- continue Deyimi

Bir döngü içerisinde continue deyimi ile karşılaşılırsa, 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:** *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)
```

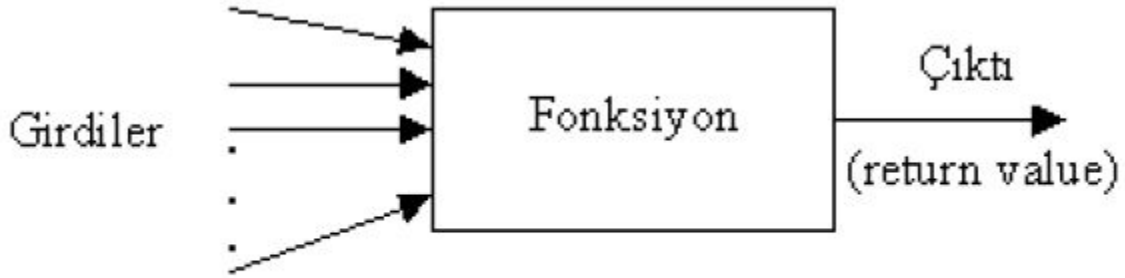
# FONKSIYONLAR I (ALT PROGRAMLAR)

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.

- Fonksiyon Kavramı

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




**Şekil 8.1:** *Bir fonksiyonun kutu gösterimi*

- Fonksiyon Bildirimi
- 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.

- 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 harffonksiyonunda 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';
}
```

**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
```

- 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.

- Fonksiyon Parametreleri

Fonksiyon parametreleri (argümanları) 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;
}
```

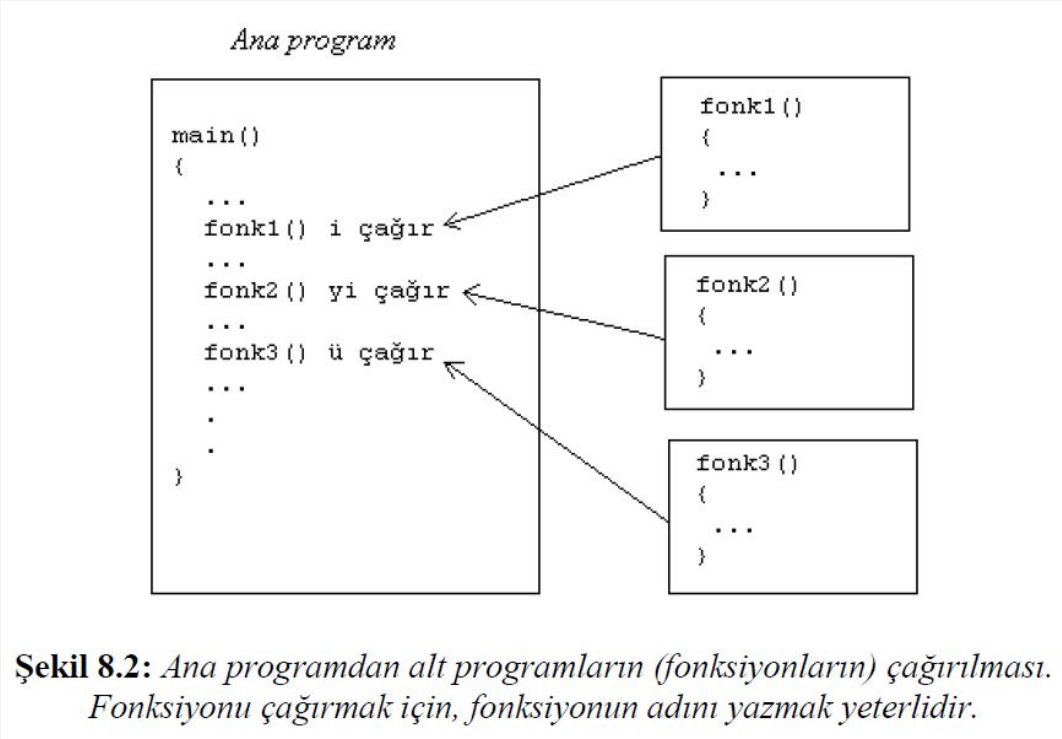
şeklinde yada:

```
int faktoriyel(int n) /* modern biçim */
{
int i=1, f=1;
while(i<=n) f *= i++;
return f;
}
```

şeklinde yazılabilir.

- 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.*

- 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.

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
```