

C Programlama Dili'ne Giriş

Ders 9: *Fonksiyonlar II (Alt programlar)*

#####-- (%90)

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

- [Giriş](#)
- [9.1 Esnek Argümanlı Fonksiyonlar](#)
- [9.2 main Fonksiyonu](#)
- [9.3 main Fonksiyonuna Parametre Aktarımı](#)
- [9.4 Komut Satırı Örnekleri](#)

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, $x = 1.7$ 'de, $P(x) = 1 - 2x$ 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++ ve Fortran 90'da olduğu gibi, C Programlama Dili kullanıcılarına argümanları *esnek* olarak geçirme imkanı verir. Bunun anlamı, belli kurallar sağlandığında, `p()` fonksiyonu aşağıdaki gibi çağırılabilmesidir:

```
/* x a b */
sonuc = p(1.7, 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
-----------------	----------

va_list	Ardışık esnek argümler için tip belirleyici
va_start(ap, n)	va_list tipinde bildirilmiş ap göstericisi için bellekten n elemanlı yer ayırır.
va_arg(ap, tip)	Veri tipi tip ile belirlenmiş küme elemanlarına erişir.
va_end(ap)	va_list tipinde bildirilmiş ap 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, ..., n) katsayıları esnek argüman olarak bildirilmiştir.

Program 9.1: Sonu -1 ile biten kümeyi ekrana yazar

```

01: /* 09prg01.c: Polinom hesabı */
02:
03: #include <stdarg.h>
04: #include <stdio.h>
05: #include <math.h>
06:
07: /* fonksiyon örneği */
08: double p(double, int, ...);
09:
10:
11: int main(void){
12:
13:     double x = 1.7;
14:
15:     printf("x = %lf için:\n",x);
16:
17:     printf("p(x, 1, 1.0) = %lf\n",
18:           p(x, 1, 1.0));
19:
20:     printf("p(x, 2, 1.0, -2.0) = %lf\n",
21:           p(x, 2, 1.0, -2.0));
22:
23:     printf("p(x, 3, 1.0, -2.0, 0.2) = %lf\n",
24:           p(x, 3, 1.0, -2.0, 0.2));
25:
26:     printf("p(x, 4, 1.0, -2.0, 0.2, 1.1) = %lf\n",
27:           p(x, 4, 1.0, -2.0, 0.2, 1.1));
28:
29:     printf("p(x, 5, 1.0, -2.0, 0.2, 1.1, -0.6) = %lf\n",
30:           p(x, 5, 1.0, -2.0, 0.2, 1.1, -0.6));
31:
32:     return 0;
33: }
34:
35: /* Verilen x, n ve ai katsayıları için,
36:    P(x,n) = a0 + a1*x + a2*x^2 + ... + an*x^n polinomu hesaplar.
37:    a0, a1, ..., an katsayıları esnek argüman olarak bildirilmiştir. */
38: double p(double x, int n, ...)
39: {
40:     double a, t = 0.0;
41:     int i;
42:
43:     /* argüman 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:         /* her bir argümanı sırasıyla al */
52:         a = va_arg(ag, double);
53:
54:         /* polinomun değerini hesapla */
55:         t += a*pow(x,i);
56:     }
57:
58:     /* belleği boşalt */
59:     va_end(ag);
60:
61:     return t;
62: }

```

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, 11,22));
13:     printf("topla(3, 11,22,33)        = %d\n", topla(3, 11,22,33));
14:     printf("topla(4, 11,22,33,44)     = %d\n", topla(4, 11,22,33,44));
15:     printf("topla(5, 11,22,33,44,55)  = %d\n", topla(5, 11,22,33,44,55));
16:     printf("topla(6, 11,22,33,44,55,66) = %d\n", topla(6, 11,22,33,44,66,66));
17:
18:     return 0;
19: }
20:
21: /* Esnek argümanla tanımlanmış n tane tamsayının sayının
22:    toplamını gönderir */
23: int topla(int n, ...)
24: {
25:     va_list ap;
26:     int i, top = 0;
27:
28:     va_start(ap, n);
29:
30:     for (i=1; i<=n; i++)
31:         top += va_arg(ap, int);
32:
33:     va_end(ap);
34:     return top;
35: }
36:

```

ÇIKTI

```

topla(2, 11, 22)           = 33
topla(3, 11, 22, 33)        = 66
topla(4, 11, 22, 33, 44)    = 110
topla(5, 11, 22, 33, 44, 55) = 165
topla(6, 11, 22, 33, 44, 55, 66) = 242

```

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 argümanların yazdırılması */
02:
03: #include <stdio.h>
04: #include <stdarg.h>
05:
06: /* herbiri int tipinde ve sonu -1 ile biten kümeyi ekrana yazar */
07: void argyaz(int arg, ...)
08: {
09:     va_list ap;
10:     int i;
11:

```

```
12: va_start(ap, arg);
13:
14: for (i = arg; i != -1; i = va_arg(ap, int))
15:     printf("%d ", i);
16:
17: va_end(ap);
18: putchar('\n');
19: }
20:
21: int main(void)
22: {
23:     argyaz(5, 2, 14, 84, 97, 15, 24, 48, -1);
24:     argyaz(84, 51, -1);
25:     argyaz(-1);
26:     argyaz(1, -1);
27:
28:     return 0;
29: }
```

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` deyimi 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` deyimi 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 */
    .
}
```

