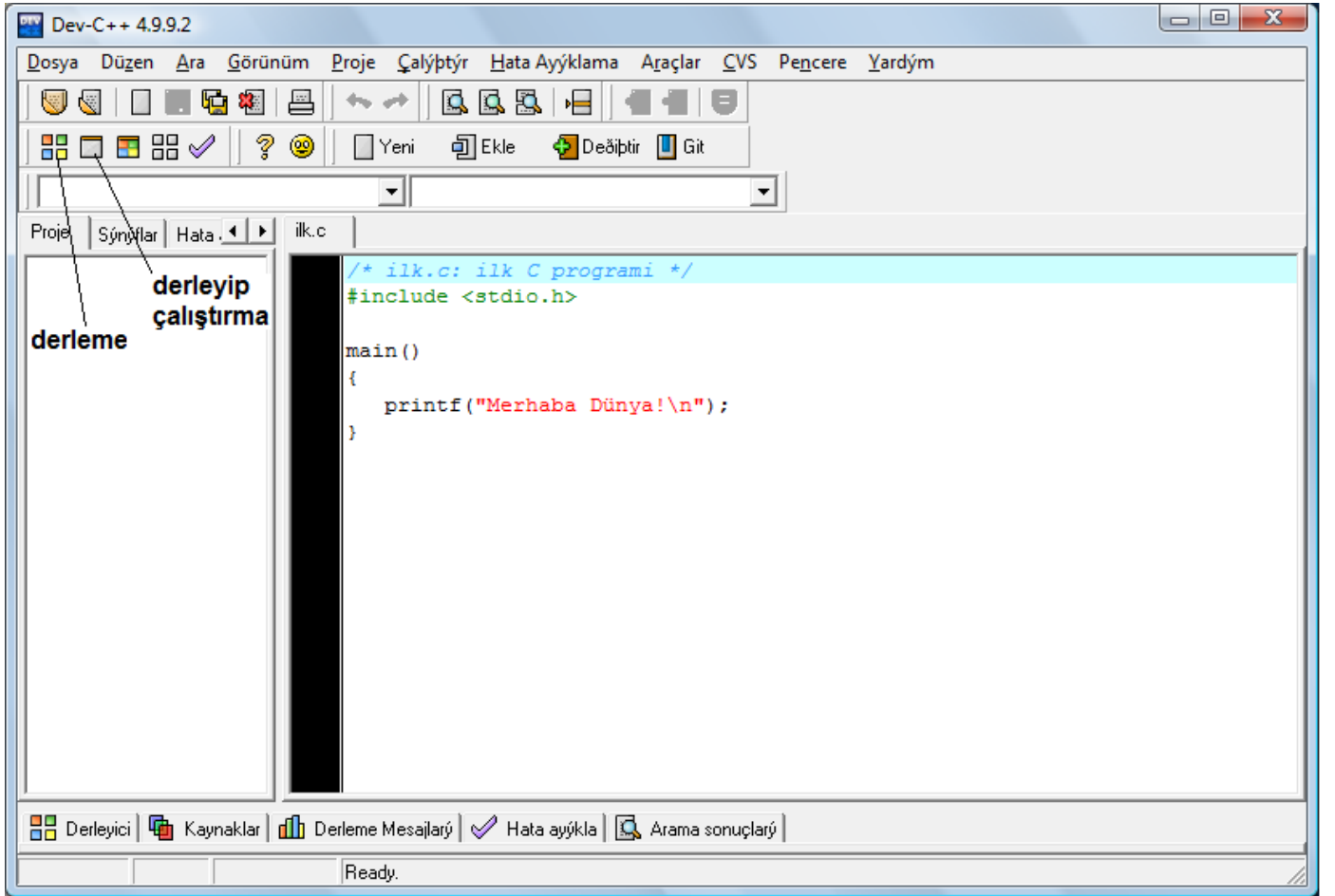
1.5 Kaynak Kodunun Derlenmesi

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. Bazı işletim sistemleri ile kullanılan C Derleyicileri ve bu derleyicilerde `ilk.c` programının komut satırında nasıl derleneceği Tablo 1.1'de verilmiştir. Eğer ismi geçen derleyicinin bir editörü varsa `ilk.c` bu editör de derlenebilir.

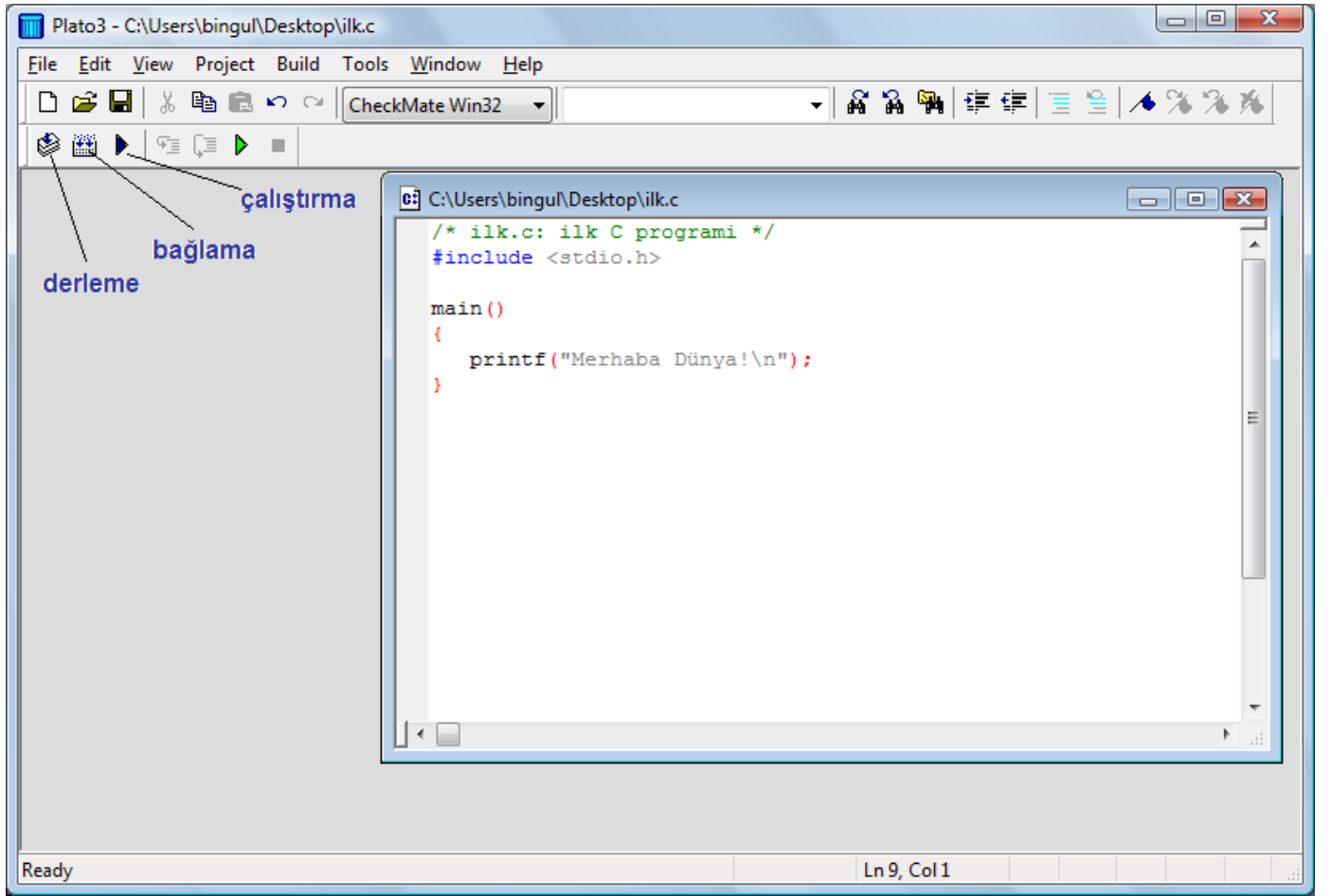
Tablo 1.1: İşletim sistemleri, bazı derleyiciler ve derleme komutları

İşletim Sistemi	Derleyici	Derleme	Çalıştırma
MS-DOS / Windows	Microsoft C	cl ilk.c	ilk.exe
	Borland Turbo C Web	tcc ilk.c	ilk.exe
	Borland C	bcc ilk.c	ilk.exe
	Zortec C	ztc ilk.c	ilk.exe
	GCC (GNU Compiler Collection) Windows için Web	gcc ilk.c -o ilk.exe	ilk.exe
UNIX / Linux	GCC (GNU Compiler Collection) Web	gcc ilk.c -o ilk	./ilk veya nice ilk

Bunların dışında, komut satırını kullanmadan, kodlarınızı Windows ortamında çalışan GCC tabanlı DevC++ veya Salford Plato3 derleyicileri ile derlemek mümkün. Bu tip derleyicilerde hata ayıklama işlemini kolaylaştırmak için kodlar farklı renkte gösterilir. Fakat program çıktıları için kullanılan ekran klasik DOS ekranıdır. Şekil 1.1 ve 1.2'de bu programların ekran görüntüleri verilmiştir.

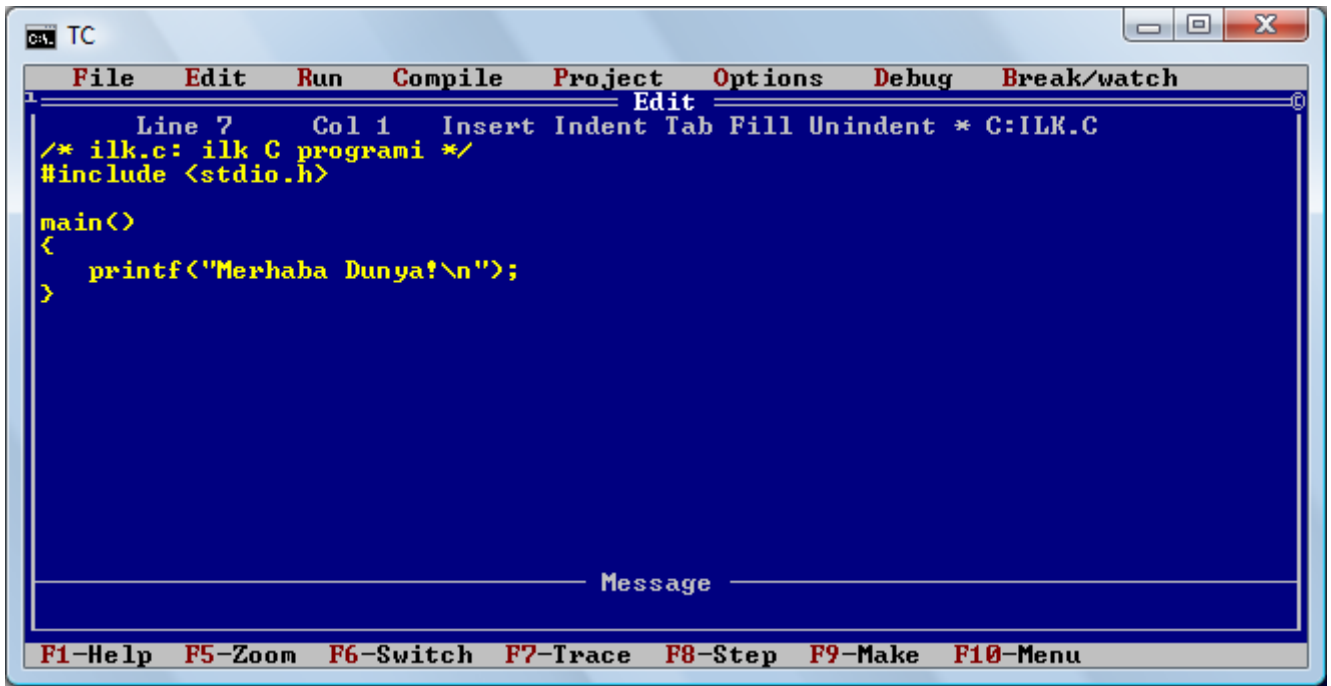


Şekil 1.1: DevC++ derleyicine ait editör. Derleme ve çalıştırma işlemleri araç çubuğu üzerindeki butonlarla yapılır.



Şekil 1.2: Silverfrost Salford (Plato3) derleyicine ait editör. Derleme, bağlama ve çalıştırma işlemleri araç çubuğu üzerindeki butonlarla yapılır.

Derslerimizde kullanılan kaynak kodları, Turbo C ve GCC derleyicileri ile komutsatırında derlenmiştir. Turbo C derleyicisi isteğe bağlı editörden veya komut satırından derlenebilir. Editörü başlatmak için C:\TC> dizini altındaki TC.EXE dosyasının çalıştırılması yeterlidir. Şekil 1.3'de Turbo C editör ortamı gösterilmiştir.



Şekil 1.3: Turbo C derleyicisine ait editör. Derleme için F9, Derleme bağlama ve çalıştırma işlemleri için CTRL+F9 tuş kombinasyonu kullanılabilir..

NOT
DevC++, Salford, GCC ve Turbo C derleyicilerini [C/C++ Derleyicileri](#) kısmında bulabilirsiniz.

ilk.c nin Borland Turbo C ve GCC Programları ile derlenmesi ve çalıştırılması:

DERLEME ve ÇALIŞTIRMA

MS DOS (Turbo C)	Linux (GCC)
C:\TC> tcc ilk.c	\$ gcc ilk.c -o ilk
C:\TC> ilk.exe	\$./ilk

ilk.c nin çıktısı:

ÇIKTI

Merhaba Dünya!

1.6 C Kodlarının Temel Özellikleri

Bir C programı aşağıda verilen özellikleri mutlaka taşımalıdır.

- Yazılımda kullanılacak olan her fonksiyon için ilgili başlık dosyası programın başına ileve 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

- Program açıklamaları ve döküman hazırlama program yazıldıkça yapın! 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 = pow(Vo,2) / (2*g);`

- `Tf = 2*Vo/g;`
- `Vy = Vo - g*t;`
- `y = Vo*t - (g*t*t)/2.0;`
- `z = ( a*cos(x) + b*sin(x) ) * log( 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.

Ders 2: *Veri Tipleri, Değişkenler ve Sabitler*

Giriş

Orta ve yüksek seviyeli dillerin hemen hemen hepsinde veri tipi ve değişken kavramı bulunmaktadır. Bu kısımda C programlama dilindeki temel veri tipleri, tanımlayıcılar, değişkenler ve sabitler konu edilecektir.

2.1 Veri Tipleri

Veri tipi (data type) program içinde kullanılacak değişken, sabit, fonksiyon isimleri gibi tanımlayıcıların tipini, yani bellekte ayrılacak bölgenin büyüklüğünü, belirlemek için kullanılır. Bir programcı, bir programlama dilinde ilk olarak öğrenmesi gereken, o dile ait veri tipleridir. Çünkü bu, programcının kullanacağı değişkenlerin ve sabitlerin sınırlarını belirler. 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.

NOT

Kısa ve uzun tamsayı tutacak tanımlayıcılar için `int` anahtar kelimesinin yazılmasına gerek yoktur.

```
short s; /* short int s; anlamında */
long k; /* long int k; anlamında */
```

Bir C programı içerisinde, veri tiplerinin bellekte kapladığı alan `sizeof` operatörü ile öğrenilebilir. İlgi çekici olan, bu alanların derleyiciye ve işletim sistemine bağlı olarak değişiklik göstermesidir. Program 2.1'de, `sizeof` operatörü kullanılarak, veri tiplerinin bellek uzunlularının nasıl ekrana yazdırılacağı gösterilmiştir. Programın çıktısı, farklı derleyiciler ve işletim sisteminde denendiğinde bu durum daha iyi anlaşılır. Lütfen inceleyin.

Program 2.1: Değişken tipleri ve türevlerinin bellekte kapladıkları alanlar

```
01: /* 02prg01.c : sizeof operatörünün kullanımı */
02:
03: #include <stdio.h>
04:
05: main()
06: {
07:     printf( "char          : %d bayt\n", sizeof(char));
08:     printf( "short         : %d bayt\n", sizeof(short));
09:     printf( "int           : %d bayt\n", sizeof(int));
10:     printf( "long          : %d bayt\n", sizeof(long));
11:     printf( "unsigned char : %d bayt\n", sizeof(unsigned
12: char));
13:     printf( "unsigned short : %d bayt\n", sizeof(unsigned
14: short));
15:     printf( "unsigned int   : %d bayt\n", sizeof(unsigned
16: int));
17:     printf( "unsigned long  : %d bayt\n", sizeof(unsigned
18: long));
19:     printf( "float          : %d bayt\n", sizeof(float));
20:     printf( "double         : %d bayt\n", sizeof(double));
21:     printf( "long double    : %d bayt\n", sizeof(long
22: double));
23: }
```

ÇIKTI

Windows (32 bit) Turbo C	Windows (32 bit) Salford	Linux (32 bit) GCC	Linux (64 bit) GCC
char : 1 bayt	char : 1 bayt	char : 1 bayt	char : 1 bayt
short : 2 bayt	short : 2 bayt	short : 2 bayt	short : 2 bayt
int : 2 bayt	int : 4 bayt	int : 4 bayt	int : 4 bayt
long : 4 bayt	long : 4 bayt	long : 4 bayt	long : 8 bayt
unsigned char : 1 bayt	unsigned char : 1 bayt	unsigned char : 1 bayt	unsigned char : 1 bayt
unsigned short : 2 bayt	unsigned short : 2 bayt	unsigned short : 2 bayt	unsigned short : 2 bayt
unsigned int : 2 bayt	unsigned int : 4 bayt	unsigned int : 4 bayt	unsigned int : 4 bayt
unsigned long : 4 bayt	unsigned long : 4 bayt	unsigned long : 4 bayt	unsigned long : 8 bayt
float : 4 bayt	float : 4 bayt	float : 4 bayt	float : 4 bayt
double : 8 bayt	double : 8 bayt	double : 8 bayt	double : 8 bayt
long double : 10	long double : 10 bayt	long double : 12	long double : 16
bayt		bayt	bayt

`int` veritipi ve türevleri ile hesaplanabilecek en küçük ve en büyük tamsayılar için aşağıdaki formül kullanılabilir:

$$\text{Alt sınır} = -2^{8 \cdot \text{sizeof}(\text{tip})}$$

$$\text{Üst sınır} = 2^{8 \cdot \text{sizeof}(\text{tip})} - 1$$

Örneğin 4 baytlık bir `int` tipi için:

$$\text{Alt sınır} = -2^{8 \cdot \text{sizeof}(\text{int})} = -2^{32} = -2147483648$$

$$\text{Üst sınır} = 2^{8 \cdot \text{sizeof}(\text{int})} - 1 = 2^{32} - 1 = 2147483647$$

Tablo 2.1'de bütün tipler, bellekte kapladıkları alanlar ve hesaplanabilecek (bellekte doğru olarak saklanabilecek) en büyük ve en küçük sayılar listelenmiştir.

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

Tablo 2.1: Değişken tipleri ve bellekte kapladıkları alanlar

2.2 Değişkenler

Değişkenler bilgisayarın geçici belleğinde bilginin saklandığı gözlere verilen sembolik adlardır. Bir C programında, bir değişken tanımlandığında bu değişken için bellekte bir yer ayrılır. Her değişkenin tuttuğu değer nasıl bir veri olduğunu gösteren (önceki bölümde anlatılan) bir veri tipi vardır.

C programlama dilinde, değişkenler ve sabitler programın başında bulunmalıdır. Bazı uygulamalarda değişkenin bir başlangıç değerinin olması istenir. Böyle durumlarda değişken bildirilirken başlangıç değeri verilebilir. Örneğin:

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

Deęişken isimleri verirken bazı kurallara uymak zorunludur. Bunlar:

- Deęişken adları en fazla 32 karakterden oluşabilir. 32 karakterden uzun deęişken adları ilk 32 karakteri deęerlendirilir. Geriye kalan karakterler işleme tabi tutulmaz.
- Deęişken adları ingiliz alfabesinde bulunan karakterler (A-Z) veya (a-z) yada rakamlar (0-9) ile yazılmalıdır. Türkçe karakterler, özel karakter veya boşluk karakteri kullanılamaz.
- Deęişken adları herhangi bir rakam ile başlayamaz. İlk karakter bir harf olmalıdır. Sonrakiler rakamlardan oluşabilir.
- Aşağıda verilen kelimeler ANSI C 'nin anahtar kelimeleridir (key words) ve deęişken 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

Bu kurallara göre aşağıdaki deęişken (sabit, fonksiyon) adlarının geçerlilięini inceleyiniz.

<u>Deęişken/Sabit/Fonksiyon/Yapı Adı</u>	<u>Geçerlilik</u>	<u>Açıklama</u>
asal	geçerli	-
Momentum	geçerli	-
ivme	geçerli	-
olasilik	geçerli	-
IsikHizi	geçerli	-
isik_hizi	geçerli	Alt çizgi karakteri '_' kullanılabilir
isik hizi	geçersiz	Boşluk karakteri kullanılamaz
ışık_hızı	geçersiz	Türkçe karakter kullanılamaz
1Bit	geçersiz	rakam ile başlanamaz
typedef	geçersiz	Anahtar kelimelerden birisi kullanılamaz

2.3 Sabitler

Sabit bildirimi, başlangıç deęeri verilen deęişken bildirimi gibi yapılır. Ancak, veri tipinin önüne `const` anahtar sözcüğü konmalıdır. Örneęin:

```
const float  PI = 3.142857;
const double NOT= 12345.8596235489;
const int    EOF= -1;
const char[] = "devam etmek için bir tuşa basın...";
```

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 programcılar tarafından geleneksel hale gelmiştir.

