C Programlama Dili'ne Giriş

Ders 8: *Fonksiyonlar I (Alt programlar)*

```
###################- (%95)
```

En son güncelleme: Wed, 30 Nov 2011 13:22:02 +0200

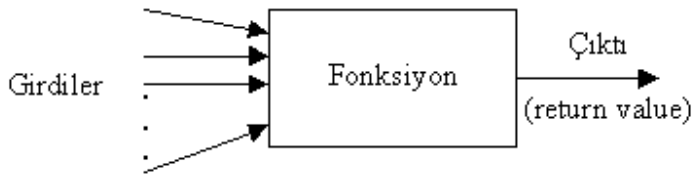
Giriş

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 (Bölüm 9) *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 parametreleri (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 *parametre* 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, kullanıcısına bu türden fonksiyon yazmasına izin verir. C dilinde hazırlanan bir fonksiyonun genel yapısı şöyledir:

```
FonksiyonTipi FonksiyonAdı(parametre listesi)
parametrelerin 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

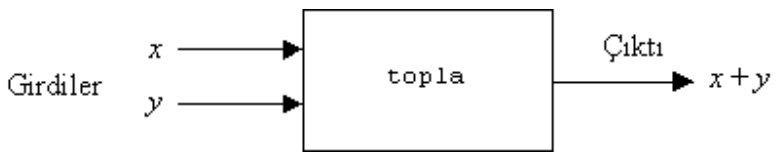
```
/* modern biçim */
int topla(int x, int y)
{
   int sonuc;
   sonuc = x + y;
   return sonuc;
}
```

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) deyimi 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 Parametre ve Argüman Kavramı

Fonksiyon bildiriminde, fonksiyona girdi olarak, kullanılan değişkenlere *parametre* denir.
Argüman ise fonksiyon çağrılırken gönderilen değere(lere) verilen addır.
Buna göre

```
/* Fonksiyon bildiriliyor ... */
int topla(int x, int y)  // Burada x ve y parametre
{
   return (x+y);
}

.
.
/* Fonksiyon çağrılıyor ... */
t = topla(9, 6);     // Burada 9 ve 6 argüman
.
.
```

8.3 Fonksiyon Bildirimi

Bir fonksiyonun bildirimi iki türlü yapılır:

1. *Ana programdan önce*:

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

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

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

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 parametre isimlerinin yazılması zorunlu değildir. Sadece 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 üzeride özetlenmiştir.

Program 8.1: `topla` *fonksiyonunun ana programda kullanılması*

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

ÇIKTI

```
Iki sayi girin : 5 12
5 ve 12 nin toplami  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 gösterir */
02:
03: #include <stdio.h>
04:
05:  int topla( int x, int y ){
06:     return (x+y);
07:  }
08:
09: int main(void)
10: {
11:     int toplam,a,b;
12:
13:     printf("Iki sayi girin : ");
14:     scanf("%d %d",&a,&b);
15:
16:     toplam = topla(a,b);
17:
18:     printf("%d ve %d nin toplami  %d dir.\n", a,b,toplam);
19:
20:   return 0;
21: }
```

8.4 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` deyimi 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 yil girin: ");
12:   scanf("%d",&yil);
13:
14:   if( artik_yil(yil) )
15:       printf("%d artik yil\n",yil);
16:   else
17:       printf("%d artik yil degil\n",yil);
18: }
19:
20: /* yil 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 yil girin: 1996
1996 artik yil
```

8.5 `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: /* 08prg03.c: Basit bankamatik simulasyonu.
02:    İstenen para miktarını 20, 10 ve 5'lik birimlere böler
03:    ve sonucu ekrana gösterir. */
04:
05: #include <stdio.h>
06:
07: void bankamatik(int para)
08: {
09:   int a,yirmilik,onluk,beslik;
10:
11:   a = para;
12:
13:   if(a%5==0)
14:   {
15:        yirmilik = a/20;
16:        a -= yirmilik*20;
17:
18:        onluk = a/10;
19:        a -= onluk*10;
20:
21:        beslik = a/5;
22:        a -= beslik*5;
23:
24:        printf("\nYirmilik = %d",yirmilik);
25:        printf("\nOnluk    = %d",onluk);
26:        printf("\nBeslik   = %d\n",beslik);
27:     }
28:     else
29:        printf("Girilen miktar 5 YTL ve katlari olmali!\a\n");
30:
31:       /* return deyimi yok !*/
32: }
33:
34: int main()
35: {
36:    int miktar;
```