Birçok C programında sabitler `#define` önışlemci komutu ile de tanımlandığını görebilirsiniz. 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. Aşağıda verilen simgesel sabit bildirimleri geçerlidir. `#define` önışlemcisi ile makro fonksiyon tanımlama işlemleri, Bölüm 8 ve Bölüm 20'de anlatılacaktır.

```
#define MAX      100
#define DATA    0x0378
#define YARICAP  14.22
```

2.4 Rakamsal Bilgiler

C programlama dili içinde tanımlanabilecek sabit rakamlar *rakamsal bilgi* (literal) olarak adlandırılır. Her veri tipi kendi rakamsal bilgisine sahiptir. Bu bilgiler, kaynak kod içerisinde, özel değerleri ifade eder. Örneğin aşağıdaki atama işleminde 25 ve 17.2 sayıları gibi:

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

C dilinde bütün tamsayı sabitler varsayılan (default) olarak `int` tipinde, gerçel sayı sabitler varsayılan olarak `double` tipindedir. Ancak sabitleri gösteren rakamların sonuna eklenecek U (veya u), L (veya l) ve F (veya f) harfleri ile bu durum değiştirilebilir. Bu yüzden, aşağıdaki atamalar aynı anlamda değildir.

```
i = 25;      /* int          rakam */
i = 25U;     /* unsigned int rakam */
i = 25L;     /* long int     rakam */
i = 25UL;    /* unsigned long rakam */
i = 25L;     /* long int     rakam */

r = 17.2;    /* double        rakam */
r = 17.2L;   /* long double   rakam */
r = 17.2F;   /* float         rakam */
```

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. Buna gösterime göre, aşağıdaki atamalar aynı anlamdadır:

```
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.

```
'A'              /* bir karakter */
"Merhaba Dünya"  /* bir karakter kümesi */
```

Program 2.1'de, program içinde tanımlanan değişken sabitlerin ekrana nasıl yazdırılacağı gösterilmiştir.

Program 2.2: Değişkenlerin ve sabitlerin ekrana yazdırılması

```
01: /* 02prg02.c : Değişkenler ve sabitlerin ekrana
02: yazdırılması */
03:
04: #include <stdio.h>
05:
06: #define PI 3.141593
07:
08: int main()
09: {
10:     const int MAX = 100;
11:     char c = 'a';
12:     char *s = "Bu bir sicim";
13:     int i = 22;
14:     float f = 33.3;
15:     double d = 44.4;
16:
17:     printf("PI = %lf\n", PI);
18:     printf("MAX= %d\n", MAX);
19:     printf("c = %c\n", c);
20:     printf("s = %s\n", s);
21:     printf("i = %d\n", i);
22:     printf("f = %f\n", f);
23:     printf("d = %lf\n", d);
24:
25:     return 0;
}
```

ÇIKTI

```
PI = 3.141593
MAX= 100
c = a
s = Bu bir sicim
i = 22
f = 33.299999
d = 44.400000
```

2.5 Değişken Bildirim Yerleri ve Türleri

Yerel (local) Bildirim

Yerel değişkenler kullanıldığı fonksiyon içerisinde bildirilir. Yalnızca bildirildiği fonksiyon içerisinde tanınır ve kullanılabilir.

```
int topla(int a,int b)
{
/* yerel (local) değişken c nin bildirimi */
int c;
c = a + b;
return c;
}
```

Genel (general) Bildirim

Genel değişkenler bütün fonksiyonların dışında bildirilir. Bir değişken program boyunca sürekli olarak kullanılıyorsa genel olarak bildirilmelidir.

```
#include <stdio.h>

void karesi();

/* m ve n global tip değişkendir.
   Bu iki değişken tüm program boyunca kullanılmaktadır. */

int m,n;

main()
{
    m=7;
    karesi();
    printf("%d nin karesi %d dir",m,n);
}

void karesi(){
    n = m*m;
}
```

2.6 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.

```
char kr;
int tam;
long int ltam;
unsigned int utam;
short int stam;
float f;
double d;
```

bildirimlerine göre:

<i>Bağıntı</i>	<i>Sonuç Tipi</i>
-----	-----
kr+5	int
kr+5.0	double
d+tam	double
f+d-2	double
utam-tam	unsigned
ltam*tam	long
tam/2	int
tam/2.0	double

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).

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

(tür tipi) bağıntı;

Örneğin:

```
int x=9;
float a,b,c;
double d;
...
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.

Ders 3: Operatörler

Giriş

Operatörler, değişkenler veya sabitler üzerinde matematiksel ve karşılaştırma işlemlerini yapan simgelerdir. Yani bir operatör bir veya daha fazla nesne (değişken) üzerinde işlem yapan sembollerdir. Bu kısımda aritmetik operatörler, atama operatörleri ve sizeof operatörü anlatılacaktır. Karşılaştırma Operatörleri, Mantıksal Operatörler ve Bit Düzeyinde işlem yapan operatörler daha sonraki bölümlerde incelenektir.

3.1 Aritmetik Operatörler

Değişken veya sabitler üzerinde temel aritmetik işlemleri gerçekleyen operatörlerdir. Bunlar Tablo 3.1'de listelenmiştir.

Tablo 3.1: *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

3.2 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 Tablo 3.2'de listelenmiştir.

Tablo 3.2: *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 artırma 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
```

Program 3.1: *Aritmetik ve atama operatörlerinin kullanımı*

```
01: /* 03prg01.c: Aritmetik ve atama operatorlerinin kullanimi
02: */
03:
04: #include <stdio.h>
05:
06:
07: main()
08: {
09:     int x, y; /* yerel degikenlerin bildirimi */
10:
11:     x = 1;    /* x in baslangic degeri */
12:     y = 3;    /* y nin baslangic degeri */
13:
14:     printf(" x = %d ve y = %d, olarak veriliyor.\n", x,
15: y);
16:
17:     x = x + y;
18:     printf("x <- x + y atamsinin sonucunda x=%d dir\n",
19: x);
20:
21:     x = 1;    /* x e tekrar 1 degeri ataniyor */
22:     x += y;
    printf("x += y atamasinin sonucunda x=%d dir\n", x);

    return 0;
}
```

ÇIKTI

```
x = 1 ve y = 3, olarak veriliyor.
x <- x + y atamasinin sonucunda x=4 dir
x += y atamasinin sonucunda x=4 dir
```

3.3 sizeof Operatörü

Veri tiplerinin, değişkenlerin ve dizilerin bellekte kapladığı alan `sizeof` operatörü ile öğrenilebilir. Genel kullanımı:

```
sizeof(nesne)
```

şeklinde. Program 3.2'de bu operatörün nasıl kullanıldığı gösterilmiştir.

Program 3.2: sizeof operatörünün kullanımı

```
01: /* 03prg02.c
02:     sizeof operatörünün değişik nesnelerle kullanımı */
03:
04: #include <stdio.h>
05:
06: int main() {
07:
08:     int    i;                /* bir tamsayı */
09:     int    dizi[5];          /* 5 elemanlı bir tamsayı dizi
10: */
11:
12:     double d;                /* bir gerçel sayı */
13:     double mizan[6];          /* 6 elemanlı bir gerçel dizi */
14:
15:     char    c;                /* tek bir karakter */
16:     char    str[] = "masa"; /* bir karakter topluluğu */
17:
18:
19:     printf("sizeof(int)    = %d\n", sizeof(int));
20:     printf("sizeof(i)     = %d\n", sizeof(i));
21:     printf("sizeof(dizi)  = %d\n\n", sizeof(dizi));
22:
23:     printf("sizeof(double)= %d\n", sizeof(double));
24:     printf("sizeof(d)     = %d\n", sizeof(d));
25:     printf("sizeof(mizan) = %d\n\n", sizeof(mizan));
26:
27:     printf("sizeof(char)  = %d\n", sizeof(char));
28:     printf("sizeof(c)     = %d\n", sizeof(c));
29:     printf("sizeof(str)   = %d\n", sizeof(str));
30:
31:     return 0;
32: }
```

ÇIKTI

```
sizeof(int)    = 4
sizeof(i)      = 4
sizeof(dizi)   = 20

sizeof(double)= 8
sizeof(d)      = 8
sizeof(mizan)  = 48

sizeof(char)   = 1
sizeof(c)      = 1
sizeof(str)    = 5
```

Programda `sizeof(int)` değeri ile `sizeof(i)` değerinin aynı olduğu görülür. dizinin boyutu 5 olduğu için, `sizeof(dizi) = sizeof(int)*5 = 20` şeklinde hesaplanmaktadır. Diğerleri için benzer durum söz konusu. Ancak, `str` 4 elemanlı bir dizi olduğu halde `sizeof(str) = 5` dir. Neden? Bunu ilerideki bölümlerde öğreneceğiz.

Ders 4: Temel Giriş/Çıkış Fonksiyonları

Giriş

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. Temel giriş/çıkış fonksiyonları kullanılırken `stdio.h` başlık dosyası programın başına eklenmelidir. Bu kısımda, en çok kullanılan giriş/çıkış fonksiyonları anlatılacaktır.

4.1 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. Daha önce yazılan örnek programlarda `printf()` fonksiyonundan yararlanmıştık. Şimdi bu fonksiyonun nasıl kullanıldığını bakalım.

Genel yazım biçimi:

```
int printf(const char *format, ...);
```

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

```
printf("Hata Oluştı!..");
```

şeklinde. Çoğu zaman ekrana, programda kullanılan bir değişkenin değeri yazdırılmak istenebilir. Örneğin ekrana bir tamsayı değişkeninin içeriğini basırmak için, `printf()`

```
....  
int x = 12;  
printf("x in değeri %d dir", x);  
....
```

gibi kullanılır. Bu program parçasının ekran çıktısı şöyle olacaktır:

```
x in değeri 12 dir
```

Bu örnekte `printf` fonksiyonuna iki parametre aktarılmıştır. Birincisi ekranda gösterilecek ve çift tırnaklar arasına yazılan ifadeler, ikincisi ise ekranda sayısal değeri gösterilmek istenen değişken (`x`).

`*format` üç kısımdan oluşmaktadır:

- I. **Düz metin (literal string):** yazdırılmak istenen ileti.
Örneğin: `printf("Ben gelmedim kavga için...");` gibi.
- II. **Konrol karakterleri (escape squence):** değişkenlerin ve sabitlerin nasıl yazılacağını belirtmek veya imlecin alt satıra geçirilmesi gibi bazı işlemlerin gerçekleştirilmesi için kullanılır. Bu karakterler Tablo 4.1'de listelenmiştir.
Örneğin: `printf("\tDostun evi gönlüdür...\n");` gibi.

Tablo 4.1: 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 (Bkz. Tablo 4.2) Örneğin: `printf("x in değeri %d dir");` gibi.

Tablo 4.2: Tip karakterleri

Tip Karakteri	Anlamı	Yazdırılacak veri tipi
%c	tek bir karakter	char
%s	karakter dizisi (string)	char
%d	işaretli ondalık tamsayı	int, short
%ld	uzun işaretli ondalık tamsayı	long
%u	işaretsiz ondalık tamsayı	unsigned int, unsigned short
%lu	işaretsiz uzun tamsayı	unsigned long
%f	Gerçel sayı	float
%lf	Çift duyarlı gerçel sayı	double

Tip karakterlerini kullanarak, birden çok veri tipi yazdırılabilir. Örneğin:

```
...
int    not= 12;
float  pi = 3.14;
char   kr = 'A';

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

gibi.

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

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

printf fonksiyonu kullanımı Program 4.1'de verilmiştir.

Program 4.1: printf() fonksiyonunun kullanımı

```
01: /* 04prg01.c
02:    Sayısal değerleri ekrana yazdırmak için printf
03:    fonksiyonunun kullanımı */
04:
05: #include <stdio.h>
06:
07: main()
08: {
09:
10:     int    a = 2,    b = 10,    c = 50;
11:     float  f = 1.05, g = 25.5, h = -0.1, yuzde;
12:
13:     printf("3 tamsayi      : %d %d %d\n", a, b, c);
14:     printf("3 tamsayi [TAB] : %d \t%d \t%d\n", a, b, c);
15:
16:     printf("\n");
17:
18:     printf("3 reel sayi (yanyana) : %f %f %f\n", f, g, h);
19:     printf("3 reel sayi (altalta) : \n%f\n%f\n%f\n\n", f,
20: g, h);
21:
22:     yuzde = 220 * 25/100.0;
23:     printf("220 nin %%25 i %f dir\n", yuzde);
24:     printf("%f/%f isleminin sonucu = %f\n", g, f, g / f);
25:
26:     printf("\nprogram sonunda beep sesi cikar...\a");
27:
    return 0;
}
```

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 cikar...
```

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. Program 4.2, printf'in bu yönünde ortaya çıkaran bir programdır.

Program 4.2: printf() fonksiyonunun kullanımı

```
01: /* 04prg02.c
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",
14: karSay);
15:
16:     return 0;
17: }
```

ÇIKTI

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

11. 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.

4.2 scanf() Fonksiyonu

Birçok programda ekrana verilerin bastırılmasının yanısıra klavyeden veri okunması gerekebilir. `scanf()` fonksiyonu klavyeden veri okumak için kullanılan fonksiyondur. `printf()` gibi `scanf()` fonksiyonunda Tablo 4.1 ve Tablo 4.2'de verilen karakterleri 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
```

yada

```
16      1.56
```

veya

```
16
1.56
```

şekilinde girilebilir. Program 4.3'de `scanf()` fonksiyonunun kullanımı gösterilmiştir.

Program 4.3: *scanf()* fonksiyonun kullanımı

```
01: /* 04prg03.c
02:     scanf() fonksiyonu ile int ve float tipindeki
03: verilerin okunması */
04:
05: #include <stdio.h>
06:
07: main()
08: {
09:     int    t;
10:     float  g;
11:
12:     printf("Bir gercel sayi girin: "); scanf("%f",&g);
13:     printf("Bir tamsayi girin      : "); scanf("%d",&t);
14:
15:     printf("\n");
16:
17:     printf("\t %f * %f = %f\n",g,g,g*g);
18:     printf("\t %d * %d = %d\n",t,t,t*t);
19:
20:     return 0;
}
```

ÇIKTI

```
Bir gercel sayi girin: 1.34
Bir tamsayi girin      : 12

    1.340000 * 1.340000 = 1.795600
    12 * 12 = 144
```

4.3 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("Sevgi varlığın mayasıdır.\n");
ile
puts("Sevgi varlığın mayasıdır.");
kullanımları eşdeğerdir.
```

`puts()` fonksiyonu Tablo 4.1 de verilen kontrol karakterleri ile kullanılabilir.

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

4.4 gets() Fonksiyonu

Klavyeden bir karakter topluluğu okumak için kullanılır. Okuma işlemi yeni satır karakteriyle(\n) karşılaşıncaya kadar sürer. `puts()` - `gets()` arasındaki ilişki, `printf()` - `scanf()` arasındaki gibidir. Yani,

```
scanf("%s",str);  
ile  
gets(str);
```

aynı anlamdadır. `puts()` - `gets()` fonksiyonlarının kullanımı daha sonra ayrıntılı işlenecektir.

4.5 getchar() Fonksiyonu

Bu fonksiyon ile standart girişten bir karakter okunur. Programı istenen bir yerde durdurup, bir karakter girinceye kadar bekletir. Örneğin:

```
for(i=0; i<10; i++)  
{  
    getchar();  
    printf("%d\n",i);  
}  
...
```

Yukarıdaki program parçası 0-9 arası sayıları sırasıyla ekranda göstermek için kullanılır. Fakat her rakamı yazdırılmadan önce klavyeden herhangi bir karakter girip [Enter] tuşuna basılması beklenir. Bu bekleme `getchar()` fonksiyonu ile gerçekleştirilir.

4.6 Formatlı Çıktı

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

Tamsayılarda `%d` yerine `%wd`
Gerçel sayılarda `%f` yerine `%w.kf`
Stringlerde `%s` yerine `%ws`

biçimindeki kullanım ile sağlanır. Burada `w` yazılacak olan sayının *alan genişliği* olarak adlandırılır. Gerçel bir değişken ekrana yazılacaksa, değişkenin virgülden sonra kaç basamağının yazdırılacağı `ksayı`sı ile belirlenir. Ancak `w > k + 2` olmalıdır.

```
int i=583,j=1453;  
  
printf("%d %d\n",i,j);    /* serbest biçim */  
printf("%5d %8d\n",i,j); /* formatlı */
```

program parçasının ekran çıktısı şöyledir:

ÇIKTI

```
583 1453
583    1453
```

Birinci satır serbest formatta ikinci satır ise formatlı yazılmıştır. `i` değişkeninin tuttuğu 583 sayısı `%5d` formatıyla yazdırılınca, bu sayı için 5 alan genişliği tanımlanır arakasından sağdan başlayarak sayı bu alana yazılır. Benzer olarak `j` değişkeni, 8 alan genişlikli bir bölgeye yazılır.

Gerçek sayılarda iş biraz daha karışık. Örneğin:

```
int x=123.456;

printf("%f\n",x);    /* serbest biçim */
printf("%.2f\n",x); /* formatlı */
```

program parçası çalıştırıldığında aşağıdaki sonuç gözlenir:

ÇIKTI

```
123.456001
123.46
```

Birinci satır serbest formatta ikinci satır ise formatlı yazılmıştır. İkinci satırda `x` değişkeni için ayrılan alan genişliği 8 ve noktadan sonra 2 basamağa kadar hassasiyet önemsenmiştir. Dikkat edilirse noktadan sonra sayı uygun bir şekilde yuvarlanmış ve sayı sağa dayalı olarak yazılmıştır.

Program 4.4: `printf()` in formatlı kullanımı

```
01: /* 04prg04.c: Formatlı çıktı */
02:
03: #include <stdio.h>
04:
05: main()
06: {
07:     float  x = 7324.25, y = 244.531;
08:     int     i = 1299;
09:     char   *c = "Merhaba C";
10:
11:     printf("%10d\n", i);
12:     printf("%10s\n", c);
13:
14:     printf("%.5f\n", x);
15:     printf("%.1f\n", y);
16:
17:     return 0;
18: }
19:
```

ÇIKTI

```
      1299
Merhaba C
7324.25000
      244.5
```

Ders 5: Temel Kütüphane Fonksiyonlar

Giriş

Bu kısımda, C Programlama Dili'nde sık kullanılan ve diğer bölümlerde yararlanacağımız kütüphane fonksiyonlarının bazıları işlenecektir. Kütüphane fonksiyonu C dilinde önceden tanımlanmış hazır fonksiyonlardır. C dilinde birçok iş bu fonksiyonlarla yapılmaktadır.

Her kütüphane fonksiyonu bir başlık dosyasında tanımlanmıştır. Bu yüzden bir kütüphane fonksiyonunu kullanmadan önce, onun hangi başlık dosyası ile kullanılması gerektiğini bilmelisiniz.

5.1 Matematiksel Fonksiyonlar (math.h)

Matematiksel fonksiyonların hemen hemen hepsi `double` veri tipindedir. Bu fonksiyonlardan biri program içinde kullanılacaksa `math.h` başlık dosyası program içine eklenmelidir. En çok kullanılan matematiksel fonksiyonlar Tablo 5.1'de listelenmiştir.

Tablo 5.1: *math.h* kütüphanesinde tanımlı bazı fonksiyonlar

Fonksiyon Bildirimi	Açıklama	Örnek	Sonuç
<code>int abs(int x);</code>	x tamsayısının mutlak değerini hesaplar	<code>abs(-4)</code>	4
<code>double fabs(double x);</code>	x gerçel sayısının mutlak değerini hesaplar	<code>fabs(-4.0)</code>	4.000000
<code>int floor(double x);</code>	x'e (x'den büyük olmayan) en yakın tamsayıyı gönderir	<code>abs(-0.5)</code>	-1
<code>int ceil(double x);</code>	x'e (x'den küçük olmayan) en yakın tamsayıyı gönderir	<code>ceil(-0.5)</code>	0
<code>double sqrt(double x);</code>	pozitif x sayısının karekökünü hesaplar	<code>sqrt(4.0)</code>	2.000000
<code>double pow(double x, double y);</code>	x^y değerini hesaplar	<code>pow(2., 3.)</code>	8.000000
<code>double log(double x);</code>	pozitif x sayısının doğal logaritmasını hesaplar, $\ln(x)$	<code>log(4.0)</code>	1.386294
<code>double log10(double x);</code>	pozitif x sayısının 10 tabanındaki logaritmasını hesaplar	<code>log10(4.0)</code>	0.602060
<code>double sin(double x);</code>	radyan cinsinden girilen x sayısının sinüs değerini hesaplar	<code>sin(3.14)</code>	0.001593
<code>double cos(double x);</code>	radyan cinsinden girilen x sayısının kosinüs değerini hesaplar	<code>cos(3.14)</code>	-0.999999
<code>double tan(double x);</code>	radyan cinsinden girilen x sayısının tanjant değerini hesaplar	<code>tan(3.14)</code>	-0.001593
<code>double asin(double x);</code>	sinüs değeri x olan açıyı gönderir. Açı $-\pi/2$ ile $\pi/2$ arasındadır.	<code>asin(0.5)</code>	0.523599
<code>double acos(double x);</code>	cosinüs değeri x olan açıyı gönderir. Açı $-\pi/2$ ile $\pi/2$ arasındadır.	<code>acos(0.5)</code>	1.047198
<code>double atan(double x);</code>	tanjant değeri x olan açıyı gönderir. Açı $-\pi/2$ ile $\pi/2$ arasındadır.	<code>atan(0.5)</code>	0.463648

NOT

Bir programda `math.h` kütüphanesi kullanılacakca, GCC derleyicisi -
`lm` seçeneği ile birlikte kullanılmalıdır.

Örneğin `test.c` içinde `math.h`'i kullanıyorsa derleme:

```
gcc -lm test.c -o test
```

şeklinde yapılmalıdır. Aksi halde bir hata mesajı ile karşılaşılır.

Trigonometrik (`sin`, `cos`, `tan`) fonksiyonlar kendisine parametre olarak gelen değeri radyan olarak kabul eder ve sonucu hesaplar. Eğer açılar derece cinsinden hesaplanması gerekiyorsa şu dönüşüm kullanılabılır:

```
radyan = (3.141593/180.0) * derece;
```

Program 5.1: *`sin()`, `cos()`, and `tan()` fonksiyonlarının kullanımı*

```
01: /* 05prg01.c
02:    30 dercelik açının sinüs, kosinüs, tanjant ve kotanjant
03:    değerleri */
04:
05: #include <stdio.h>
06: #include <math.h>
07:
08: #define PI      3.141593
09:
10: int main()
11: {
12:     double aci = 30.0;
13:
14:     aci *= PI/180.0;    /* radyana çevir */
15:
16:     puts("30 derecenin");
17:     printf("sinusu      : %lf\n", sin(aci));
18:     printf("kosinusu    : %lf\n", cos(aci));
19:     printf("tanjanti    : %lf\n", tan(aci));
20:     printf("kotanjanti: %lf\n", 1.0/tan(aci));
21:
22:     return 0;
}
```

ÇIKTI

```
30 derecenin
sinusu      : 0.500000
kosinusu    : 0.866025
tanjanti    : 0.577350
kotanjanti: 1.732051
```

5.2 Standart Kütüphane Fonksiyonları (stdlib.h)

Standart kütüphanede, programı sonlandıran, dinamik bellek yönetiminde kullanılan veya rastgele sayı üretme vb. işlevleri yerine getiren bir çok fonksiyon mevcuttur. Bu kısımda, bunlardan bir kaçını Tablo 5.2'de listelenmiştir.

Tablo 5.2: *stdlib.h* kütüphanesinde tanımlı bazı fonksiyonlar

Fonksiyon Bildirimi	Açıklama	Örnek	Sonuç
<code>int atoi(const char *s);</code>	Bir karakter topluluğunu tamsayıya çevirir	<code>atoi("-12345")</code>	-12345
<code>long atol(const char *s);</code>	Bir karakter topluluğunu uzun tamsayıya çevirir	<code>atol("1234567890")</code>	1234567890
<code>double atof(const char *s);</code>	Bir karakter topluluğunu gerçel sayıya çevirir	<code>atof("-123.546")</code>	-123.456
<code>void exit(int durum);</code>	Programı sonlandırarak kontrolü işletim sistemine geri verir.	<code>exit(1)</code>	-
<code>int rand(void);</code>	0 ile RAND_MAX arasında rastgele sayı üretir. RAND_MAX, stdlib.h içinde tanımlanmış bir sembolik sabittir	<code>rand()</code>	50485132
<code>max(a,b)</code>	stdlib.h'de tanımlanmış iki sayıdan en büyüğünü bulan makro fonksiyon	<code>max(5, 9)</code>	9
<code>min(a,b)</code>	stdlib.h'de tanımlanmış iki sayıdan en küçüğünü bulan makro fonksiyon	<code>min(5, 9)</code>	5

Program 5.2: *rand()* fonksiyonu kullanımı

```
01: /* 05prg02.c
02:    0-100 arasında 10 tane rasgele sayı üretir */
03:
04: #include <stdio.h>
05: #include <stdlib.h>
06:
07: int main()
08: {
09:     int i, ri;
10:
11:     for(i=1; i<=10; i++)
12:     {
13:         ri = rand() % 100;          /* 0-100 arası
14: tamsayı */
15:         printf("%d\t%d\n",i,ri);
16:     }
17:     puts("10 tane rasgele sayi uretildi.");
18:
19: return 0;
}
```

ÇIKTI

```
1      83
2      86
3      77
4      15
5      93
6      35
7      86
8      92
9      49
10     21
```

5.3 Karakter Üzerinde İşlem Yapan Fonksiyonlar (ctype.h)

Tablo 5.3: *ctype.h* Kütüphanesinde tanımlı fonksiyonlar

Fonksiyon Bildirimi	Açıklama	Örnek	Sonuç
isalpha(c)	c bir harf ise 0 dan farklı, değilse 0 gönderir	isalpha('a')	8
isalnum(c)	c A-Z, a-z veya 0-9 arasında ise 0 dan farklı, değilse 0 gönderir	isalnum('a')	1
isascii(c)	c bir ASCII karakter ise 0 dan farklı, değilse 0 gönderir	isascii('a')	1
isdigit(c)	c bir rakam ise 0 dan farklı, değilse 0 gönderir	isdigit('4')	2
islower(c)	c a-z arasında ise 0 dan farklı, değilse 0 gönderir	islower('P')	0
isupper(c)	c A-Z arasında ise 0 dan farklı, değilse 0 gönderir	islower('P')	4
toascii(c)	c sayısı ile verilen ASCII koda sahip karakteri elde eden makro	toascii(65)	A
tolower(c)	c karakterini küçük harfe çevirir	tolower('D')	d
toupper(c)	c karakterini büyük harfe çevirir	toupper('b')	B

Program 5.3: *ctype.h* kütüphanesinde bulunan bazı makroların kullanımı

```
01: /* 05prg03.c
02:     ASCII kodları 32-127 arasında olan karakterler üzerinde
03:     ctype.h kütüphanesinde tanımlı bazı makroların
04:     kullanımı */
05:
06: #include <stdio.h>
07: #include <ctype.h>
08:
09: int main(void)
10: {
11:     int i;
12:     char c;
13:
14:     for(i=32; i<127; i++)
15:     {
16:         c = toascii(i);
17:         printf("%d\t%c\t%c\t%d\n",
18: i,c,tolower(c),isdigit(c));
19:     }
20:
21:     return 0;
22: }
```

ÇIKTI

32			0
33	!	!	0
34	"	"	0
35	#	#	0
36	\$	\$	0
37	%	%	0
38	&	&	0
39	'	'	0
40	(	(	0
41	)	)	0
42	*	*	0
43	+	+	0
44	,	,	0
45	-	-	0
46	.	.	0
47	/	/	0
48	0	0	1
49	1	1	1
50	2	2	1
51	3	3	1
52	4	4	1
53	5	5	1
54	6	6	1
55	7	7	1
56	8	8	1
57	9	9	1
58	:	:	0
59	;	;	0
60	<	<	0
61	=	=	0
62	>	>	0
63	?	?	0
64	@	@	0
65	A	a	0
66	B	b	0
67	C	c	0
68	D	d	0
69	E	e	0
70	F	f	0
71	G	g	0
72	H	h	0
73	I	i	0
74	J	j	0
75	K	k	0
76	L	l	0
77	M	m	0
78	N	n	0
79	O	o	0
80	P	p	0
81	Q	q	0
82	R	r	0
83	S	s	0
84	T	t	0
85	U	u	0
86	V	v	0
87	W	w	0
88	X	x	0
89	Y	y	0
90	Z	z	0
91	[	[	0
92	\	\	0
93	]	]	0
94	^	^	0
95	~	~	0
96			0
97	a	a	0
98	b	b	0
99	c	c	0

100	d	d	0
101	e	e	0
102	f	f	0
103	g	g	0
104	h	h	0
105	i	i	0
106	j	j	0
107	k	k	0
108	l	l	0
109	m	m	0
110	n	n	0
111	o	o	0
112	p	p	0
113	q	q	0
114	r	r	0
115	s	s	0
116	t	t	0
117	u	u	0
118	v	v	0
119	w	w	0
120	x	x	0
121	y	y	0
122	z	z	0
123	{	{	0
124			0
125	}	}	0
126	~	~	0

Ders 6: Karşılaştırma Deyimleri

Giriş

Program içerisinde bazen iki veya daha fazla değerin karşılaştırılması gerekebilir. Bunun için, bütün programlama dillerinde karşılaştırma deyimleri mevcuttur. C dili, `if`, `switch` ve `?` olmak üzere üç tip karşılaştırma işlemi yapmaya izin verir. Ancak `?` bir operatördür. `if` karşılaştırma deyimi ile, diğer programlama dilinde olduğu gibi `if-else` yapısı da kurulabilir. `switch` deyimi, bir değişkenin içeriğine göre program akışını yönlendirir.

6.1 Karşılaştırma Operatörleri ve Mantıksal Operatörler

Tablo 6.1'de listelenen Karşılaştırma Operatörleri, sayısal değerleri veya karakterleri mukayese etmek için kullanılır.

Tablo 6.1: 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, Tablo 6.2'deki Mantıksal Operatörler'le birleştirilebilir.

Tablo 6.2: 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. Buna göre, aşağıdaki program parçasının

```
...
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 sağtarafının sonucu olumlu olduğu için z değişkenine 1 atanır.

6.2 if, if-else Yapısı

Bu deyimler, koşullu işlem yapan deyimlerdir. `if` ve `else` tek bir karşılaştırma deyimini olup `else` kullanımı isteğe bağlıdır. Eğer bu koşul olumlu ise `if` den sonraki bölüm yürütülür ve `else` den sonraki bölüm atlanır. Koşul olumsuz ise `if` den sonraki küme atlanır ve eğer varsa, `else` den sonraki kümedeki işlemler gerçekleştirilir.

`if` deyiminin yapının genel biçimi şöyledir:

```
if (koşul)
```



```
{
    ...
    deyimler; (küme)
    ...
}
```

`if` deyimi kullanılırken kümenin başlangıcı ve bitişini gösteren, küme parantezleri kullanılmasında kullanıcıya bir esneklik sunulmuştur. Eğer `if` deyiminden sonra icra edilecek deyimler tek satırdan oluşuyorsa, bu işaretlerin kullanılması zorunlu değildir. Yani, `if` deyimden sonra `{ ve }` işaretleri kullanılmamışsa, bu deyimi takip eden sadece ilk satır işleme konur. 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.

`if` deyiminin `else` ile birlikte kullanımı şu şekildedir:

```
if(koşul){
    ...
    deyimler; (küme1)
    ...
}

else{
    ...
    deyimler; (küme2)
    ...
}
```

Bu yapının kullanılmasına dair bir örnek Program 6.1'de gösterilmiştir. Program, klavyeden girilen bir tamsayının çift olup olmadığını sınar. Bilindiği gibi, çift sayılar, 2 ile kalansız bölünebilen sayılardır.

Program 6.1: *if-else* deyiminin kullanımı

```
01: /* 06prg01.c
02:     Klavyeden girilen bir sayının çift olup olmadığını
03:     sınırlar. */
04:
05: #include <stdio.h>
06:
07: int main()
08: {
09:     int sayi;
10:
11:     printf("Bir sayi girin: ");
12:     scanf("%d",&sayi);
13:
14:
15:     if (sayi % 2 == 0)
16:         printf("sayi cifttir.\n");
17:     else
18:         printf("sayi tektir.\n");
19:
20:     return 0;
}
```

ÇIKTI

```
Bir sayi girin: 3
sayi tektir.
```

Mantıksal Operatörler kullanarak birden çok karşılaştırma birleştirilebilir. Buna iyi bir örnek Program 6.2'de gösterilmiştir. Program, bir yılın artık yıl olup olmadığını sınırlar. Bir yıl içinde, Şubat ayı 29 gün olursa o yıl artık yıl olarak adlandırılır. Artık yıl periyodik olarak 4 yılda bir gelir. 1800 artık yıl değildir. Genel sorgulama şöyle olmalıdır: Eğer bir yıl

- 4 ile tam bölünüyorsa VE 100'e tam bölünmüyorsa VEYA
- 400 'e tam bölünüyorsa

o yıl artık yıldır.

Program 6.2: *if-else* deyiminin kullanımı

```
01: #include <stdio.h>
02:
03: void main()
04: {
05:     int yil;
06:
07:     printf("Bir yil girin: ");
08:     scanf("%d",&yil);
09:
10:     if( yil % 4 == 0 && yil % 100 != 0 || yil % 400 == 0 )
11:         printf("%d artik yil\n",yil);
12:
13:     else
14:         printf("%d artik yil degil\n",yil);
15: }
```

ÇIKTI

```
Bir yıl girin: 1996
1996 artık yıl
```

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

```
if(koşul_1)
{
    ...
    deyimler; (küme_1)
    ...
}
else if(koşul_2)
{
    ...
    deyimler; (küme_2)
    ...
}
.
.
.
else if(koşul_n-1)
{
    ...
    deyimler; (küme_n-1)
    ...
}
else
{
    ...
    deyimler; (küme_n)
    ...
}
```

Program 6.3, $ax^2 + bx + c = 0$ formundaki ikinci dereceden bir polinomun köklerini hesaplamaktadır. Programda `delta` değerinin sıfırdan küçük olması durumunda köklerin karmaşık sayıya dönüşeceğide göz önüne alınmıştır. Bu program `if`, `else if` ve `else` yapısı göstermek için klasik bir örnektir.

Program 6.3: if, else if, else yapısı

```
01:  /*  ax*x + bx + c = 0 denkleminin (karmaşık sayılı kökler
02:  dahil) çözümü */
03:
04:  #include <stdio.h>
05:  #include <math.h>
06:
07:  int main()
08:  {
09:      float a, b, c, delta, x1, x2, x, kok_delta;
10:
11:      printf("a, b, c degerlerini girin:\n");
12:      scanf("%f %f %f",&a,&b,&c);
13:
14:      delta = b*b - 4.0*a*c;
15:
16:      if( delta > 0.0 ){
17:          x1 = ( -b + sqrt(delta) )/( 2.0*a );
18:          x2 = ( -b - sqrt(delta) )/( 2.0*a );
19:
20:          printf("\nReel kokler:");
21:          printf("\nx1 = %f",x1);
22:          printf("\nx2 = %f",x2);
23:      }
24:      else if( delta < 0.0 ){
25:          kok_delta = ( sqrt(-delta) ) / (2.0*a);
26:          x = -0.5*b/a;
27:
28:          printf("\nKarmasik kokler:");
29:          printf("\nx1 = %f + (%f)i", x, kok_delta);
30:          printf("\nx2 = %f - (%f)i", x, kok_delta);
31:      }
32:      else{
33:          x = -0.5*b/a;
34:
35:          printf("\nKokler eşit:");
36:          printf("\nx1 = x2 = %f",x);
37:      }
38:
39:      return 0;
40:  }
```

ÇIKTI

a, b, c degerlerini girin:

2 4 -8

Reel kokler:

x1 = 1.236068

x2 = -3.236068

ÇIKTI

a, b, c degerlerini girin:

1 1 1

Karmasik kokler:

x1 = -0.500000 + (0.866025)i

x2 = -0.500000 - (0.866025)i

6.3 switch - case Yapısı

Bu deyim 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 (deyimler) işleme konur. Bütün durumların aksi söz konu olduğunda gerçekleştirilmesi istenen deyimler *default* deyiminden sonraki kısımda bildirilir. Genel yazım biçimi:

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

Program Program 6.4'te *switch* deyiminin basit bir kullanımı gösterilmiştir.

Program 6.4: *switch-case yapısının kullanımı*

```
01: /* 06prg04.c: switch - case yapısının kullanımı */
02:
03: #include <stdio.h>
04:
05: int main(void)
06: {
07:     char kr;
08:
09:     printf("Lutfen bir karakter girin\n");
10:
11:     kr = getchar(); /* tek bir karakterin okunması */
12:
13:     switch (kr)
14:     {
15:         case 'a':
16:             printf("a harfine bastınız\n");
17:         case 'b':
18:             printf("b harfine bastınız\n");
19:         default:
20:             printf("a veya b ye basmadınız\n");
21:     }
22:
23:     return 0;
24: }
```

ÇIKTI

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

ÇIKTI

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

ÇIKTI

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

ÇIKTI

```
Lütfen bir karakter girin
c
a veya b ye basmadiniz
```

Programda, klavyeden okunan tek bir karakter değişkenin içeriğine bakılıp uygun dallanmalar yaptırılmıştır. 11. satırda değişken `getchar()` fonksiyonu ile okutulmuştur. Eğer 'a' veya 'b' karakterlerinden biri girilirse, ekrana bu harflerin girildiğine dair mesaj yazılacak, aksi takdirde bu karakterin dışında bir karakterin giriş olarak kullanıldığı gösteren bir mesaj yazılacaktır. Örneğin 'c' karakteri klavyeden girilmiş ise `a veya b ye basmadiniz` gibi. Fakat 'a' karakterleri girildiğinde ekrana her üç durumda yazdırılmaktadır. Bunun sebebi, `case 'a':` durumunda sırasıyla 16, 18 ve 20. satırların işleme konmasıdır. Bunu engellemek için 16. satırdan sonra programın başka bir yere yönlendirilmesi gerekir. Bu yönlendirme `break` deyimi ile yapılır. Derleyici bu deyim ile karşılaştığında, bulunduğu yapının içinden koşulsuz olarak ayrılır ve takip eden işleme başlar.

Program 6.4'te `case 'a':` durumu için 16, 18 ve 20. satırlar da işleme konmuştu. Eğer klavyeden 'a' karakterini girip ekrana sadece `a harfine bastiniz` iletisi yazdırılmak isteniyorsa, 20. satıra `break` deyimi ilave edilmelidir. `break` deyiminin kullanımı Program 6.5'te gösterilmiştir.

Program 6.5: switch-case yapısı ve break kullanımı

```
01: /* 06prg05.c: switch - case yapısı ve break kullanımı */
02:
03: #include <stdio.h>
04:
05: int main(void)
06: {
07:     char kr;
08:
09:     printf("Lutfen bir karakter girin\n");
10:
11:     kr = getchar(); /* tek bir karakterin okunması */
12:
13:     switch (kr)
14:     {
15:         case 'a':
16:             printf("a harfine bastınız\n");
17:             break;
18:         case 'b':
19:             printf("b harfine bastınız\n");
20:             break;
21:         default:
22:             printf("a veya b ye basmadınız\n");
23:             break;
24:     }
25:
26:     return 0;
27: }
```

ÇIKTI

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

ÇIKTI

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

Program 6.6 switch-case yapısının farklı bir kullanımı ile ilgili bir örnektir. Programda, önce iki sayı isteniyor ardından yapılan seçimle bu sayıların toplamı, farkı, çarpımı veya oranı ekrana yazdırılıyor.

Program 6.6: switch-case yapısı ve break kullanımı

```
01: /* 06prg06.c: switch-case yapısı */
02:
03: #include <stdio.h>
04: #include <stdlib.h>
05:
06: int main(void)
07: {
08:     int    secim;
09:     float  x,y, sonuc;
10:
11:     printf("İki sayı girin: ");
12:     scanf("%f %f",&x,&y);
13:
14:     puts("*** Menu ***");
15:     puts("[1] Toplama");
16:     puts("[2] Cikarma");
17:     puts("[3] Carpma");
18:     puts("[4] Bolme");
19:
20:     printf("Seciminiz: ");
21:     scanf("%d",&secim);
22:
23:     switch( secim )
24:     {
25:         case 1:
26:             sonuc = x + y;
27:             printf("Toplam = %f\n",sonuc);
28:             break;
29:         case 2:
30:             sonuc = x-y;
31:             printf("Fark = %f\n",sonuc);
32:             break;
33:         case 3:
34:             sonuc = x * y;
35:             printf("Carpim = %f\n",sonuc);
36:             break;
37:         case 4:
38:             sonuc = x/y;
39:             printf("Oran = %f\n",sonuc);
40:             break;
41:         default:
42:             puts("Yanlis secim !\a");
43:     }
44:
45:     return 0;
46: }
```

ÇIKTI

```
İki sayı girin: 3 8
*** Menu ***
[1] Toplama
[2] Cikarma
[3] Carpma
[4] Bolme
Seciminiz: 1
Toplam = 11.000000
```


ÇIKTI

```
Iki sayi girin: 3 8
*** Menu ***
[1] Toplama
[2] Cikarma
[3] Carpma
[4] Bolme
Seciminiz: 7
Yanlis secim !
```

switch-case yapısı if-else yapısının bir alternatifidir. Yani, Program 6.6'daki switch-case kısmı, if-else yapısı ile de aşağıdaki gibi yazılabilirdi. İnceleyiniz.

```
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");
}
```

```
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");
}
```

6.4 ? Karşılaştırma Operatörü

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

```
(koşul) ? deyim1 : deyim2;
```

İlk önce koşul sınanır. Eğer koşul olumluysa 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. Aşağıda bu deyimin kullanımına ait örnekler verilmiştir.

```
x = ( a > b ) ? a : b;
```

Yukarıdaki ifadede koşul a'nın b'den büyük olmasıdır. Eğer olumluysa x adlı değişkene a, değilse b değeri atanır. Bu şekilde kullanım if-else yapısı ile kurulmak istenirse:

```
if( a > b )   x = a;
else         x = b;
```

şeklinde olacaktır. Program 6.7 ? karşılaştırma operatörünün basit bir kullanımını göstermektedir.

Program 6.7: ? ve if kullanımı

```
01: /* 06prg07.c: ? ve if-else yapısının kullanımı */
02:
03: #include <stdio.h>
04:
05: int main(void)
06: {
07:     float x, y, z;
08:
09:     printf("x : "); scanf("%f",&x);
10:     printf("y : "); scanf("%f",&y);
11:
12:     if(y)                                /* y, 0'dan farklı mı?
13: */
14:         z = ( y > x ) ? x/y : x*y; /* y>x ise z = x/y,
15: değilse z = x*y */
16:     else
17:         z = 0.0;
18:
19:     printf("z = %f\n",z);
20:
21:     return 0;
22: }
```

ÇIKTI

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

ÇIKTI

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

12. satırdaki if deyimindeki koşul biraz farklıdır. Genel olarak koşul bu şekilde bildirilirse, koşulun 0 dan farklı olup olmadığı sınanır. Yani:

```
if(y)
ile
if( y != 0 )
aynı anlamdadır.
```

Bu kullanım çok yagındır. Eğer y, 0 dan farklı ise koşul olumlu olarak değerlendirilecektir. 13. satırda ? ile bir sinama yapılmaktadır. Eğer y, x den büyük ise z değişkenine x/y, aksi takdirde x*ydeğeri atanmaktadır. Eğer y = 0 ise z değişkenine 0 değeri atanmaktadır.

Ders 7: Döngüler

Giriş

Döngü (loop) deyimleri, bir kümenin belli bir koşul altında tekrar edilmesi için kullanılır. C programlama dilinde, `while`, `do...while` ve `for` olmak üzere üç tip döngü deyimi vardır. Diğer programlama dillerinde olduğu gibi, bu deyimlerle istenildiği kadar iç-içe döngü yapısı kullanılabilir.

7.1 while Döngüsü

Tekrarlama deyimidir. Bir küme ya da deyim `while` kullanılarak bir çok kez yinelenebilir. Yinelenmesi için koşul sınaması döngüye girilmeden yapılır. Koşul olumlu olduğu sürece çevrim yinelenir. Bu deyim kullanımı Program 7.1 de gösterilmiştir. Genel yazım biçimi:

```
while(koşul)
{
    ...
    döngüdeki deyimler; [küme]
    ...
}
```

Program 7.1: while döngüsü

```
01: /* 07prg01.c: while döngüsü */
02:
03: #include <stdio.h>
04:
05: main()
06: {
07:     int x=0;
08:
09:     while(x <= 10)
10:         printf("%d\n",x++);
11:
12:     return 0;
13: }
```

ÇIKTI

```
0
1
2
3
4
5
6
7
8
9
10
```

Program 7.1, 0-10 arasındaki sayıları ekrana yazdırmaktır. 9. satırdaki `while` deyiminden sonra `{` işareti kullanılmamıştır. Bu durumda, sadece takip eden satır (10. satır) döngünün içine dahil edilir.