```
37:
38:     printf("Cekilecek para miktari (YTL) = ");
39:     scanf("%d",&miktar);
40:
41:     bankamatik(miktar);  /* fonksiyon bir değişkene atanmamış ! */
42:
43:    retrun 0;
44: }
```

ÇIKTI

```
Cekilecek para miktari = 135

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

ÇIKTI

```
Cekilecek para miktari = 456
Girilen miktar 5 YTL ve katlari olmali!
```

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

```
void bankamatik(int para)
{
   ...
}
```

```
bankamatik(int para)
{
   ...
}
```

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*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

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

8.6 Fonksiyon Parametreleri

Fonksiyon parametreleri klasik ve modern olmak üzere iki türlü tanımanabilir. 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, parametre 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");
}
```

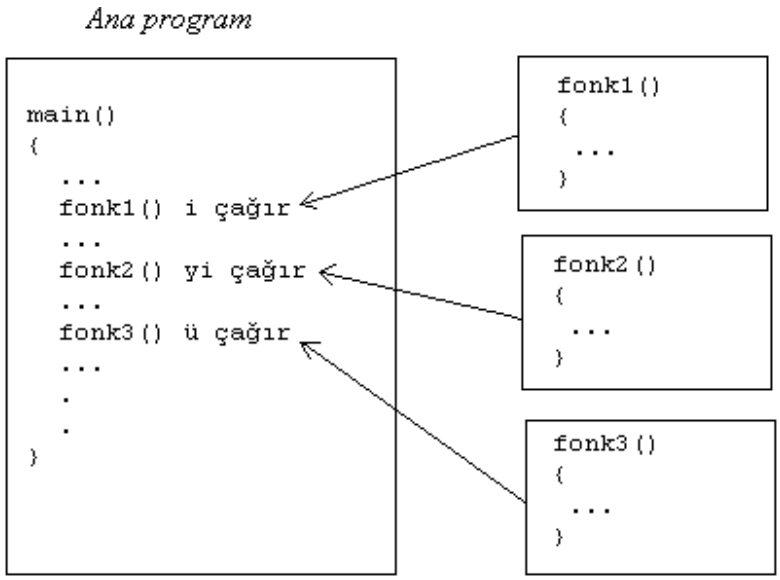
yada

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

8.7 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 çağrılması zorunlu değildir. Bir fonksiyon başka bir fonksiyon tarafından da çağrılabilir. Bu tür kullanıma dair bir örnek Program 8.5'de verilmiştir. `yilin_gunu` fonksiyonu, verilen bir tarihin yılın kaçıncı günü olduğunu hesaplar ve çağrıldığı yere gönderir. İnceleyiniz.

Program 8.5: *bir fonksiyonun başka bir fonksiyon tarafından çağrılması*

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

```
36:       case 10: ygun += 31;
37:       case  9: ygun += 30;
38:       case  8: ygun += 31;
39:       case  7: ygun += 31;
40:       case  6: ygun += 30;
41:       case  5: ygun += 31;
42:       case  4: ygun += 30;
43:       case  3: ygun += 31;
44:       case  2: ygun += 28 + artik_yil(yil);  /* 28+1 veya 28+0 */
45:       case  1: ygun += 31;
46:     }
47:
48:    return ygun;
49: }
```

ÇIKTI

```
01 08  2003 yilinin
213. gunudur
```

8.8 Makro Fonksiyon Tanımlaması

Başlık dosyalarında, bol miktarda makro fonksiyon uygulamalarına rastlanır. Makro tanımlaması `#define` önişlemci 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. Makro fonksiyon tanımlamaları Bölüm 20'de tekrar ele alınacaktır.

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
```