7.2 do ... while Döngüsü

Bu deyimin `while` deyiminden farkı, koşulun döngü sonunda sınanmasıdır. Yani koşul sınanmadan döngüye girilir ve döngü kümesi en az bir kez yürütülür. Koşul olumsuz ise döngüden sonraki satıra geçilir. Bu deyimin kullanımı Program 7.2 de gösterilmiştir. Genel yazım biçimi:

```
do{
    ...
    döngüdeki deyimler;
    ...
}while(koşul);
```

Program 7.2: do-while döngüsü

```
01: /* 07prg02.c: do-while yapısı */
02:
03: #include <stdio.h>
04:
05: main()
06: {
07:     int sayi;
08:
09:     do
10:     {
11:         printf("Bir sayi girin : ");
12:         scanf("%d",&sayi);
13:         printf("iki kati      : %d\n",2*sayi);
14:
15:     }while( sayi>0 );    /* koşul */
16:
17:     puts("Döngü sona erdi.");
18:
19:     return 0;
20: }
```

ÇIKTI

```
Bir sayi girin : 1
iki kati      : 2
Bir sayi girin : 3
iki kati      : 6
Bir sayi girin : 4
iki kati      : 8
Bir sayi girin : -3
iki kati      : -6
Cevrim sona erdi.
```

15. satırdaki koşul olumlu olduğu sürece (`sayi>0` olduğu sürece), klavyeden yeni bir değer 12. satırda okunur. Aksi takdirde (`sayi<=0` ise) çevrimin sona erdiğine dair mesajla program sonlanır.

7.3 for Döngüsü

Bu deyim, diğer döngü deyimleri gibi bir kümeyi bir çok kez tekrarlamak için kullanılır. Koşul sınaması `while` da olduğu gibi döngüye girmeden yapılır. Bu döngü deyiminde diğerlerinden farklı olarak başlangıç değeri ve döngü sayacına sahip olmasıdır. Bu deyim kullanımı Program 7.3 de gösterilmiştir Genel yazım biçimi:

```
for( başlangıç ; koşul ; artım )
{
    ...
    döngüdeki deyimler;
    ...
}
```

Program 7.3: *for döngüsü*

```
01: /* 07prg03.c: for döngüsü ile faktoriyel hesabı. */
02:
03: #include <stdio.h>
04:
05: int main()
06: {
07:     long i, n, faktor;
08:
09:     printf("Faktoriyeli hesaplanacak sayı girin : ");
10:     scanf("%ld",&n);
11:
12:     faktor=1;
13:     for(i=1; i<=n; i++){
14:         faktor *= i;      /* n! = 1 x 2 x 3 x ... x n */
15:     }
16:
17:     printf("%ld! = %ld\n", n, faktor);
18:
19:     return 0;
20: }
```

ÇIKTI

```
Faktoriyeli hesaplanacak sayı girin : 4
4! = 24
```

ÇIKTI

```
Faktoriyeli hesaplanacak sayı girin : 15
15! = 2004310016
```

Program da faktoriyel hesabı 16. satırda gerçekleştirilmiştir. Faktöriyel, bilindiği gibi $n! = 1 \times 2 \times 3 \times \dots \times n$ tanımlanır. Gerçekte $15! = 1307674368000$ olmasına rağmen, program $15! = 2004310016$ olarak hesaplamıştır. Sizce bunun sebebi nedir?

Program 7.3'de döngüye girilmeden, `faktor = 1` atması yapılmıştır.

```
faktor = 1;
for(i=1; i<=n; i++)
    faktor *= i;
```

Bu döngü öncesi ilk değer ataması, döngünün başlangıç kısmında şu şekilde de yapılabilir:

```
for(faktor=1, i=1; i<=n; i++)
    faktor *= i;
```

`printf` fonksiyonu ile desimal (taban-10) sayılarının nasıl yazdırılacağı bundan önceki kısımlarda gösterilmişti. Program 7.4'te 0-15 arası desimal sayıların Oktal (taban-8) ve Heksadesimal (taban-16) karşılıkları ile `printf` kullanılarak yazdırılması gösterilmiştir.

Program 7.4: Sayı sistemi

```
01: /* 07prg04.c: Sayı sistemi
02:    %d : desimal      10 tabanındaki sayı
03:    %o : oktal       8 tabanındaki sayı
04:    %x : hexadesimal 16 tabanındaki sayı (küçük harf)
05:    %X : hexadesimal 16 tabanındaki sayı (büyük harf) */
06:
07: #include <stdio.h>
08:
09: int main()
10: {
11:     int i;
12:
13:     for (i=0; i<16; i++)
14:         printf("%2d  %2o  %x  %X\n", i, i, i, i);
15:
16:     return 0;
17: }
```

ÇIKTI

0	0	0	0
1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4
5	5	5	5
6	6	6	6
7	7	7	7
8	10	8	8
9	11	9	9
10	12	a	A
11	13	b	B
12	14	c	C
13	15	d	D
14	16	e	E
15	17	f	F

7.4 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.

Üç basamaklı, basamaklarının küpleri toplamı kendisine eşit olan tam sayılara Armstrong sayı denir. Örneğin: 371 bir Armstrong sayıdır çünkü $3^3 + 7^3 + 1^3 = 371$. Program 7.5'de iç içe geçmiş üç `for` döngüsü ile bütün Armstrong sayıları bulup ekrana yazar. İnceleyiniz.

Program 7.5: iç-içe `for` döngüleri

```
01: /* 07prg05.c:
02:    Üç basamaklı, basamaklarının küpleri toplamı kendisine
03:    eşit olan tam
04:    sayılara Armstrong sayı denir. Örneğin: 371 = 3^3 +
05:    7^3 + 1^3.
06:    Bu program iç-içe geçmiş 3 döngü ile bütün Armstrong
07:    sayıları bulur. */
08:
09: #include <stdio.h>
10:
11: int main()
12: {
13:     int a,b,c, kup, sayi, k=1;
14:
15:     for(a=1; a<=9; a++)
16:     for(b=0; b<=9; b++)
17:     for(c=0; c<=9; c++)
18:     {
19:         sayi = 100*a + 10*b + c;          /* sayi = abc (üç
20: basamaklı) */
21:         kup  = a*a*a + b*b*b + c*c*c;    /* kup  =
22: a^3+b^3+c^3 */
23:
24:         if( sayi==kup ) printf("%d. %d\n",k++,sayi);
25:     }
26:
27:     return 0;
28: }
```

ÇIKTI

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

7.5 Sonsuz Döngü

Bir döngü işlemini sonsuz kere tekrarlırsa bu döngü sonsuz döngü olarak adlandırılır. Böyle bir döngü için, koşul çok önemlidir. Örneğin `while` döngüsü için:

```
...
while(1)
{
    printf("Sonsuz döngü içindeyim...\n");
}
...
```

yada

```
...
while(7>3)
{
    printf("Sonsuz döngü içindeyim...\n");
}
...
```

Her iki durumda da çevrimler, sonsuz döngü durumundadır.

Çünkü `while(1)` ve `while(7>3)` ifadelerdeki koşullar daima olumludur. Bu durumda çevrim sonsuz döngüye girer.

`for` döngüsünde, başlangıç, koşul ve artım parametrelerinden herhangi birini kullanmak isteğe bağlıdır. Her hangi biri verilmediğinde döngünün nasıl davranacağı iyi yorumlanmalıdır.

Örneğin `for` döngüsünün hiçbir parametresi verilmezse, döngü sonsuz çevrime girer. Yani:

```
for(;;)
    printf("Sonsuz döngü içindeyim...\n");
```

gibi.

7.6 break Deyimi

Bir C programında, bir işlem gerçekleştirilirken, işlemin sona erdirilmesi bu deyim ile yapılır. Örneğin, döngü deyimleri içindekiler yürütülürken, çevrimin, koşuldan bağımsız kesin olarak sonlanması gerektiğinde bu deyim kullanılır. Mesela:

```
...
do{
    scanf("%d", &x);

    if(x==0) break;

    printf("%f", 1.0/x);
}while(1);
...
```

Yukarıdaki program parçasında, `do ... while` döngüsündeki koşul daima olumludur. Bu durumda döngü sonsuzdur. Fakat döngü içinde `if` deyimindeki koşul gerçekleşirse, döngü koşuluna bakılmaksızın terkedilir. Bu işlemi sağlayan `break` deyimidir.

Program 7.6 klavyeden girilen sayı pozitif olduğu sürece sayının faktoriyelini hesaplar. Sayı negatif olduğunda döngü `break` ile sonlandırılır. İnceleyiniz.

Program 7.6: *break* deyiminin kullanımı

```
01: /* 07prg06.c: n>=0 olduğu sürece n! değerini hesaplar */
02:
03: #include <stdio.h>
04:
05: int main()
06: {
07:     long int i,n,faktor;
08:
09:     while(1) /* sonsuz döngü */
10:     {
11:         printf("Faktoriyeli hesaplanacak sayı girin : ");
12:         scanf("%ld",&n);
13:
14:         if(n<0) break; /* döngüyü sonlandır */
15:
16:         for(faktor=1, i=1; i<=n; i++)
17:             faktor *= i;
18:
19:         printf("%ld! = %ld\n",n,faktor);
20:     }
21:
22:     return 0;
23: }
```

ÇIKTI

```
Faktoriyeli hesaplanacak sayı girin : 2
2! = 2
Faktoriyeli hesaplanacak sayı girin : 3
3! = 6
Faktoriyeli hesaplanacak sayı girin : 5
5! = 120
Faktoriyeli hesaplanacak sayı girin : 9
9! = 362880
Faktoriyeli hesaplanacak sayı girin : 0
0! = 1
Faktoriyeli hesaplanacak sayı girin : -4
```

7.7 continue Deyimi

Bir döngü içerisinde `continue` deyimi ile karşılaşılsa, ondan sonra gelen deyimler atlanır ve döngü bir sonraki çevrime girer. Örneğin:

```
...
for(x=-50; i<=50; x++)
{
    if(x<0) continue; /* x<0 ise alttaki satırı atla */
    printf("%d\t%f",x,sqrt(x));
}
...
```

Program parçasının çıktısı:

0	0.000000
1	1.000000
2	1.414213
3	1.732050
.	.
.	.
.	.
50	7.071067

Program 7.7, x, y 'den farklı olmak üzere $|x|+|y| \leq 3$ eşitsizliğini sağlayan tamsayı çiftlerini bulup ekrana yazar. Bu eşitsizliği sağlayan toplam 22 çift vardır. Programda, her bir çift parantez içinde yazdırılmıştır. İnceleyiniz.

Program 7.7: *continue* deyiminin kullanımı

```
01: /* x, y'den farklı olmak üzere |x|+|y|<=3 eşitsizliğini
02: sağlayan tamsayı çiftlerini ekrana yazar */
03:
04:
05: #include <stdio.h>
06:
07: int main()
08: {
09:     int x,y,k=1;
10:
11:     for (x=-3;x<=3;x++)
12:     for (y=-3;y<=3;y++)
13:     {
14:         /* x=y ise yeni çevrime gir, alt satırları atla */
15:         if(x==y) continue;
16:
17:         if( abs(x)+abs(y)<=3 )
18:             printf("%2d. (%2d,%2d) \n",k++,x,y);
19:     }
20:     return 0;
}
```

ÇIKTI

```
1. (-3, 0)
2. (-2, -1)
3. (-2, 0)
4. (-2, 1)
5. (-1, -2)
6. (-1, 0)
7. (-1, 1)
8. (-1, 2)
9. ( 0, -3)
10. ( 0, -2)
11. ( 0, -1)
12. ( 0, 1)
13. ( 0, 2)
14. ( 0, 3)
15. ( 1, -2)
16. ( 1, -1)
17. ( 1, 0)
18. ( 1, 2)
19. ( 2, -1)
20. ( 2, 0)
21. ( 2, 1)
22. ( 3, 0)
```

Ders 8: *Fonksiyonlar I (Alt Programlar)*

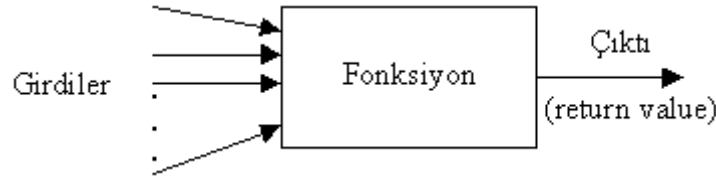
Giriş

C Programlama Dili fonksiyon olarak adlandırılan alt programların birleştirilmesi kavramına dayanır. Bir C programı bir yada daha çok fonksiyonun bir araya gelmesi ile oluşur. Bu özellik bütün Yapısal Diller'in (C, Fortran, Pascal, ...) temelini oluşturur. Yapısal Diller'e hakim olmak için fonksiyon oluşturmayı ve kullanmayı iyi öğrenmek gerekir.

Bu bölümde, C Programlama Dili'ndeki *fonksiyon kavramı*, sonraki bölümde *esnek argümanlı fonksiyonlar* ve `main()` fonksiyonu irdelenecektir.

8.1 Fonksiyon Kavramı

Fonksiyon, belirli sayıda verileri kullanarak bunları işleyen ve bir sonuç üreten komut grubudur. Her fonksiyonun bir adı ve fonksiyona gelen değerleri gösteren argümanları (bağımsız değişkenleri) vardır. Genel olarak bir fonksiyon Şekil 8.1'deki gibi bir kutu ile temsil edilir:



Şekil 8.1: Bir fonksiyonun kutu gösterimi

Fonksiyonların girdilerine parametreler yada argümanlar denir. Bir fonksiyon bu parametreleri alıp bir işleme tabi tutar ve bir değer hesaplar. Bu değer, *çıktı* veya *geri dönüş değeri* (return value) olarak adlandırılır. Unutmayın ki, 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(argüman listesi)
argümanların tip bildirimleri
{
    Yerel değişkenlerin bildirimi
    ...
    fonksiyon içindeki deyimler veya diğer fonksiyonlar
    ...
    return geri dönüş değeri;
}
```

Örneğin iki sayının toplamının hesaplayacak bir fonksiyon şöyle tanımlanabilir:

```
/* klasik biçim */
int topla(x,y)
int x,y
{
    int sonuc;
    sonuc = x + y;
    return sonuc;
}
```

veya

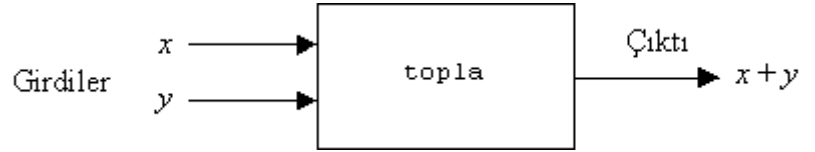
```
/* modern biçim */
int topla(int x,int y)
{
    int sonuc;
    sonuc = x + y;
    return sonuc;
}
```

veya

```
/* modern biçim */
int topla(int x,int y)
{
    return (x+y);
}
```

Bu örnekte, fonksiyonun *kimlik kartı*! ve kutu gösterimi şöyledir:

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



Her üç program parçasında da `return` (geri dönüş) deyimini kullanılmaktadır. Bu deyim 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. `topla()` fonksiyonunun kullanımı Program 8.1'in üzerinde açıklanmıştır.

8.2 Fonksiyon Bildirimi

Bir fonksiyonun bildirimi iki türlü yapılır:

1. *Ana programdan önce:*

```
2. ...
3. int topla(int x,int y)          /* fonksiyon */
4. {
5.     ...
6. }
7. ...
8. main()
9. {
10.     ...
11. }
```

12. *Ana programdan sonra:* Bu durumda fonksiyon örneği (function prototype) ana programdan önce bildirilmelidir.

```
13. ...
14. int topla(int x, int y);      /* fonksiyon örneği */
15. ...
16. main()
17. {
18.     ...
19. }
20. ...
21. int topla(int x, int y)      /* fonksiyon */
22. {
23.     ...
24. }
```

Bir C programı içinde, yazmış olduğunuz fonksiyonlar genellikle bu iki tipte kullanılır. İkinci kullanımda fonksiyon prototipi mutlaka bildirilmelidir. Aksi halde bir hata mesajı ile karşılaşılır. Fonksiyon prototipinde arguman isimlerinin yazılması zorunlu değildir. Sadece arguman 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.

Buraya kadar anlatılanlar Program 8.1 üzerinde özetlenmiştir.

Program 8.1: *topla* fonksiyonunun ana programda kullanılması

```
01: /* 08prg01.c: iki sayıyı toplar ve sonucu ekranda gösterir
02: */
03:
04: #include <stdio.h>
05:
06: int topla(int, int);  /*** fonksiyon prototipi ***/
07:
08: int main()
09: {
10:     int toplam,a,b;
11:
12:     printf("İki sayı girin : ");
13:     scanf("%d %d",&a,&b);
14:
15:     /* fonksiyon çağırılıp, a ve b değerleri parametre
16: olarak aktarılıyor.
17:     topla(a,b) = a + b değeri toplam değişkenine
18: atanması */
19:     toplam = topla(a,b);
20:
21:     printf("%d ve %d nin toplamı %d dir.\n", a,b,toplam);
22:
23:     return 0;
24: }
25:
26: /*** fonksiyon tanımlanması ***/
27:
28: /* Bu fonksiyon iki tamsayıyı toplar */
29: int topla( int x, int y )
30: {
31:     int sonuc;
32:     sonuc = x + y;
33:     return sonuc;
34: }
```

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. 16. satırda *toplam* adlı tamsayı değişkenine *topla* fonksiyonunun dönüş değeri (*a* + *b* değeri) atanmıştır.

Belki karmaşık gelmiş olabilir. Fakat Program 8.1 daha kısa şöyle yazılabilirdi:

Program 8.1b: *topla fonksiyonunun ana programda kullanılması*

```
01: /* 08prg01b.c: iki sayıyı toplar ve sonucu ekranda
02: gösterir */
03:
04: #include <stdio.h>
05:
06: int topla( int x, int y ){
07:     return (x+y);
08: }
09:
10: int main(void)
11: {
12:     int toplam,a,b;
13:
14:     printf("İki sayı girin : ");
15:     scanf("%d %d",&a,&b);
16:
17:     toplam = topla(a,b);
18:
19:     printf("%d ve %d nin toplamı %d dir.\n", a,b,toplam);
20:
21:     return 0;
}
```

8.3 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 sonlananır 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.

Program 8.2'de bildirilen `artik_yil` fonksiyonu, kendisine parametre olarak gelen bir tamsayıyı yıl bilgisi olarak kabul eder. Eğer yıl artık yıl ise 1 aksi halde 0 gönderir. Programda iki tane `return` deyimi kullanıldığına dikkat ediniz. Artık yıl tanımı Bölüm 6'da verilmişti.

Program 8.2: *iki return deyimi kullanan bir fonksiyon*

```
01: /* 08prg02.c: Bir fonksiyonda iki return deyimi */  
02:  
03: #include <stdio.h>  
04:  
05: int artik_yil(int); /* fonksiyon prototipi */  
06:  
07: void main()  
08: {  
09:     int yil;  
10:  
11:     printf("Bir yıl girin: ");  
12:     scanf("%d",&yil);  
13:  
14:     if( artik_yil(yil) )  
15:         printf("%d artık yıl\n",yil);  
16:     else  
17:         printf("%d artık yıl değil\n",yil);  
18: }  
19:  
20: /* yıl artıl yıl ise 1 aksi halde 0 gönderir */  
21: int artik_yil(int yil)  
22: {  
23:     if( yil % 4 == 0 &&  
24:         yil % 100 != 0 ||  
25:         yil % 400 == 0 ) return 1;  
26:     else return 0;  
27: }
```

ÇIKTI

```
Bir yıl girin: 1996  
1996 artık yıl
```


8.4 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, hükümsüz) 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.

Program 8.3'de `void` fonksiyona örnek olarak `bankamatik` fonksiyonu ve kullanımı gösterilmiştir. Bu fonksiyon kendisine parametre olarak gelen YTL cinsinden para miktarını 20, 10 ve 5 YTL'lik birimler halinde hesaplar. Girilen miktar 5 YTL'nin bir katı değilse, ekrana uygun bir mesaj gönderir. `bankamatik` fonksiyonu bir dizi hesap yapmasına rağmen geriye hiç bir değer göndermez.

Program 8.3: `void` tipinde bir fonksiyon kullanımı

```
01: #include <stdio.h>
02:
03: void bankamatik(int para)
04: {
05:     int a,yirmilik,onluk,beslik;
06:
07:     a = para;
08:
09:     if(a%5==0)
10:     {
11:         yirmilik = a/20;
12:         a -= yirmilik*20;
13:
14:         onluk = a/10;
15:         a -= onluk*10;
16:
17:         beslik = a/5;
18:         a -= beslik*5;
19:
20:         printf("\nYirmilik = %d",yirmilik);
21:         printf("\nOnluk   = %d",onluk);
22:         printf("\nBeslik   = %d\n",beslik);
23:     }
24:     else
25:         printf("Girilen miktar 5 YTL ve katlari
26: olmali!\a\n");
27:
28:     /* return deyimi yok !*/
29: }
30:
31: int main()
32: {
33:     int miktar;
34:
35:     printf("Cekilecek para miktarı (YTL) = ");
36:     scanf("%d",&miktar);
37:
38:     bankamatik(miktar); /* fonksiyon bir değişkene
39: atanmamış ! */
40:
41:     return 0;
42: }
```

ÇIKTI

```
Cekilecek para miktarı = 135
```

```
Yirmilik = 6  
Onluk    = 1  
Beslik   = 1
```

ÇIKTI

```
Cekilecek para miktarı = 456  
Girilen miktar 5 YTL ve katları olmalı!
```

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

<pre>void bankamatik(int para) { ... }</pre>	<pre>bankamatik(int para) { ... }</pre>
--	---

şeklindeki kullanımlar geçerli ve aynı anlamdadır.

Başka bir `void` fonksiyon örneği Program 8.4'de verilmiştir. Programdaki `kutu_ciz` fonksiyonu, iki `for` döngüsü kullanarak 'x' karakterlerinden oluşan basit bir kutu çizimi yapar. Programda de sadece 18. satır defalarca işleme konur. Program çalıştırıldığında $8 \times 35 = 280$ adet 'x' karakteri ekrana bastırılır. İnceleyiniz.

Program 8.4: basit kutu çizen fonksiyon

```
01: /* 08prg04.c: Basit bir kutu çizen fonksiyon */  
02:  
03: #include <stdio.h>  
04:  
05: void kutu_ciz( int satir, int sutun )  
06: {  
07:     int sut;  
08:     for ( ; satir > 0; satir-- )  
09:     {  
10:         for (sut = sutun; sut > 0; sut--)  
11:             printf("X");  
12:  
13:         printf("\n");  
14:     }  
15: }  
16:  
17: int main() {  
18:  
19:     kutu_ciz(8,35);  
20:  
21:     return 0;  
22: }  
23:
```

ÇIKTI

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

8.5 Fonksiyon Parametreleri

Fonksiyon parametreleri (argümanları) klasik ve modern olmak üzere iki türlü tanımlanabilir. Örneğin aşağıdaki fonksiyon kendisine parametre olarak gelen tamsayının faktoriyelini gönderir. Bu fonksiyonun parametresi (n):

```
int faktoriyel(n)      /* kalsik biçim */
int n
{
    int i=1, f=1;

    while(i<=n) f *= i++;

    return f;
}
```

şeklinde yada:

```
int faktoriyel(int n)  /* modern biçim */
{
    int i=1, f=1;

    while(i<=n) f *= i++;

    return f;
}
```

şeklinde yazılabilir.

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. Örneğin standard C'de `stdlib.h` kütüphanesinde tanımlı `rand` fonksiyonu şöyle tanımlanmıştır:

```
int rand(void);
```

Son olarak, ne parametresi ne de geri dönüş değerine olan bir fonksiyon şöyle tanımlanabilir:

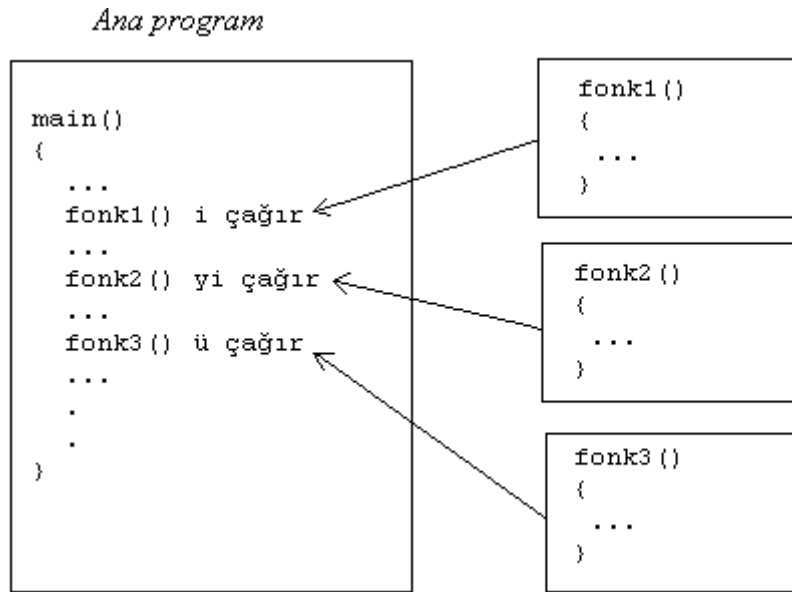
```
void mesaj_yaz()  
{  
    printf("Hata olustu !..\n");  
}
```

yada

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

8.6 Yapısal Programlama

Program içinde birden çok fonksiyon tanımlayıp kullanmak mümkündür. Yani C Programlama Dili fonksiyonların inşası dayalı bir dildir. Bu özelliklik bütün Yapısal Programlama Dilleri'nin (Structred Programming) temelini oluşturur. Birden çok fonksiyonun `main` tarafından nasıl çağrıldığını temsil eden blok diyagram Şekil 8.2'de gösterilmiştir.



Şekil 8.2: Ana programdan alt programların (fonksiyonların) çağırılması.
Fonksiyonu çağırmak için, fonksiyonun adını yazmak yeterlidir.

Fonksiyonların sadece ana program tarafından çağırılması zorunlu değildir. Bir fonksiyon başka bir fonksiyon tarafından da çağırılabilir. Bu tür kullanıma dair bir örnek Program 8.5'de verilmiştir. `yilin_gunu` fonksiyonu, verilen bir tarihin yılın kaçınıcı günü olduğunu hesaplar ve çağrıldığı yere gönderir. İnceleyiniz.

Program 8.5: bir fonksiyonun başka bir fonksiyon tarafından çağırılması

```
01: /* 08prg05.c: Verilen bir tarihin yılın kaçınıcı günü
02: olduğunu hesaplar. */
03:
04: #include <stdio.h>
05:
06: int yilin_gunu(int, int, int);
07: int artik_yil(int);
08:
09: int main(void)
10: {
11:     int gun = 1;      /* tarih: 01 Ağustos 2003 */
12:     int ay  = 8;
13:     int yil = 2003;
14:
15:     printf("%02d %02d  %d yilinin\n",gun,ay,yil );
16:     printf("%d. gunudur\n",yilin_gunu(gun,ay,yil) );
17:
18:     return 0;
19: }
20:
21: /* yıl artıl yıl ise 1 aksi halde 0 gönderir */
22: int artik_yil(int yil)
23: {
24:     if( yil%4==0 && yil%100!=0 || yil%400==0 ) return 1;
25:     else return 0;
26: }
27:
28: /* yılın kaçınıcı günü olduğunu hesaplar ve o günü
29: gönderirir */
30: int yilin_gunu(int gun, int ay, int yil)
31: {
32:     int ygun = gun;
33:
34:     switch(ay-1)
35:     {
36:         case 12: ygun += 31;
37:         case 11: ygun += 30;
38:         case 10: ygun += 31;
39:         case 9:  ygun += 30;
40:         case 8:  ygun += 31;
41:         case 7:  ygun += 31;
42:         case 6:  ygun += 30;
43:         case 5:  ygun += 31;
44:         case 4:  ygun += 30;
45:         case 3:  ygun += 31;
46:         case 2:  ygun += 28 + artik_yil(yil); /* 28+1 veya
47: 28+0 */
48:         case 1:  ygun += 31;
49:     }
50:
51:     return ygun;
52: }
```

ÇIKTI

01 08 2003 yılının
213. gunudur

8.7 Makro Fonksiyon Tanımlaması

Başlık dosyalarında, bol miktarda makro fonksiyon uygulamalarına rastlanır. Makro tanımlaması `#define` önışlemci komutu kullanılarak yapılır. Örneğin aşağıdaki makro fonksiyonlar geçerlidir.

```
#define kare(x) (x)*(x)
#define delta(a,b,c) ((b)*(b)-4*(a)*(c))
#define yaz() puts("Devam etmek için bir tuşa basın...")
```

Bu şekilde tanımlanan fonksiyonların kullanımı diğerleri gibidir. Yalnızca programın başında tanımlanır. Ancak, bu tanımlamalarla fonksiyon bellekte bir yer işgal etmez.

Basit bir makro fonksiyon uygulaması Program 8.6'da gösterilmiştir. `buyuk(a,b)` makrosu `a>b` ise `a` değerini aksi halde `b` değerini gönderir.

Program 8.6: Makro fonksiyon uygulaması

```
01: /* 08prg06.c: makro fonksiyon uygulaması */
02:
03: #include <stdio.h>
04:
05: #define buyuk(a,b) ( (a>b) ? a:b)
06:
07: int main()
08: {
09:     int x,y,eb;
10:
11:     printf("iki sayı girin: ");
12:     scanf("%d,%d",&x,&y);
13:
14:     eb = buyuk(x,y);
15:
16:     printf("buyuk olan  %d\n",eb);
17:
18:     return 0;
19: }
```

ÇIKTI

iki sayı girin: 8,6
buyuk olan 8

Ders 9: Fonksiyonlar II (Alt programlar)

Giriş

Bu kısımda, esnek argümanlı fonksiyonlar, `main()` fonksiyonu ve komut satırından `main()` fonksiyonuna parametre aktarımı incelenektir.

9.1 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 geçirilmelidir. Kullanılmayan argümanlar geçirilmeden de fonksiyonu çağırmak mümkündür. C++, Fortran 90 gibi dillerde olduğu gibi, C Programlama Dili, kullanıcılarına argümanları *esnek* olarak geçirme imkanı verir. Bunun anlamı, belli kurallar sağlandığında, `p()` fonksiyonu aşağıdaki gibi çağırılabilmesidir:

```
/* 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 9.1'de listelenmiştir.

Tablo 9.1: *stdarg.h'te tanımlı tip ve makro fonksiyonlar*

Tip / Fonksiyon	Açıklama
<code>va_list</code>	ardışık esnek argümler için tip belirleyici
<code>va_start(ap, n)</code>	<code>va_list</code> tipinde bildirilmiş <code>ap</code> göstericisi için bellekten <code>n</code> elemanlı yer ayırır.
<code>va_arg(ap, tip)</code>	Veri tipi <code>tip</code> ile belirlenmiş küme elemanlarına erişir.
<code>va_end(ap)</code>	<code>va_list</code> tipinde bildirilmiş <code>ap</code> göstericisi için bellekten bölgeyi boşaltır.

Bu kurallar ışığında, `p()` fonksiyonunun genel kullanımı Program 9.1'de gösterilmiştir. `p()`, kendisine parametre olarak gelen x , n ve a_i katsayılarına göre

$$P(x,n) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$$

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

Program 9.1: Sonu -1 ile biten kümeyi ekrana yazar

```
01: #include <stdarg.h>
02: #include <stdio.h>
03: #include <math.h>
04:
05: /* fonksiyon örneği */
06: double p(double, int, ...);
07:
08:
09: int main(void) {
10:
11:     double x = 1.7;
12:
13:     printf("x = %lf için:\n", x);
14:
15:     printf("p(x, 1, 1.0) = %lf\n",
16:           p(x, 1, 1.0));
17:
18:     printf("p(x, 2, 1.0, -2.0) = %lf\n",
19:           p(x, 2, 1.0, -2.0));
20:
21:     printf("p(x, 3, 1.0, -2.0, 0.2) = %lf\n",
22:           p(x, 3, 1.0, -2.0, 0.2));
23:
24:     printf("p(x, 4, 1.0, -2.0, 0.2, 1.1) = %lf\n",
25:           p(x, 4, 1.0, -2.0, 0.2, 1.1));
26:
27:     printf("p(x, 5, 1.0, -2.0, 0.2, 1.1, -0.6) = %lf\n",
28:           p(x, 5, 1.0, -2.0, 0.2, 1.1, -0.6));
29:
30:     return 0;
31: }
32:
33: /* Verilen x, n ve ai katsayıları için,
34:     $P(x, n) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$  polinomu
35:    hesaplar.
36:    a0, a1, ..., an katsayıları esnek arguman olarak
37:    bildirilmiştir. */
38: double p(double x, int n, ...)
39: {
40:     double a, t = 0.0;
41:     int i;
42:
43:     /* arguman göstericisi; ag va_list tipinde */
44:     va_list ag;
45:
46:     /* ag için bellekten n adet hücre ayır */
47:     va_start(ag, n);
48:
49:     for(i=0; i<n; i++)
50:     {
51:         /* herbir argumanı sırasıyla al */
52:         a = va_arg(ag, double);
53:
54:         /* polinomun değerini hesapla */
55:         t += a*pow(x, i);
56:     }
57:
58:     va_end(ag);
59:
60:     return t;
61: }
```

ÇIKTI

```
x = 1.700000 için:
p(x, 1, 1.0) = 1.000000
p(x, 2, 1.0, -2.0) = -2.400000
p(x, 3, 1.0, -2.0, 0.2) = -1.822000
p(x, 4, 1.0, -2.0, 0.2, 1.1) = 3.582300
p(x, 5, 1.0, -2.0, 0.2, 1.1, -0.6) = -1.428960
```

Program 9.2'de, argümanları esnek olarak bildirilmiş `topla(int n, ...)` fonksiyonu, n tane tamsayının sayının toplamını hesaplar.

Program 9.2: n tane sayının toplamını hesaplar

```
01: /* 09prg02.c
02:    n tane sayının toplamının hesaplanması */
03:
04: #include <stdarg.h>
05: #include <stdio.h>
06:
07:
08: int topla(int, ...);
09:
10: int main(void)
11: {
12:     printf("topla(2, 11,22)           = %d\n", topla(2,
13: 11,22));
14:     printf("topla(3, 11,22,33)        = %d\n", topla(3,
15: 11,22,33));
16:     printf("topla(4, 11,22,33,44)     = %d\n", topla(4,
17: 11,22,33,44));
18:     printf("topla(5, 11,22,33,44,55)  = %d\n", topla(5,
19: 11,22,33,44,55));
20:     printf("topla(6, 11,22,33,44,55,66) = %d\n", topla(6,
21: 11,22,33,44,66,66));
22:
23:     return 0;
24: }
25:
26: /* Esnek argumanla tanımlanmış n tane tamsayının sayının
27:    toplamını gönderir */
28: int topla(int n, ...)
29: {
30:     va_list ap;
31:     int i, top = 0;
32:
33:     va_start(ap, n);
34:
35:     for (i=1; i<=n; i++)
36:         top += va_arg(ap, int);
37:
38:     va_end(ap);
39:     return top;
40: }
```

ÇIKTI

```
topla(2, 11,22)           = 33
topla(3, 11,22,33)        = 66
topla(3, 11,22,33,44)     = 110
topla(5, 11,22,33,44,55)  = 165
topla(6, 11,22,33,44,55,66) = 242
```

Argüman sayısı bildirilmeden de bir küme üzerinde işlem yapılabilir. Ancak bu durumda kümenin boyutu başka bir yöntemle hesaplanmalıdır. Program 9.3'de, argümanları esnek olarak bildirilmiş `argyaz(int arg, ...)` fonksiyonu, son elemanı -1 olan bir kümenin elemanlarını ekrana yazar. Kümenin sonu (yani boyutu) -1 ile belirlenmiş olur.

Program 9.3: Sonu -1 ile biten kümeyi ekrana yazar

```
01: /* 09prg03.c: Esnek argumanların yazdırılması */
02:
03: #include <stdio.h>
04: #include <stdarg.h>
05:
06: /* herbiri int tipinde ve sonu -1 ile biten kümeyi ekrana
07: yazar */
08: void argyaz(int arg, ...)
09: {
10:     va_list ap;
11:     int i;
12:
13:     va_start(ap, arg);
14:
15:     for (i = arg; i != -1; i = va_arg(ap, int))
16:         printf("%d ", i);
17:
18:     va_end(ap);
19:     putchar('\n');
20: }
21:
22: int main(void)
23: {
24:     argyaz(5, 2, 14, 84, 97, 15, 24, 48, -1);
25:     argyaz(84, 51, -1);
26:     argyaz(-1);
27:     argyaz(1, -1);
28:
29:     return 0;
}
```

ÇIKTI

```
5 2 14 84 97 15 24 48
84 51
1
```

9.2 main Fonksiyonu

Ana program anlamına gelen `main` de bir fonksiyondur. C programlarının başlangıcı ve sonu bu fonksiyonla belirlenir. Buna göre, bir C (veya C++) programı sadece bir tane `main` içerebilir.

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

```
return 0;
```

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

Bazı programcılar `main` fonksiyonunun başına şey yazmaz.

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

Bu durumda geri dönüş değeri tamsayı (`int`) kabul edilir. Bu şekilde kullanımda, yeni tip derleyiciler uyarı (warning) mesajı verebilirler. Bu yüzden, aşağıdaki kullanımı tavsiye ediyoruz.

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

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

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

yada

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

9.3 main() Fonksiyonuna Parametre Aktarımı

NOT

Bu ve sonraki kısımda (9.3) anlatılanlar Bölüm 10, 11 ve 16 okunduktan sonra daha iyi anlaşılacaktır. Ancak, konu akışını bozmamak için, bu konunun buraya konması uygun bulunmuştur.

Ana programa parametre aktarımı, derlenmiş (çalıştırılabilir) bir program komut satırından (işletim sistemi ortamından) çalıştırılacağı zaman yapılır. Parametre aktarımı, programın adı yazılıp bir boşluk bırakıldıktan hemen sonra yapılır. Parametreler, komut satırından sayısal olarak girilse bile program içinde karakter topluluğu (string) olarak gelir. Bu durumda, bu ifadeleri sayısal değerlere çeviren (`atoi()`, `atol()`, `atof()` gibi) fonksiyonlar kullanılır[1].

Genel kullanım biçimi:

```
...
int main(arguman_sayısı, arguman_vektörü)
int arguman_sayısı;
char *arguman_vektörü[];
{
    .
    .
    .
    if(arguman_sayısı < ...){
        printf("Eksik parametre !\n");
        exit(1);
    }
    if(arguman_sayısı > ...){
        printf("Cok fazla parametre !\n");
        exit(1);
    }
    .
    ... arguman_vektörü[0] ... /* 1. eleman program adı */
    ... arguman_vektörü[1] ... /* 2. eleman 1. parametre */
    ... arguman_vektörü[2] ... /* 3. eleman 2. parametre */
    .
}
```

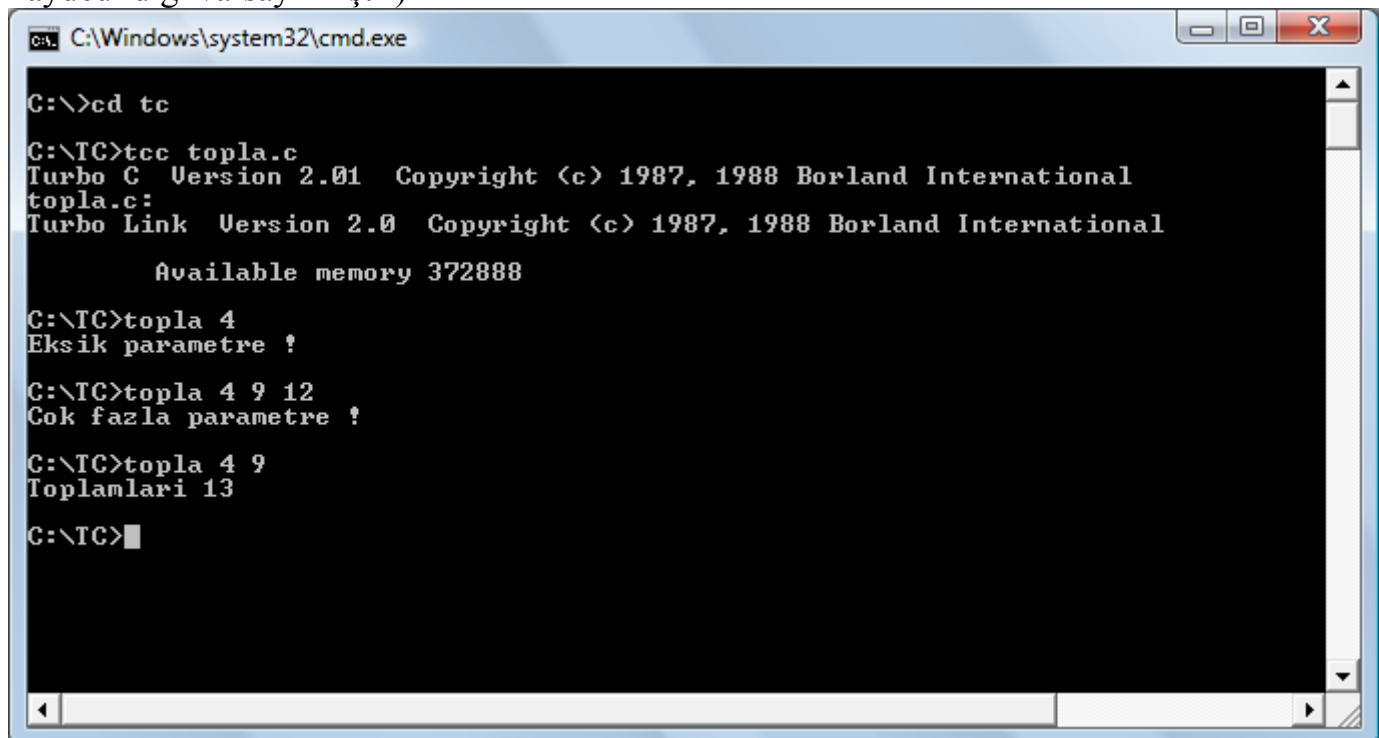
Program 9.1, komut satırından girilen iki sayının toplamını hesaplar.

Program 9.1: Komut satırından girilen iki sayının toplamını hesaplar

```
01: /* topla.c
02:    Komut satırından girilen iki sayının toplamını
03:    hesaplar.
04:    Kullanımı:  topla sayı1 sayı2 */
05:
06: #include <stdio.h>
07: #include <stdlib.h>
08:
09: int main(int argsay, char *argvek[]){
10:
11:     int toplam;
12:
13:     if(argsay < 3){
14:         printf("Eksik parametre !\n");
15:         exit(1);
16:     }
17:
18:     if(argsay > 3){
19:         printf("Cok fazla parametre !\n");
20:         exit(1);
21:     }
22:
23:     toplam = atoi(argvek[1]) + atoi(argvek[2]);
24:
25:     printf("Toplamlari %d\n",toplam);
26:
27:     return 0;
28: }
```

Program 9.1, topla.c derlendikten sonra üretilen Windows ortamında üretilen topla.exe ve Linux ortamında üretilen topla dosyasının çalıştırılması şöyledir:

- **Turbo C 2.0 Derlecisi kullanılarak** (topla.c programı C:\TC adlı dizinin altına kaydedildiği varsayılmıştır)



```
C:\Windows\system32\cmd.exe

C:\>cd tc

C:\TC>tcc topla.c
Turbo C Version 2.01 Copyright (c) 1987, 1988 Borland International
topla.c:
Turbo Link Version 2.0 Copyright (c) 1987, 1988 Borland International

    Available memory 372888

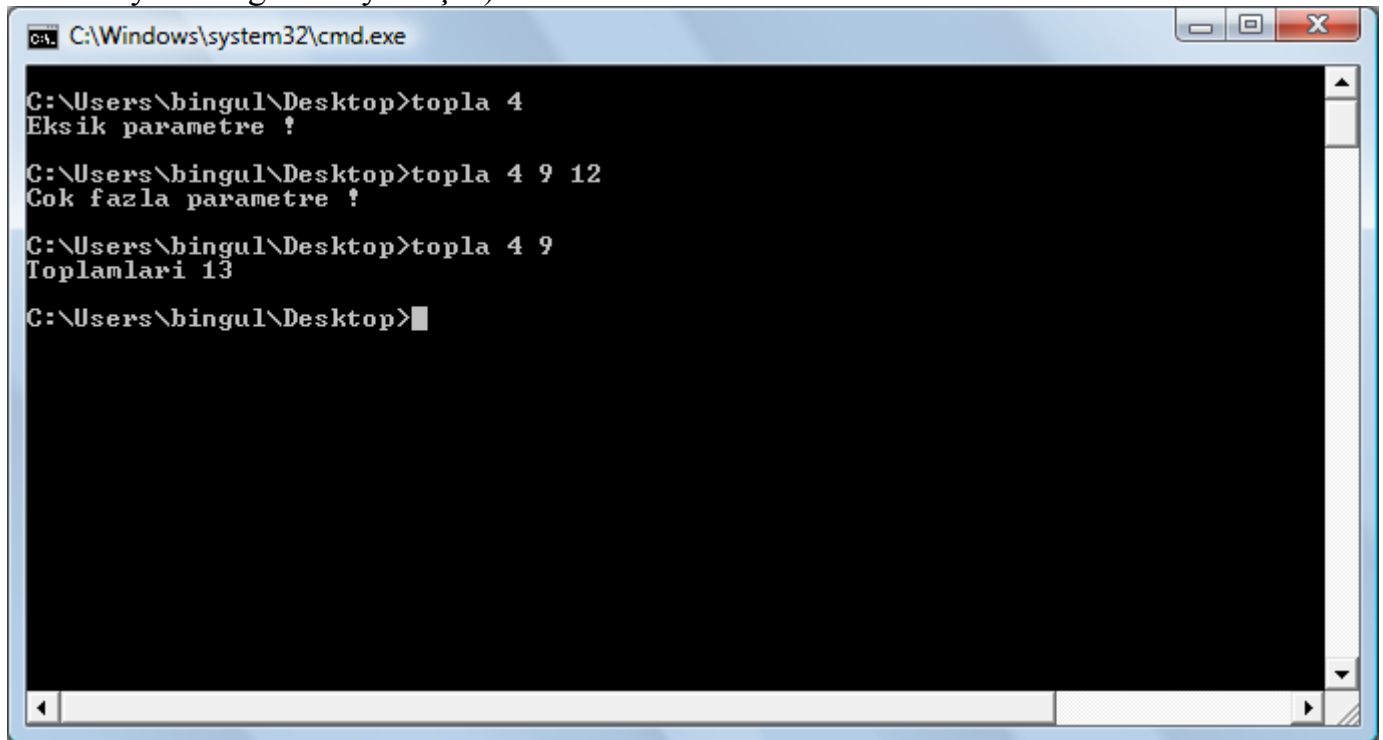
C:\TC>topla 4
Eksik parametre !

C:\TC>topla 4 9 12
Cok fazla parametre !

C:\TC>topla 4 9
Toplamlari 13

C:\TC>
```

- **Dev-C++ Derlecisi kullanılarak** (topla.c programı C:\Users\bingul\Desktop adlı dizinin altına kaydedildiği varsayılmıştır)



```
C:\Windows\system32\cmd.exe

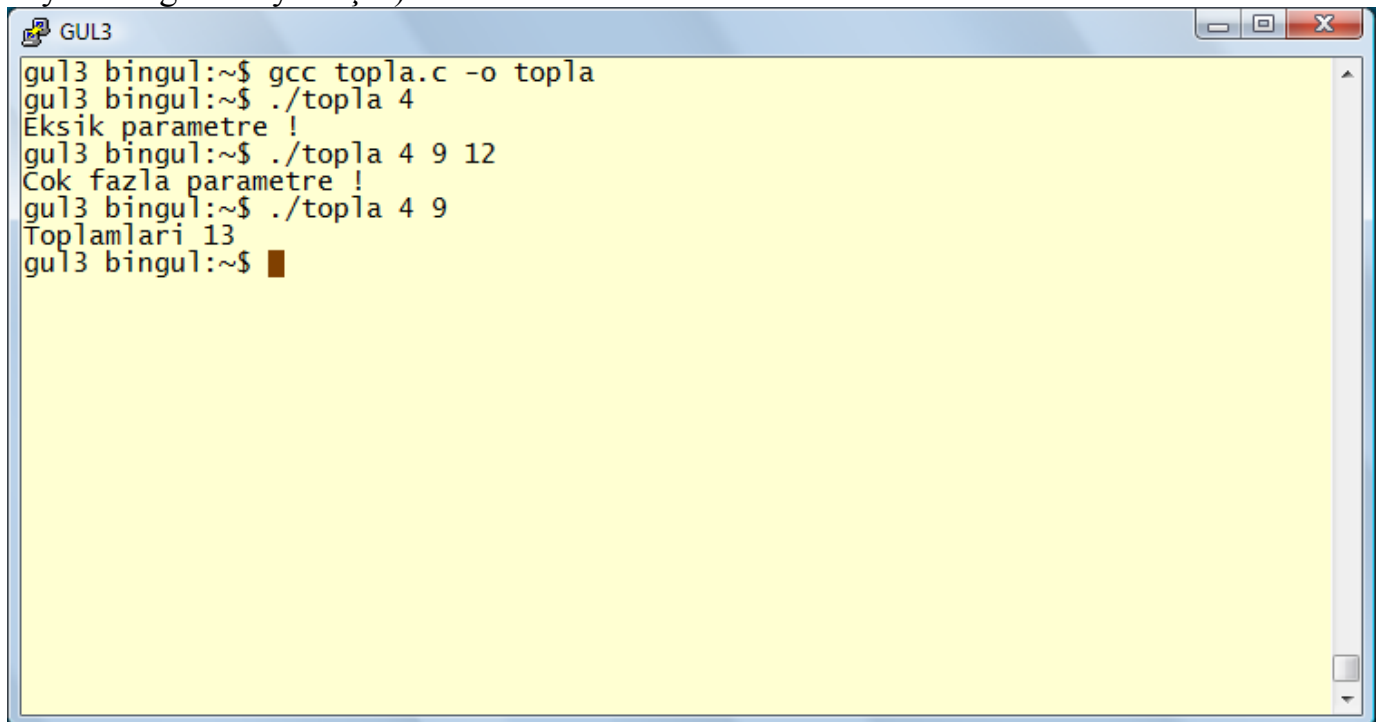
C:\Users\bingul\Desktop>topla 4
Eksik parametre !

C:\Users\bingul\Desktop>topla 4 9 12
Cok fazla parametre !

C:\Users\bingul\Desktop>topla 4 9
Toplamlari 13

C:\Users\bingul\Desktop>■
```

- **Linux gcc derleyicisi kullanılarak** (topla.c programı /home/bingul/ adlı dizinin altına kaydedildiği varsayılmıştır)



```
GUL3

gul3 bingul:~$ gcc topla.c -o topla
gul3 bingul:~$ ./topla 4
Eksik parametre !
gul3 bingul:~$ ./topla 4 9 12
Cok fazla parametre !
gul3 bingul:~$ ./topla 4 9
Toplamlari 13
gul3 bingul:~$ ■
```

Komut satırında yazılan dosya adı dahil toplam parametre sayısı 3 tür. Bunlar:

topla	4	9
v	v	v
argvek[0]	argvek[1]	argvek[2]

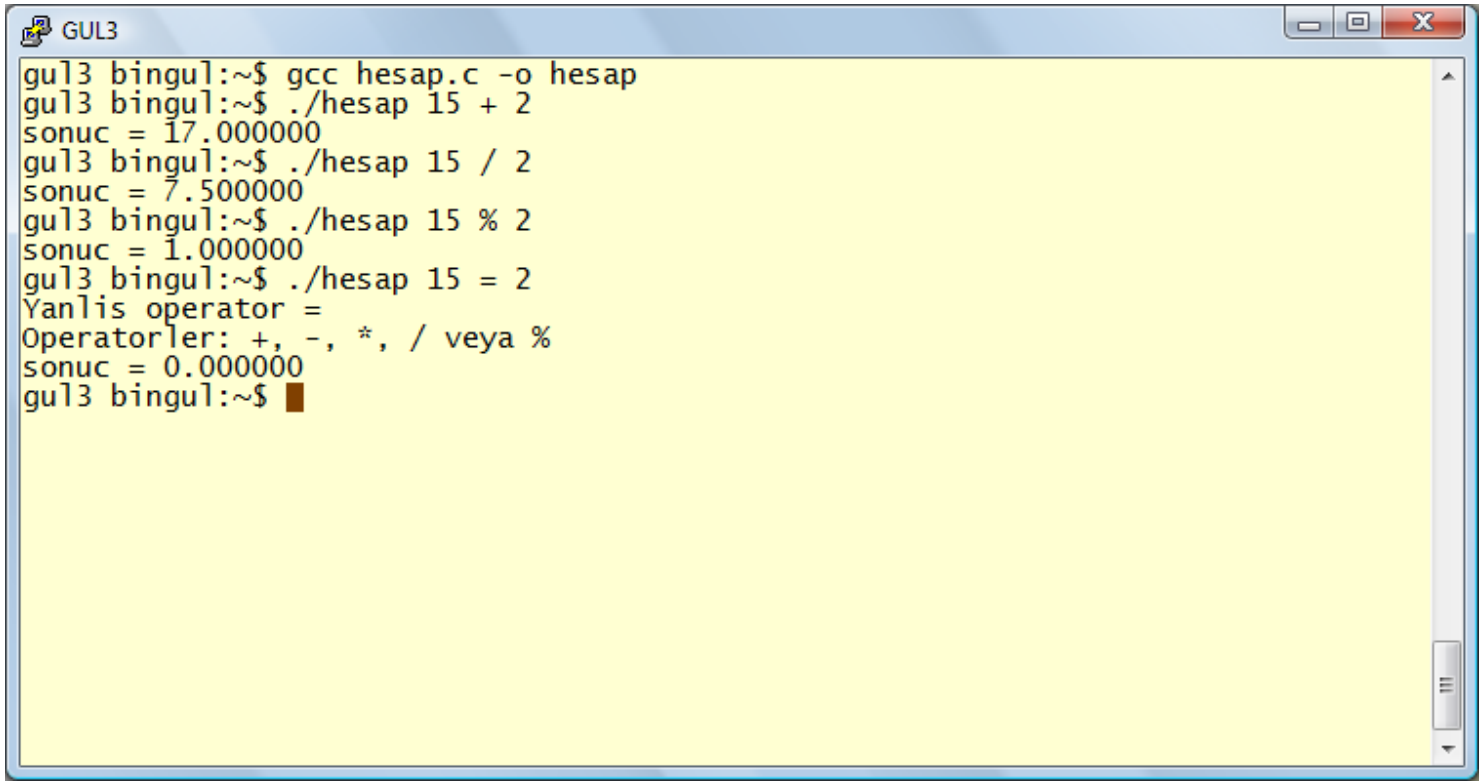
şeklindedir.

Program 9.1, komut satırından girilen iki sayının toplamını hesaplar. Bu programın daha gelişmiş hali Program 9.2'de verilmiştir. Program 9.2 çalıştırıldığında, komut satırından girilen iki sayı ve bir operatör bekler. Girilen operatöre göre beş aritmetik işlemden birini yapıp sonucu ekranda gösterir. İnceleyiniz.

Program 9.2: *Komut satırından girilen iki sayı ve bir operatör bilgisine göre 5 işlemden birini hesaplar*

```
01: /* hesap.c: Komut satırından girilen iki sayı üzerinde 5
02: işlem yapar.
03:
04:     Kullanımı: hesap <sayi1> <operator> <sayi2> */
05:
06: #include <stdio.h>
07: #include <stdlib.h>
08:
09: int main(int args, char **argv)
10: {
11:     int    s1, s2;
12:     float  sonuc;
13:     char   op;
14:
15:     if(args != 4){
16:         printf("Eksik veya fazla parametre !\n");
17:         printf("Kullanımı: hesap <sayi1> <operator>
18: <sayi2>\n");
19:         return 1;
20:     }
21:
22:     s1 = atoi(argv[1]);    /* 1. parametre: sayi1 */
23:     op = argv[2][0];      /* 2. parametrenin ilk
24: karakteri: operator */
25:     s2 = atoi(argv[3]);    /* 3. parametre: sayi2 */
26:
27:     switch(op)
28:     {
29:         case '+':
30:             sonuc = s1 + s2; break;
31:         case '-':
32:             sonuc = s1 - s2; break;
33:         case '*':
34:             sonuc = s1 * s2; break;
35:         case '/':
36:             sonuc = (float) s1 / s2; break;
37:         case '%':
38:             sonuc = s1 % s2; break;
39:         default:
40:             sonuc = 0.0;
41:             printf("Yanlis operator %c\n",op);
42:             printf("Operatorler: +, -, *, / veya %%\n");
43:     }
44:
45:     printf("sonuc = %f\n",sonuc);
46:
47:     return 0;
48: }
```

Program hesap.c adlı dosyada saklandığı varsayılırsa, programın Linux ortamındaki çıktısı şöyle olacaktır:



```
gul3 bingul:~$ gcc hesap.c -o hesap
gul3 bingul:~$ ./hesap 15 + 2
sonuc = 17.000000
gul3 bingul:~$ ./hesap 15 / 2
sonuc = 7.500000
gul3 bingul:~$ ./hesap 15 % 2
sonuc = 1.000000
gul3 bingul:~$ ./hesap 15 = 2
Yanlis operator =
Operatorler: +, -, *, / veya %
sonuc = 0.000000
gul3 bingul:~$
```

9.4 Komut Satırı Örnekleri

Aşağıda verilen iki program, Linux işletim sistemindeki `cp` ve `wc` komutlarının basit kaynak kodlarıdır:

- `cp` (copy) komutu, bir text dosyasının kopyasını oluşturur.
Kullanımı: `cp kaynak_dosya hedef_dosya`
- `wc` (word count) komutu, bir dosyanın kaç karakter, kelime satırdan oluştuğunu bulup ekrana yazar.
Kullanımı: `wc dosya_adı`

Ders 10: Diziler

Giriş

Dizi, aynı tipteki verilere tek bir isimle erişmek için kullanılan bir kümedir. Bu küme matematikteki küme kavramından biraz farklıdır. Bir dizi bildirildikten sonra, dizinin bütün elemanları bellekte peşpeşe saklanır. Bu yüzden dizilere tek bir isim altında çok sayıda değişken içeren bellek bölgesi de denir. Buna göre, bir diziyi dizi yapan iki temel özellik vardır:

- dizi elemanların bellekte (program çalıştığı sürece) sürekli biçimde bulunması

- dizi elemanların aynı türden değişkenler olması

10.1 Dizilerin Bildirimi

Bir dizi çok sayıda değişken barındırdığından, bunları birbirinden ayırtetmek için *indis* adı verilen bir bilgiye ihtiyaç vardır. C Programlama Dili'nde, bir dizi hangi tipte tanımlanmış olursa olsun başlangıç indisi her zaman 0'dır.

Bir dizinin bildirim işleminin genel biçimi şöyledir:

```
veriTipi dizi_adi[eleman_sayısı];
```

Örneğin, 5 elemanlı, kütle verilerini bellekte tutmak için, `kutle` dizisi şöyle tanımlanabilir:

```
float kutle[5];
```

Bu dizinin elemanlarına bir değer atama işlemi şöyle yapılabilir:

```
kutle[0] = 8.471
kutle[1] = 3.683
kutle[2] = 9.107
kutle[3] = 4.739
kutle[4] = 3.918
```

NOT

1. elemanın indisi 0,
5. elemanın indisinin 4 olduğuna dikkat edin.

Bildirim sırasında dizilerin eleman sayısı tamsayı türünden bir sabit ifadesiyle belirtilmesi zorunludur. Örneğin:

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

şeklindeki tanımlama, dizi uzunluğunun değişken (n) ile belirtilmesi nedeniyle geçersizdir. Bunun yerine, dizilerin eleman sayısı aşağıdaki gibi sembolik sabitlerle belirtmek mümkündür.

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

Bir dizinin bellekte kapladığı alanın bayt cinsinden karşılığı `sizeof` operatörü ile öğrenilebilir.

```
int a[5], b, c;
...
b = sizeof(a); /* bellekte kapladığı alan: b = 4*5 = 20 bayt */
c = sizeof(a) / sizeof(int); /* Dizinin boyutu : c = 20/4 = 5 */
```

10.2 Dizilere Başlangıç Değeri Verme

Bir diziye başlangıç değerleri aşağıdaki gibi kısa formda atanabilir:

```
float kutle[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 bittiğine dikkat ediniz.

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.

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

daha sonra bu değerlerini ekrana yazmak için:

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

Program 10.1, klavyeden girilen $N = 10$ adet sayının ortalamasını hesaplar.

$$\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i$$

Ortalama formülü ile hesaplanabilir.

Program 10.1: 10 sayının ortalamasını hesaplar

```
01: /* 10prg01.c: 10 tamsayının aritmetik ortalamasını
02: hesaplar */
03:
04: #include <stdio.h>
05:
06: #define N 10
07:
08: int main()
09: {
10:     int i;
11:     float x[N], ort, toplam = 0.0;
12:
13:     for(i=0; i<N; i++)
14:     {
15:         /* i. eleman okunuyor ... */
16:         printf("%d. sayı : ", i+1);
17:         scanf("%f", &x[i]);
18:
19:         toplam += x[i];
20:     }
21:
22:     ort = toplam/N;
23:
24:     printf("Sayıların ortalaması = %f\n", ort);
25:
26:     return 0;
}
```

ÇIKTI

```
1. sayı : 1
2. sayı : 0
3. sayı : 9
4. sayı : 7
5. sayı : 2
6. sayı : 10
7. sayı : 11
8. sayı : 4
9. sayı : 6
10. sayı : 5
Sayıların ortalaması = 5.500000
```

Bu programda, ortalaması alınacak sayılar adı x olan 10 elemanlı tamsayı tipindeki bir dizide saklanmıştır. Bu şekilde saklanan sayıların hepsi program çalıştığı sürece bellekte kalacaktır. Bu sayede, program içinde daha sonra (gerektiğinde) aynı sayılar tekrar kullanılabilir. Bu program, dizi kullanmadan da yazılabilirdi. Fakat, bazı hallerde dizi kullanmak kaçınılmaz olur.

Program 10.2, $n = 10$ tane sayının ortalamasını ve standart sapmasını hesaplar. Standart sapma,

$$\sigma = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2}$$

formülü ile hesaplanabilir. Burada,

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$

Program 10.2: 10 sayının ortalamasını ve standart sapmasını hesaplar

```
01: /* 10prg02.c
02:     10 tane sayının aritmetik ortlamasını ve standart
03:     sapmasını hesaplar. */
04:
05: #include <stdio.h>
06: #include <math.h>
07:
08: #define N 10
09:
10: int main(void)
11: {
12:     int i;
13:     float x[N], toplam = 0.0, ort, std_sap = 0.0;
14:
15:     /* ortalama hesabı */
16:     for(i=0; i<N; i++)
17:     {
18:         printf("%d. sayı : ",i+1);
19:         scanf("%f",&x[i]);
20:
21:         toplam += x[i];
22:     }
23:
24:     ort = toplam/N;
25:
26:     /* standart sapma hesabı */
27:     for(toplam = 0.0, i=0; i<N; i++)
28:         toplam += pow(x[i]-ort, 2.0);
29:
30:     std_sap = sqrt( toplam/(N-1) );
31:
32:     printf("Ortalama          = %f\n",ort);
33:     printf("Standart sapma = %f\n",std_sap);
34:
35:     return 0;
}
```

ÇIKTI

```
1. sayı : 7
2. sayı : 8
3. sayı : 9
4. sayı : 6
5. sayı : 5
6. sayı : 8
7. sayı : 8
8. sayı : 10
9. sayı : 11
10. sayı : 6
Ortalama          = 7.000000
Standart sapma = 2.054805
```

10.4 Sıralama (Sorting)

Bazı uygulamalarda bir grup sayının büyükten küçüğe, veya küçüktten büyüğe, doğru sıralanması gerekebilir. Bu tip sıralama problemleri için çeşitli algoritmalar geliştirilmiştir. Sıralama mantığını anlamadan önce bir dizinin en büyük (veya en küçük) elemanının nasıl bulunduğunu inceleyelim. Program 10.3, bir dizinin en büyük elemanını bulup ekrana yazar.

Program 10.3: *Bir dizinin en büyük elemanının bulunuşu*

```
01: /* 10prg03.c
02:     Bir dizinin en büyük elemanını bulup ekrana yazar */
03:
04: #include <stdio.h>
05:
06: int main(void)
07: {
08:     int    a[10] = {100, -250, 400, 125 ,550, 900, 689, 450,
09: 347, 700};
10:     int    k, eb;
11:
12:     /* ilk eleman en büyük kabul ediliyor */
13:     eb = a[0];
14:
15:     for(k=1; k<10; k++)
16:         if( a[k]>eb ) eb = a[k];
17:
18:     printf("En buyuk eleman = %d\n",eb);
19:
20:     return 0;
21: }
```

ÇIKTI

```
En buyuk eleman = 900
```

En büyük sayıyı bulan bu algoritma oldukça kolaydır. 12. satırda `eb = a[0];` ataması ile dizinin ilk elemanının en büyük olduğu varsayılır. Daha sonra büyüğe rastladıkça (15. satır) `eb = a[k];` ile `eb` değiştirilmektedir.

Program 10.3 bir dizinin en büyük elemanını bulur. En büyük elemanın kaçınıcı indis (eleman) olduğu sorgulanmak istendiğinde: programdaki koşul yapısını aşağıdaki gibi değiştirmek yeterlidir. `eb` değiştikçe, `i` değişkeni en büyük elemanın indisini tutar.

```
for(k=0; k<10; k++){
    if( a[k] > eb ){
        eb = a[k];
        i = k;
    }
}
```

n elemanlı bir dizinin, elemanlarını büyükten küçüğe doğru sıralamak için çok popüler iki algoritma aşağıda verilmiştir[2].

Yanyana elemanları karşılaştırarak yer değiştir biçimde sıralamadır. Algoritmanın uygulaması Program 10.5'de gösterilmiştir.

Bu algoritmanın karmaşıklığı: $(n-1)^2$ dir.

```
01:  /* 09prg05.c
02:      Kabarcık Sıralama (Bubble Sort) Algoritması ile bir
03:      dizinin elemanlarını büyükten küçüğe doğru sıralar */
04:
05:  #include <stdio.h>
06:
07:  #define n 10
08:
09:  int main(void)
10:  {
11:      int    a[n] = {100, -250, 400, 125 ,550, 900, 689, 450,
12: 347, 700};
13:      int    j,k,gecici;
14:
15:      /* Dizinin kendisi */
16:      printf("Once : ");
17:      for(k=0; k<n; k++)
18:          printf("%5d ",a[k]);
19:
20:      /* Sırala */
21:      for(k=0; k<n-1; k++)
22:          for(j=0; j<n-1; j++)
23:              if( a[j]<a[j+1] ){
24:                  gecici = a[j];
25:                  a[j] = a[j+1];
26:                  a[j+1] = gecici;
27:              }
28:
29:      /* Sıralama bitti */
30:      printf("\nSonra: ");
31:      for(k=0; k<n; k++)
32:          printf("%5d ",a[k]);
33:
34:      printf("\n");
35:
36:      return 0;
}
```

Once : 700	100	-250	400	125	550	900	689	450	347
Sonra: -250	900	700	689	550	450	400	347	125	100

10.5 Karakter Dizileri (Strings)

C dilinde, karakter dizileri oldukça sık kullanılır. Sadece karakter dizilerine özel olarak, karakter dizilerinin sonuna sonlandırıcı karakter olarak adlandırılan bir simge eklenir. Sonlandırıcı karakter, işlemlerin hızlı ve etkin bir biçimde yapılabilmesine olanak sağlar[2].

Sonlandırıcı karakter:

- dizinin bittiği yeri gösterir,
- ASCII tablosunun sıfır numaralı ('\\0') karakteridir.

Karakter dizilerine iki şekilde başlangıç değeri verilebilir.

```
char s[7] = {'d','e','n','e','m','e','\\0'};
```

yada

```
char s[7] = "deneme";
```

Birinci tanımlamada sonlandırıcı karakter programcı tarafından konmalıdır. İkinci tanımlamada ise buna gerek yoktur. Çünkü, sonlandırıcı karakter bu atamayla, derleyici tarafından eklenir.

NOT

```
char s[7] = "deneme";
```

ataması geçeli olmasına rağmen, aşağıdaki atama geçersizdir:

```
char s[7];  
char s = "deneme";
```

Karakter dizileri `gets()` fonksiyonu ile klavyeden okunabilir.

```
char ad[20];  
...  
gets(ad);
```

Karakter dizleri veya katarlar Bölüm 12'de daha ayrıntılı işlenecektir. Burada sadece iki basit örnek sunulmuştur. Program 10.6'da bir karakter dizisinin uzunluğunun nasıl bulunduğu, Program 10.7'de ise bir karakter dizisinin tersyüz edilişi gösterilmiştir. İnceleyiniz.

Program 10.6: Bir karakter dizisinin uzunluğunu bulur

```
01: /* 09prg06.c: Bir karakter dizisinin uzunluğunu bulur */
02:
03: #include <stdio.h>
04:
05: int main(void)
06: {
07:     char s[40];
08:     int k = 0;
09:
10:     /* diziyi oku */
11:     printf("Bir seyler yazın : ");
12:     gets(s);
13:
14:     /* sonlandırıcı karaktere kadar karakterleri say */
15:     while( s[k]!='\0' )
16:         k++;
17:
18:     printf("Dizinin uzunluğu : %d\n",k);
19:
20:     return 0;
21: }
```

ÇIKTI

```
Birseyler yazın : Marmara Universitesi
Dizinin uzunluğu : 21
```

Program 10.7: Bir karakter dizisinin tersini bulur

```
01: /* 09prg07.c: Bir karakter dizisini tersyüz eder */
02:
03: #include <stdio.h>
04:
05: int main(void)
06: {
07:     char s[40], gecici;
08:     int i, n;
09:
10:     /* diziyi oku */
11:     printf("Bir seyler yazın : ");
12:     gets(s);
13:
14:     /* sonlandırıcı karaktere kadar */
15:     for(n=0; s[n] != '\0'; n++)
16:         ;
17:
18:     for(i=0; i<n/2; i++){
19:         gecici = s[n-i-1];
20:         s[n-i-1] = s[i];
21:         s[i] = gecici;
22:     }
23:
24:     printf("Tersi : %s\n",s);
25:
26:     return 0;
27: }
```

ÇIKTI

Bir seyler yazın : Deneme
Tersi : emeneD

10.6 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. Bu durumda, genel olarak bir dizi şöyle gösterilir:

Tablo 10.1: 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 diziler tek boyuta indir generek bellekte tutulurlar. Tek indisli dizilerde olduğu gibi, çok indisli dizilere de başlangıç değeri vermek mümkün. Örneğin 3 satır ve 4 sütunlu ($3 \times 4 = 12$ elemanlı) bir *x* matrisinin elemanları şöyle tanımlanabilir:

```
int x[3][4] = {11,34,42,60, 72,99,10,50, 80,66,21,38};
```

Bu matris ekrana matris formunda yazılmak istendiğinde:

```
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 olacaktır.

Program 10.8 iki matrisin toplamını başka bir matrise aktarır. Matris

$$c_{ij} = a_{ij} + b_{ij}$$

toplamı formülü ile tanımlıdır. İnceleyiniz.

Program 10.8: İki matrisin toplamı

```
01: /* 09prg08.c: iki matrisin toplamı */
02:
03: #include <stdio.h>
04:
05: #define SAT 2
06: #define SUT 3
07:
08: int main()
09: {
10:     int a[SAT][SUT] = {5, 3, 7, 0, 1, 2};
11:
12:     int b[SAT][SUT] = {1, 2, 3, 4, 5, 6};
13:     int c[SAT][SUT];
14:     int i, j;
15:
16:     puts("A Matrisi:");
17:     for(i=0; i<SAT; i++){
18:         for(j=0; j<SUT; j++){
19:             printf("%4d",a[i][j]);
20:             printf("\n");
21:         }
22:
23:     puts("B Matrisi:");
24:     for(i=0; i<SAT; i++){
25:         for(j=0; j<SUT; j++){
26:             printf("%4d",b[i][j]);
27:             printf("\n");
28:         }
29:
30:     puts("\nC Matrisi:");
31:     for(i=0; i<SAT; i++){
32:         for(j=0; j<SUT; j++){
33:             c[i][j] = a[i][j] + b[i][j];
34:             printf("%4d",c[i][j]);
35:         }
36:         printf("\n");
37:     }
38:
39:     return 0;
40: }
```

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
```

Program $c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$ matrisin çarpımı başka bir matrise aktarır. Matris çarpımı formülü ile tanımlıdır.

Program 10.9: İki matrisin çarpımı

```
01: /* 10prg09.c: 3x3 boyutundaki iki kare matrisin çarpımı */
02:
03: #include <stdio.h>
04:
05: #define N 3
06:
07: int main()
08: {
09:     int a[N][N], b[N][N], c[N][N];
10:     int i,j,k,toplam;
11:
12:     puts("A Matrisini girin:");
13:     for(i=0; i<N; i++)
14:         for(j=0; j<N; j++)
15:             scanf("%d",&a[i][j]);
16:
17:     puts("B Matrisini girin:");
18:     for(i=0; i<N; i++)
19:         for(j=0; j<N; j++)
20:             scanf("%d",&b[i][j]);
21:
22:
23:     puts("\nC Matrisi:");
24:     for(i=0; i<N; i++){
25:         for(j=0; j<N; j++){
26:
27:             for(toplam=0, k=0; k<N; k++)
28:                 toplam += a[i][k]*b[k][j];
29:
30:             c[i][j] = toplam;
31:             printf("%4d",c[i][j]);
32:         }
33:         printf("\n");
34:     }
35:
36:     return 0;
37: }
```

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
```

10.7 Dizilerin Fonksiyonlarda Kullanılması

Diziler de sıradan değişkenler gibi bir fonksiyona parametere olarak aktratılabilir. Fakat, aktarma kuralı biraz farklıdır. Her zaman dizinin yanında boyutunun da bilinmesi gerekir.

Program 10.10'da, bir dizinin elemanlarının yazdırılması işi bir fonksiyona yaptırılmıştır. Fonksiyona parametre olarak dizinin yanında boyutu da ilave edilmiştir. İnceleyiniz.

Program 10.10: Bir dizinin yazdırılması

```
01: /* 10prg10.c: bir dizinin yazdırılması */
02:
03: #include <stdio.h>
04:
05: void dizi_yaz(float x[], int n);
06:
07: int main(){
08:
09:     float kutle[5]= { 8.471, 3.683, 9.107, 4.739, 3.918 };
10:
11:     dizi_yaz(kutle, 5);
12:
13:     return 0;
14: }
15:
16: void dizi yaz(float x[], int n)
17: {
18:     int i;
19:
20:     for(i=0; i<n; i++)
21:         printf("%7.3f", x[i]);
22:
23:     printf("\n");
24: }
```

ÇIKTI

8.471 3.683 9.107 4.739 3.918

Eğer dizi boyutu #define önişlemcisi ile belirtilirse boyutun ayrıca parametre olarak kullanılmasına gerek yoktur. Bu durumda Program 10.10 şöyle değiştirilebilir:

```
...
#define BOYUT 5
void dizi_yaz(float x[]);

void main(void)
{
    float kutle[BOYUT]= { 8.471, 3.683, 9.107, 4.739, 3.918 };
    dizi_yaz(kutle);
}
...
```

Program 10.3'de bir dizinin en büyük elemanının nasıl bulunduğu gösterilmişti. En büyük elemanı bulma işlemi bir fonksiyona nasıl yaptırıldığı Program 10.11'de gösterilmiştir.

Program 10.11: Bir dizinin en büyük elemanının bulunması

```
01: /* 10prg11.c
02:     Bir dizinin en büyük elemanının fonksiyonla bulunması
03: */
04:
05: #include <stdio.h>
06:
07: /* n elemanlı bir dizinin enbüyük elemanını gönderir */
08: int enBuyuk(int a[], int n)
09: {
10:     int k, en_buyuk_eleman;
11:
12:     /* ilk eleman en büyük kabul ediliyor */
13:     en_buyuk_eleman = a[0];
14:
15:     for(k=1; k<n; k++)
16:         if( a[k]>en_buyuk_eleman )
17:             en_buyuk_eleman = a[k];
18:
19:     return en_buyuk_eleman;
20: }
21:
22: int main()
23: {
24:     int a[10] = {100, -250, 400, 125, 550, 900, 689, 450,
25: 347, 700};
26:     int eb;
27:
28:     eb = enBuyuk(a,10);
29:
30:     printf("En büyük eleman = %d\n",eb);
31:
32:     return 0;
33: }
```

ÇIKTI

En büyük eleman = 900

Son olarak, bir kare matrisin iz (trace) değerini bulup ekrana yazan bir fonksiyon Program 10.12'de verilmiştir. Bir kare matrisin izi, matrisin asal köşegen üzerinde bulunan elemanların toplamı olarak tanımlıdır. Bu tanıma göre, aşağıdaki matrisin izi $2 + 8 + 4 = 14$ tür..

$$\begin{pmatrix} 2 & 1 & -5 \\ 3 & 8 & 6 \\ 7 & 1 & 4 \end{pmatrix}$$

İz matematiksel olarak şöyle gösterilir:

$$\text{İz}(\mathbf{A}) = a_{11} + a_{22} + \cdots + a_{nn} = \sum_{k=1}^n a_{kk}$$

Program 10.12: Bir matrisin izi

```
01: /* 10prgl2.c
02:     Bir 3x3 bir matrisin izinin fonksiyonla bulunması */
03:
04: #include <stdio.h>
05:
06: double iz(double a[][3], int);
07:
08: int main()
09: {
10:     double a[3][3], izA;
11:     int i,j;
12:
13:     puts("matrisi girin:");
14:     for(i=0; i<3; i++)
15:         for(j=0; j<3; j++)
16:             scanf("%lf",&a[i][j]);
17:
18:     izA = iz(a,3);
19:
20:     printf("matrisin izi = %lf\n",izA);
21:
22:     return 0;
23: }
24:
25: double iz(double a[][3], int n)
26: {
27:     int i;
28:     double toplam = 0.0;
29:
30:
31:     for(i=0; i<n; i++)
32:         toplam += a[i][i];
33:
34:     return toplam;
35: }
36:
37:
```

ÇIKTI

```
matrisi girin:
2  1  -5
3  8  6
7  1  4
matrisin izi = 14.000000
```

Matrisler, fonksiyonlara parametre olarak geçirilirken ikinci boyunda verildiğine dikkat edin.

Ders 11: Gösterici (Pointer) Kavramı

Giriş

Hemen hemen bütün programlama dillerinin temelinde *gösterici* (pointer) veri tipi bulunmaktadır. Bir çok dil gösterici kullanımını kullanıcıya sunmamıştır veya çok sınırlı olarak sunmuştur. Fakat C Programlama Dili'nde göstericiler yoğun olarak kullanılır. Hatta gösterici kavramı C dilinin bel kemiğidir. Kavranması biraz güç olan göstericiler için -latife yapıp- C kullanıcılarını "gösterici kullanabilenler ve kullanmayanlar" olmak üzere iki gruba ayıranlar da olmuştur. Özetle, bir C programcısı gösterici kavramını anlamadan C diline hakim olamaz.

Türkçe yazılan C kitaplarda *pointer* kelimesi yerine aşağıdaki ifadelerden biri karşılaşılabılır:

pointer = işaretçi = gösterici = gösterge

11.1 Değişken ve Bellek Adresi

Bilgisayarın ana belleği (RAM) sıralı kaydetme gözlemlerinden oluşmuştur. Her göze bir adres atanmıştır. Bu adreslerin değerleri 0 ila belleğin sahip olduğu üst değere bağlı olarak değişebilir. Örneğin 1GB MB bir bellek, $1024 * 1024 * 1024 = 1073741824$ adet gözden oluşur. Değişken tiplerinin bellekte işgal ettiği alanın bayt cinsinden uzunluğu `sizeof()` operatörüyle öğrenildiğini hatırlayın.

Bir programlama dilinde, belli bir tipte değişken tanımlanıp ve bir değer atandığında, o değişkene dört temel özellik eşlik eder:

1. değişkenin adı
2. değişkenin tipi
3. değişkenin sahip olduğu değer (içerik)
4. değişkenin bellekteki adresi

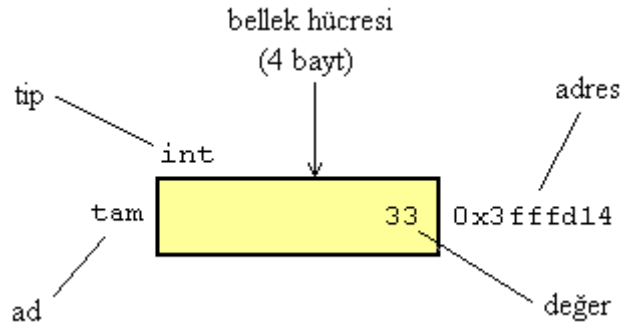
Örneğin `tam` adlı bir tamsayı değişkenini aşağıdaki gibi tanımladığımızı varsayalım:

```
int tam = 33;
```

Bu değişken için, `int` tipinde bellekte bir hücre ayrılır ve o hücreye 33 sayısı ikilik (binary) sayı sistemindeki karşılığı olan 4 baytlık (32 bitlik):

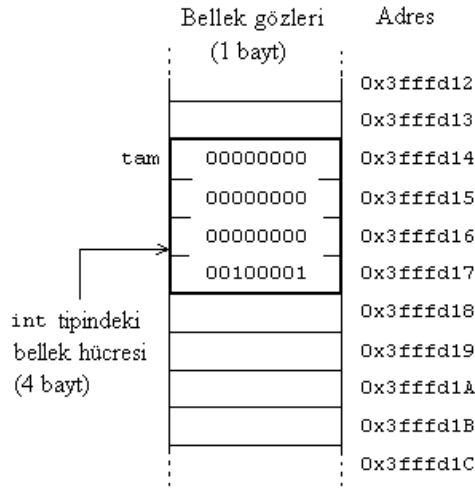
00000000 00000000 00000000 00100001

sayısı elektronik olarak yazılır. `tam` değişkenine ait dört temel özellik Şekil 11.1'deki gibi gösterilebilir:



Şekil 11.1: Bir değişkene eşlik eden dört temel özellik

Bellek adresleri genellikle onaltılık (hexadecimal) sayı sisteminde ifade edilir. `0x3fffd14` sayısı onluk (decimal) sayı sisteminde `67108116` sayısına karşık gelir. Bunun anlamı, `tam` değişkeni, program çalıştığı sürece, bellekte `67108116. - 67108120.` numaralı gözler arasındaki 4 baytlık hücreyi işgal edecek olmasıdır. Şekil 11.1'deki gösterim, basit ama anlaşılır bir tasvirdir. Gerçekte, `int` tipindeki `tam` değişkeninin bellekteki yerleşimi ve içeriği (değeri) Şekil 11.2'de gösterildiği gibi olacaktır.



Şekil 11.2: `tam` adlı değişkenin bellekteki gerçek konumu ve ikilik düzendeki içeriği

Değişkenin saklı olduğu adres, `&` karakteri ile tanımlı *adres operatörü* ile öğrenilebilir. Bu operatör bir değişkenin önüne konursa, o değişkenin içeriği ile değil adresi ile ilgileniliyor anlamına gelir.

Aşağıdaki program parçasının:

```
int tam = 33;

printf("icerik: %d\n", tam);
printf("adres : %p\n", &tam);
```

çıktısı:

```
icerik: 33
adres : 3fffd14
```

şeklindedir. Burada birinci satır `tam` değişkeninin içeriği, ikinci ise adresidir. Adres yazdırılırken `%p` tip belirleyicisinin kullanıldığına dikkat ediniz.

11.2 Gösterici Nedir?

Gösterici, bellek alanındaki bir gözün adresinin saklandığı değişkendir. Göstericilere veriler (yani değişkenlerin içeriği) değil de, o verilerin bellekte saklı olduğu hücrenin başlangıç adresleri atanır. Kısaca gösterici adres tutan bir değişkendir.

Bir gösterici, diğer değişkenler gibi, sayısal bir değişkendir. Bu sebeple kullanılmadan önce program içinde bildirilmelidir. Gösterici tipindeki değişkenler şöyle tanımlanır:

```
tip_adı *gösterici_adı;
```

Burada `tip_adı` herhangi bir C tip adı olabilir. Değişkenin önündeki `*` karakteri yönlendirme (indirection) operatörü olarak adlandırılır ve bu değişkenin veri değil bir adres bilgisini tutacağını işaret eder. Örneğin:

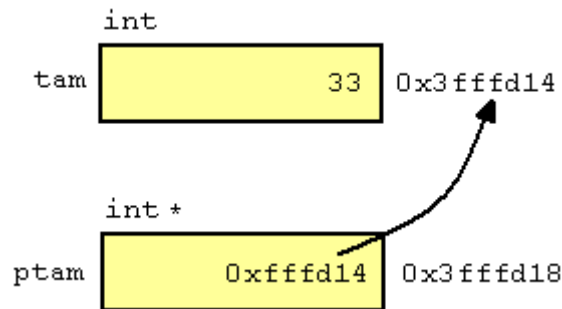
```
char *kr;           /* tek bir karakter için */
int *x;             /* bir tamsayı için */
float *deger, sonuc; /* deger gösterici tipinde, sonuc sıradan bir gerçel
değişkenler */
```

Yukarıda bildirilen göstericilerden; `kr` bir karakterin, `x` bir tamsayının ve `deger` bir gerçel sayının bellekte saklı olduğu yerlerin adreslerini tutar.

Bir göstericiye, bir değişkenin adresini atamak için adres operatörünü kullanabiliriz. Örneğin tamsayı tipindeki `tam` adlı bir değişken ve `ptam` bir gösterici olsun. Derleyicide, aşağıdaki gibi bir atama yapıldığında:

```
int *ptam, tam = 33;
.
.
.
ptam = &tam;
```

`ptam` göstericisinin `tam` değişkeninin saklandığı adresi tutacaktır. Bu durum Şekil 11.3'deki gibi tasvir edilir.



Şekil 11.3: Göstericinin bir değişkenin adresini göstermesi

Şekil 11.3'deki gösterimde, `ptam` göstericisinin içeriği `tam` değişkeninin içeriği (33) değil adresidir (0x3fffd14). Ayrıca, `ptam` değişkeni, bellekte başka bir hücrede saklandığına ve bu hücrenin `int` değil `int *` tipinde bir bölge olduğuna dikkat ediniz. Buraya kadar anlatılanlar, Program 11.1'de özetlenmiştir.

Program 11.1: Bir değişkenin içeriğini ve adresini ekrana yazdırma

```
01: /* 10prg01.c: ilk gösterici programı */
02:
03: #include <stdio.h>
04:
05: int main()
06: {
07:     int *ptam, tam = 33;
08:
09:     ptam = &tam;
10:
11:     printf("tam: icerik = %d\n", tam);
12:     printf("tam:  adres = %p\n", &tam);
13:     printf("tam:  adres = %p\n", ptam);
14:
15:     return 0;
16: }
```

7. satırda değişkenler bildirilmiştir. 9. satırdaki atama ile `tam` değişkeninin *adres*i, `ptam` göstericisine atanmıştır. Bu satırdan itibaren `ptam`, `tam` değişkeninin gösterir. 11. satırda `tam`'ın içeriği (33 sayısı), 12. ve 13. satırda `tam`'ın adresi, `%p` tip karakteri ile, ekrana yazdırılmıştır. Ekran çıktısı incelendiğinde, `&tam` ve `ptam` içereriğinin aynı anlamda olduğu görülür.

ÇIKTI

```
tam: icerik = 33
tam:  adres = 0x3fffd14
tam:  adres = 0x3fffd14
```

`tam` adlı değişkenin içeriğine `ptam` gösterici üzerinde de erişilebilir. Bunun için program içinde `ptam` değişkeninin önüne yönelendirme operatörü (*) koymak yeterlidir. Yani `*ptam`, `tam` değişkeninin adresini değil içeriğini tutar. Buna göre:

```
*ptam = 44;
```

komutuyla, `ptam`'ın adresini tuttuğu hücreye 44 değeri atanır. Bu durum, Program 11.2'de gösterilmiştir.

Program 11.2: Bir değişkenin içeriğini ve adresini ekrana yazdırma

```
01: /* 10prg02.c: ikinci gösterici programı */
02:
03: #include <stdio.h>
04:
05: int main()
06: {
07:     int *ptam, tam = 33;
08:
09:     ptam = &tam;    /* ptam -> tam */
10:
11:     printf("&tam = %p\n", &tam);
12:     printf("ptam = %p\n", ptam);
13:     printf("\n");
14:
15:     printf("tam = %d\n", tam);
16:     printf("*ptam = %d\n", *ptam);
17:     printf("\n");
18:
19:     *ptam = 44;    /* tam = 44 anlamında */
20:
21:     printf("tam = %d\n", tam);
22:     printf("*ptam = %d\n", *ptam);
23:
24:     return 0;
25: }
```

ÇIKTI

```
&tam = 0x3fffd14
ptam = 0x3fffd14

tam = 33
*ptam = 33

tam = 44
*ptam = 44
```

Özetle `ptam = &tam` atamasıyla:

- `*ptam` ve `tam`, `tam` adlı değişkenin içeriği ile ilgilidir.
- `ptam` ve `&tam`, `tam` adlı değişkenin adresi ile ilgilidir.
- `*` yönlendirme ve `&` adres operatörüdür.

11.3 Gösterici Aritmetiği

Göstericiler kullanılırken, bazen göstericinin gösterdiği adres taban alınıp, o adresten önceki veya sonraki adreslere erişilmesi istenebilir. Bu durum, göstericiler üzerinde, aritmetik işlemcilerin kullanılmasını gerektirir. Göstericiler üzerinde yalnızca toplama (+), çıkarma (-), bir artırma (++) ve bir eksiltme (--) operatörleri işlemleri yapılabilir.

Aşağıdaki gibi üç tane gösterici bildirilmiş olsun:

```
char    *kar;  
int     *tam;  
double  *ger;
```

Bu göstericiler sırasıyla, bir karakter, bir tamsayı ve bir gerçel sayının bellekte saklanacağı adreslerini tutar. Herhangi bir anda, tuttukları adresler de sırasıyla 10000 (0x2710), 20000 (0x4e20) ve 30000 (0x7530) olsun. Buna göre aşağıdaki atama işlemlerinin sonucu:

```
kar++;  
tam++;  
ger++;
```

sırasıyla 10001 (0x2711), 20004 (0x4e24) ve 30008 (0x7538) olur. Bir göstericiye ekleme yapıldığında, o anda tuttuğu adres ile eklenen sayı doğrudan toplanmaz. Böyle olsaydı, bu atamaların sonuçları sırasıyla 10001, 20001 ve 30001 olurdu. Gerçekte, göstericiye bir eklemek, göstericinin gösterdiği yerdeki veriden hemen sonraki verinin adresini hesaplamaktır.

Genel olarak, bir göstericiye n sayısını eklemek (veya çıkarmak), bellekte gösterdiği veriden sonra (veya önce) gelen n . elemanın adresini hesaplamaktır. Buna göre aşağıdaki atamalar şöyle yorumlanır.

```
kar++;           /* kar = kar + sizeof(char) */  
tam = tam + 5;   /* tam = tam + 5*sizeof(int) */  
ger = ger - 3;   /* ger = ger - 3*sizeof(double) */
```

Program 11.3, bu bölümde anlatılanları özetlemektedir. İnceleyiniz.

Program 11.3: Gösterici aritmetiği

```
01: /* 10prg03.c: gösterici aritmetiği */  
02:  
03: #include <stdio.h>  
04:  
05:  
06: int main()  
07: {  
08:     char    *pk, k = 'a';  
09:     int     *pt, t = 22;  
10:     double  *pg, g = 5.5;  
11:  
12:     pk = &k;  
13:     pt = &t;  
14:     pg = &g;  
15:  
16:     printf("Onceki adresler: pk= %p pt= %p pg= %p \n",  
17: pk, pt, pg);  
18:  
19:     pk++;  
20:     pt--;  
21:     pg = pg + 10;  
22:  
23:     printf("Sonraki adresler: pk= %p pt= %p pg= %p \n",  
24: pk, pt, pg);  
25:     return 0;  
}
```

ÇIKTI

```
Önceki adresler: pk= 0xbfbbe88f  pt= 0xbfbbe888  pg=
0xbfbbe880
Sonraki adresler: pk= 0xbfbbe890  pt= 0xbfbbe884  pg=
0xbfbbe8d0
```

11.4 Gösterici ve Diziler Arasındaki İlişki

C dilinde göstericiler ve diziler arasında yakın bir ilişki vardır. Bir dizinin adı, dizinin ilk elemanının adresini saklayan bir göstericidir. Bu yüzden, bir dizinin herhangi bir elemanına gösterici ile de erişilebilir. Örneğin:

```
int kutle[5], *p, *q;
```

şeklinde bir bildirim yapılsın. Buna göre aşağıda yapılan atamalar geçerlidir:

```
p = &kutle[0];    /* birinci elemanın adresi p göstericisine atandı */
p = kutle;        /* birinci elemanın adresi p göstericisine atandı */
q = &kutle[4];    /* son elemanın adresi q göstericisine atandı */
```

İlk iki satırdaki atamalar aynı anlamdadır. Dizi adı bir gösterici olduğu için, doğrudan aynı tipteki bir göstericiye atanabilir. Ayrıca, *i* bir tamsayı olmak üzere,

```
kutle[i];
```

ile

```
*(p+i);
```

aynı anlamdadır. Bunun sebebi, *p* göstericisi *kutle* dizisinin başlangıç adresini tutmuş olmasıdır. *p+i* işlemi ile *i+1*. elemanın adresi, ve **(p+i)* ile de bu adresteki değer hesaplanır.

NOT

Bir dizinin, *i*. elemanına erişmek için **(p+i)* işlemi yapılması zorunludur. Yani

```
*p+i;    /* p nin gösterdiği değere (dizinin ilk elemanına) i
sayısını ekle */
*(p+i); /* p nin gösterdiği adresten i blok ötedeki sayıyı
hesapla */
```

anlamındadır. Çünkü, *** operatörü *+* operatörüne göre işlem önceliğine sahiptir.

Program 11.4'de tanımlanan fonksiyon kendine parameter olarak gelen *n* elemanlı bir dizinin aritmetik ortalamasını hesaplar.

Program 11.4: Bir dizi ile gösterici arasındaki ilişki

```
01: /* 10prg04.c: gösterici dizi ilişkisi */
02:
03: #include <stdio.h>
04:
05: double ortalama(double dizi[], int n);
06:
07: int main()
08: {
09:
10:     double a[5] = {1.1, 2.2, 3.3, 4.4, 5.5};
11:     double o;
12:
13:     o = ortalama(a,5);
14:
15:     printf("Dizinin ortalaması = %lf\n",o);
16:
17:     return 0;
18: }
19:
20: double ortalama(double dizi[], int n)
21: {
22:     double *p, t=0.0;
23:     int i;
24:
25:     p = dizi;    /* veya p = &dizi[0] */
26:
27:     for(i=0; i<n; i++)
28:         t += *(p+i);
29:
30:     return (t/n);
31: }
```

ÇIKTI

```
Dizinin ortalaması = 3.300000
```

20. - 31. satırda tanımlanan fonksiyon aşağıdaki gibi de yazılabilirdi:

```
double ortalama(double dizi[], int n)
{
    double *p, t=0.0;

    for(p=dizi; p < &dizi[n]; p++)
        t += *p;

    return (t/n);
}
```

Bu fonksiyonda, döngü sayacı için (*i* değişkeni) kullanılmayıp, döngü içinde dizinin başlangıç adresi *p* göstericisine atanmış ve koşul kısmında adres karşılaştırılması yapılmıştır. Bu durumda döngü, *p*'nin tuttuğu adresten başlar, ve *p*'nin adresi dizinin son elemanının adresinden (*&dizi[n-1]*) küçük veya eşit olduğu sürece çevrim yinelenir.

11.5 Fonksiyon Parametresi Olan Göstericiler

C (ve C++) programlama dilinde fonksiyon parametreleri değer geçerek (pass by value) yada adres geçerek (pass by reference) olarak geçilebilir. Bölüm 8'deki uygulamalarda fonksiyonlara parametreler değer geçerek taşınmıştı. Bu şekilde geçirilen parametreler, fonksiyon içerisinde değiştirilse bile, fonksiyon çağırıldıktan sonra bu değişim çağrılan yerdeki değerini değiştirmez. Fakat, bir parametre adres geçerek aktarılsa, fonksiyon içindeki değişiklikler geçilen parametreyi etkiler. Adres geçerek aktarım, gösterici kullanmayı zorunlu kılar.

Örneğin, Program 11.5'de fonksiyonlara değer ve adres geçerek aktarımın nasıl yapılacağı gösterilmiştir.

Program 11.5: *Bir değişkenin içeriğini ve adresini ekrana yazdırma*

```
01: /* 10prg05.c: Değer geçerek ve adres geçerek aktarım */
02:
03: #include <stdio.h>
04:
05: void f1(int ); /* iki fonksiyon */
06: void f2(int *);
07:
08: int main()
09: {
10:     int x = 55;
11:
12:     printf("x in degeri,\n");
13:     printf("Fonksiyonlar cagrilmadan once: %d\n",x);
14:
15:     /* f1 fonksiyonu çağırılıyor...*/
16:     f1(x);
17:     printf("f1 cagirildikten sonra      : %d\n",x);
18:
19:
20:     /* f2 fonksiyonu çağırılıyor...*/
21:     f2(&x);
22:     printf("f2 cagirildikten sonra      : %d\n",x);
23:
24:     return 0;
25: }
26:
27:
28: /* Değer geçerek aktarım */
29: void f1(int n){
30:     n = 66;
31:     printf("f1 fonksiyonu icinde      : %d\n",n);
32: }
33:
34: /* Adres geçerek aktarım */
35: void f2(int *n){
36:     *n = 77;
37:     printf("f2 fonksiyonu icinde      : %d\n",*n);
38: }
```

5. ve 6. satırlarda kendine geçilen parametrenin değerini alan `f1` fonksiyonu ve parametrenin adresini alan `f2` adlı iki fonksiyon örneği belirtilmiştir. 11. satırdaki `x` değişkeni 16. ve 21. satırlarda, `f1(x)` ve `f2(&x)` fonksiyonlarına, sırasıyla değer ve adres geçerek aktarılmıştır. `f1` içinde `x` (`n = 66;` işlemi ile) değişime uğramış, fakat çağırılma işleminin

sonucunda, x 'in değeri değişmemiştir. Ancak f_2 içinde x 'in ($*n = 77$ işlemi ile) değişimi, çağırıldıktan sonrada korunmuştur. Yani, adres geçerek yapılan aktarımda, f_2 'ye aktarılan değer değil adres olduğu için, yollanan x parametresi f_2 içinde değişikliğe uğrayacak ve bu değişim çağırıldığı 21. satırdan itibaren devam edecektir.

ÇIKTI

```
x in degeri,  
Fonksiyonlar cagrilmadan once: 55  
f1 fonksiyonu icinde      : 66  
f1 cagirildiktan sonra    : 55  
f2 fonksiyonu icinde      : 77  
f2 cagirildiktan sonra    : 77
```

Program 11.6'da iki tamsayı değişkeninin nasıl takas (swap) edileceği gösterilmiştir. Bu işlemi C porgramlama dilinde, eğer değişkenler global olarak bildirilmemişse, gösterici kullanmadan bu işlemi yapmak imkansızdır.

Program 11.6: İki tamsayının birbiri ile takas edilmesi

```
01: /* 10prg06.c: iki sayının birbiri ile takas edilmesi */  
02:  
03: #include <stdio.h>  
04:  
05: void takas(int *, int *);  
06:  
07: int main()  
08: {  
09:     int a, b;  
10:  
11:     a=22; b=33;  
12:  
13:     printf("takas oncesi : a=%d    b=%d\n",a,b);  
14:  
15:     takas(&a, &b);  
16:  
17:     printf("takas sonrasi: a=%d    b=%d\n",a,b);  
18:  
19:  
20:     return 0;  
21: }  
22:  
23:  
24: void takas(int *x, int *y)  
25: {  
26:     int z;  
27:  
28:     z = *x;  
29:     *x = *y;  
30:     *y = z;  
31: }
```

ÇIKTI

```
takas oncesi : a=22    b=33  
takas sonrasi: a=33    b=22
```

11.6 Geri Dönüş Değeri Gösterici Olan Fonksiyonlar

Fonksiyonların geri dönüş değeri bir gösterici olabilir. Bu durumda fonksiyon bir değer değil adres döndürecek demektir.

Program 11.7'da önce bir dizinin indisleri, dizi değerleri ve dizi elemanlarının adresleri ekrana basılır. Daha sonra, `maxAdr()`; fonksiyonu ile dizinin en büyük elemanının adresi döndürülür. *Bu örnek program, göstericilerin gücünü çok zarif bir biçimde bize sunmaktadır.* Lütfen inceleyiniz.

Program 11.7: Bir dizinin en büyük elemanının adresini öğrenmek

```
01: /* 10prg07.c: geri donus degeri gosterici olan fonksiyon
02: */
03:
04: #include <stdio.h>
05:
06: double* maxAdr(double a[], int boyut){
07:     double ebd = a[0];
08:     double *eba = &a[0];
09:     int i;
10:     for(i=1; i<boyut; i++){
11:         if(a[i]>ebd){
12:             ebd = a[i]; // en büyük deger
13:             eba = &a[i]; // en büyük adres
14:         }
15:     }
16:     return eba;
17: }
18:
19:
20: int main()
21: {
22:     double x[6] = {1.1, 3.3, 7.1, 5.4, 0.2, -1.5};
23:     double *p;
24:     int k;
25:     // indis, dizi ve adresini ekrana bas
26:     for(k=0; k<6; k++){
27:         printf("%d %lf %p\n", k, x[k], &x[k]);
28:     }
29:
30:     p = maxAdr(x,6);
31:
32:     printf("En büyük deger: %lf\n", *p);
33:     printf("En büyük adres: %p \n", p);
34:     printf("En büyük konum: %d \n", int(p-&x[0]));
35:
36:     return 0;
}
```

Dizi elemanları 21. satırda belirlenir. Bu dizinin indisleri, değerleri ve adresleri 26. satırda ekrana basılmıştır. En büyük elemanın adresi 29. satırdaki `p = maxAdr(a, 6);` ile `p` göstericisine atanmıştır. 5. satırda bildirilen `maxAdr()`; fonksiyonu, en büyük elemanın adresini hesaplayıp çağrılan yere gönderir. Burada dikkat edilmesi gereken husus, fonksiyonun dönüş değerinin yerel `eba` göstericisi olmasıdır. `eba` göstericisi 12. satırda hesaplanan ve fonksiyon parametresi olan dizinin en büyük elemanın adresini tutmaktadır. Son olarak, fonksiyon çağırıldıktan sonra, `p` göstericisinin gösterdiği değer, tuttuğu adres ve dizinin birinci elemanına göre konumu (indisi) ekrana basılmıştır. Indis

hesabı `int(p-&x[0])` işlemi ile yapılabilir. Bu aslında, `p` göstericinin tuttuğu adres ile dizinin ilk elemanının adresi arasındaki farktır. Sonuç yine bir adres olduğu için tamsayı değer elde etmek için `int()` takısı kullanılmıştır. Netice itibarıyla bir fonksiyon ile üç şey aynı anda öğrenilmiş olur.

ÇIKTI

```
0 1.100000 0x7fff41b29ec0
1 3.300000 0x7fff41b29ec8
2 7.100000 0x7fff41b29ed0
3 5.400000 0x7fff41b29ed8
4 0.200000 0x7fff41b29ee0
5 -1.500000 0x7fff41b29ee8

En büyük deger: 7.100000
En büyük adres: 0x7fff41b29ed0
En büyük konum: 2
```

11.7 Fonksiyon Göstericileri

Fonksiyon göstericileri, gösterici (pointer) kavramının gücünü gösterin diğer bir uygulama alanıdır. Dizilerde olduğu gibi, fonksiyon adları da sabit göstericidir.

Fonksiyon betiğinin (kodlarının) bellekte bir adreste tutulduğu şeklinde düşünebiliriz. Fonksiyon göstericisi basit olarak fonksiyon adının saklandığı bellek adresini tutan bir göstericidir. *Fonksiyon göstericileri sayesinde fonksiyonlar başka fonksiyonlara parametre olarak aktarılabilir.*

Fonksiyon adının bellekte yer işgal ettiği şöyle öğrenilebilir:

```
int f(int);          /* fonksiyon bildirimi */
int (*pf)(int);      /* fonksiyon göstericisi bildirimi */
pf = &f;             /* f'nin adresini pf'ye ata! */
```

Program 11.8: Bir fonksiyonun 'adresini' iki yoldan öğrenme

```
01: /* 10prg08.c: Bir fonksiyonun 'adresini' öğrenme */
02:
03: #include <stdio.h>
04:
05: int f(int n){
06:     int f=1, i;
07:     for(i=1; i<n; i++)
08:         f*=i;
09:     return f;
10: }
11:
12: int main()
13: {
14:     int (*pf)(int);
15:     pf = &f;
16:
17:     printf("Fonksiyonun adresi = %p\n", &f);
18:     printf("Fonksiyonun adresi = %p\n", pf);
19:
20:     return 0;
21: }
```

ÇIKTI

Fonksiyonun adresi = 0x4005b0
Fonksiyonun adresi = 0x4005b0

Aşağıdaki ikinci örnekte, bir fonksiyon diğer fonksiyona parametre olarak geçirilmiş ve sayısal türevi hesaplanmıştır. Türev hesaplanırken merkezi fark yaklaşımı (central difference approximation) yöntemi kullanılmıştır.

NOT

mfy yönteminde $f(x)$ fonksiyonunun (h küçük bir değer olmak üzere) Taylor açılımları şöyledir:

$$\begin{aligned} f(x+h) &= f(x) + h*f'(x) + \frac{h^2*f''(x)}{2!} + \frac{h^3*f'''(x)}{3!} + \dots \\ f(x-h) &= f(x) - h*f'(x) + \frac{h^2*f''(x)}{2!} - \frac{h^3*f'''(x)}{3!} + \dots \end{aligned}$$

-

$$f(x+h) - f(x-h) = 2*h*f'(x) + O(h^3)$$

Burada $O(h^3)$ 'lü terimler ihmal edilirse birinci türev yaklaşık olarak:

$$f'(x) = [f(x+h) - f(x-h)] / 2h$$

formülü ile hesaplanır.

Program 11.9: Türev alan fonksiyon

```
01: #include <stdio.h>
02:
03: double f(double);
04: double turev( double (*)(double), double);
05:
06: int main()
07: {
08:
09:     double x = 1.1;
10:
11:     printf("Fonksiyon x = %lf deki degeri = %lf\n", x,
12: f(x));
13:     printf("Fonksiyon x = %lf deki turevi = %lf\n", x,
14: turev(f, x) );
15:
16:     return 0;
17: }
18:
19:
20: // türevi hesaplanacak fonksiyon
21: double f(double x){
22:     return x*x*x - 2*x + 5.;
23: }
24: // sayısal türev alan fonksiyon
25: double turev( double (*fonk)(double), double x){
26:     double h = 1.0e-3;
27:     return (fonk(x+h)-fonk(x-h)) / (2*h);
28: }
```

ÇIKTI

```
Fonksiyon x = 1.100000 deki degeri = 4.131000
Fonksiyon x = 1.100000 deki turevi = 1.630001
```

11.8 NULL Gösterici

Bir göstericinin bellekte herhangi bir adresi göstermesi, veya önceden göstermiş olduğu adres iptal edilmesi istemirse `NULL` sabiti kullanılır. Bu sabit derleyicide ASCII karakter tablosunun ilk karakteridir ve `'\0'` ile sembolize edilir.

```
int *ptr, a = 12;
.
.
ptr = &a;      /* ptr bellekte a değişkenin saklandığı yeri gösteriyor */
.
.
ptr = NULL;    /* ptr bellekte hiç bir hücreyi göstermiyor */
*ptr = 8       /* hata! NULL göstericinin gösterdiği yere bir değer atanamaz */
```

11.9 void Tipindeki Göstericiler

`void` göstericiler herhangi bir veri tipine ait olmayan göstericilerdir. Bu özelliğinden dolayı, `void` gösterici genel gösterici (generic pointer) olarak da adlandırılır.

`void` göstericiler, `void` anahtar sözcüğü ile bildirilir. Örneğin:

```
void *adr;
```

gibi.

`void` göstericiler yalnızca adres saklamak için kullanılır. Bu yüzden diğer göstericiler arasında atama işlemlerinde kullanılabilir. Örneğin aşağıdaki atamada derleyici bir uyarı veya hata mesajı vermez:

```
void *v;
char *c;
.
.
.
v = c;    /* sorun yok !*/
```

Program 11.10'de `void` tipindeki bir göstericinin, program içinde, farklı tipteki verileri nasıl göstereceği ve kullanılacağı örneklenmiştir. İnceleyiniz.

Program 11.10: *void* gösterici ile farklı tipteki verileri gösterme

```
01: /* 10prg10.c: void gosterici (generic pointer) uygulaması
02: */
03:
04: #include <stdio.h>
05:
06: int main()
07: {
08:     char    kar = 'a';
09:     int      tam = 66;
10:     double   ger = 1.2;
11:     void     *veri;
12:
13:     veri = &kar;
14:     printf("veri -> kar: veri  %c  karakter degerini
15: gosteriyor\n", *(char *) veri);
16:
17:     veri = &tam;
18:     printf("veri -> tam: simdi veri  %d  tamsayi degerini
19: gosteriyor\n", *(int *) veri);
20:
21:     veri = &ger;
22:     printf("veri -> ger: simdi de veri  %lf  gercel sayi
23: degerini gosteriyor\n", *(double *) veri);
24:
25:     return 0;
26: }
```

ÇIKTI

```
veri -> kar: veri  a  karakter degerini gosteriyor
veri -> tam: simdi veri  66  tamsayi degerini gosteriyor
veri -> ger: simdi de veri  1.200000  gercel sayi degerini
gosteriyor
```

Benzer olarak, fonksiyon parameterelerinin kopyalanması sırasında da bu türden atama işlemleri kullanılabilir. Uygulamada, tipten bağımsız adres işlemlerinin yapıldığı fonksiyonlarda, parametre değişkeni olarak *void* göstericiler kullanılır. Örneğin

```
void free (void *p)
{
    .
    .
    .
}
```

Parametresi *void *p* olan *free* fonksiyonu, herhangi türden gösterici ile çağrılabilir.