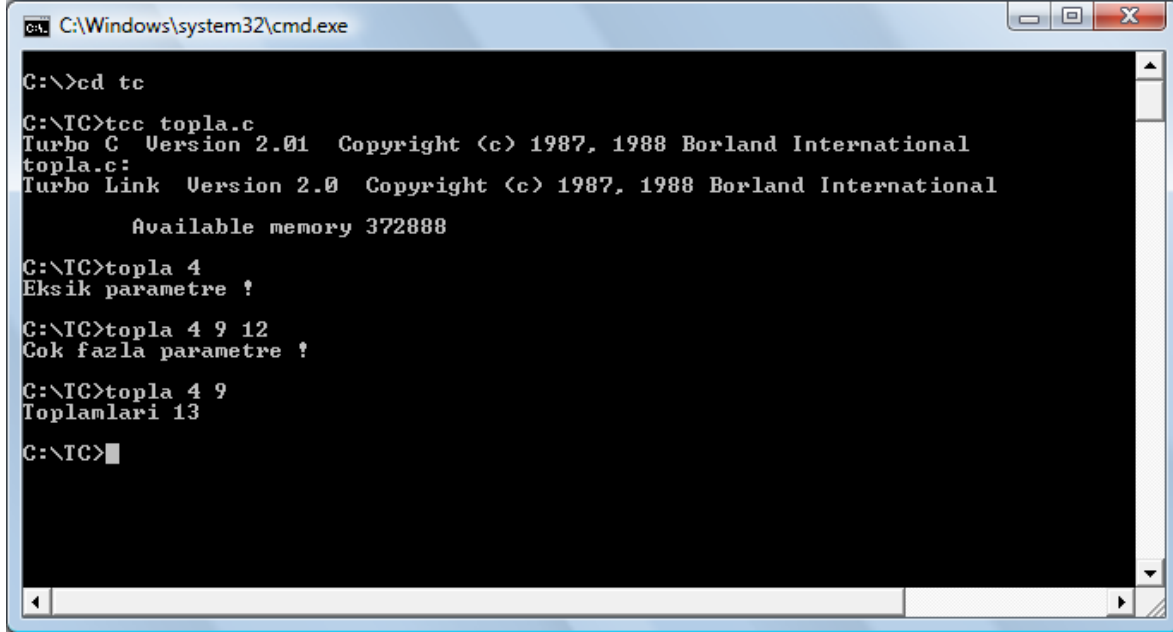
Program 9.1, komut satırından girilen iki sayının toplamını hesaplar.

Program 9.1: Komut satırından girilen iki sayının toplamını hesaplar

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

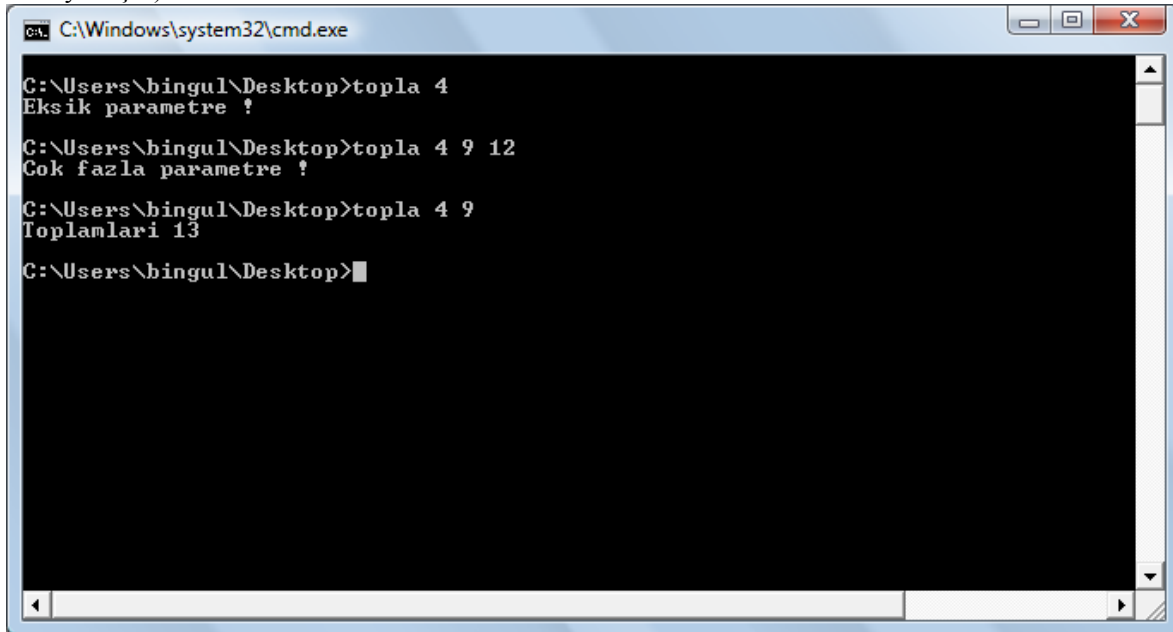
C:\TC>topla 4
Eksik parametre !

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 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
argv[0]	argv[1]	argv[2]

şeklinde.

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*

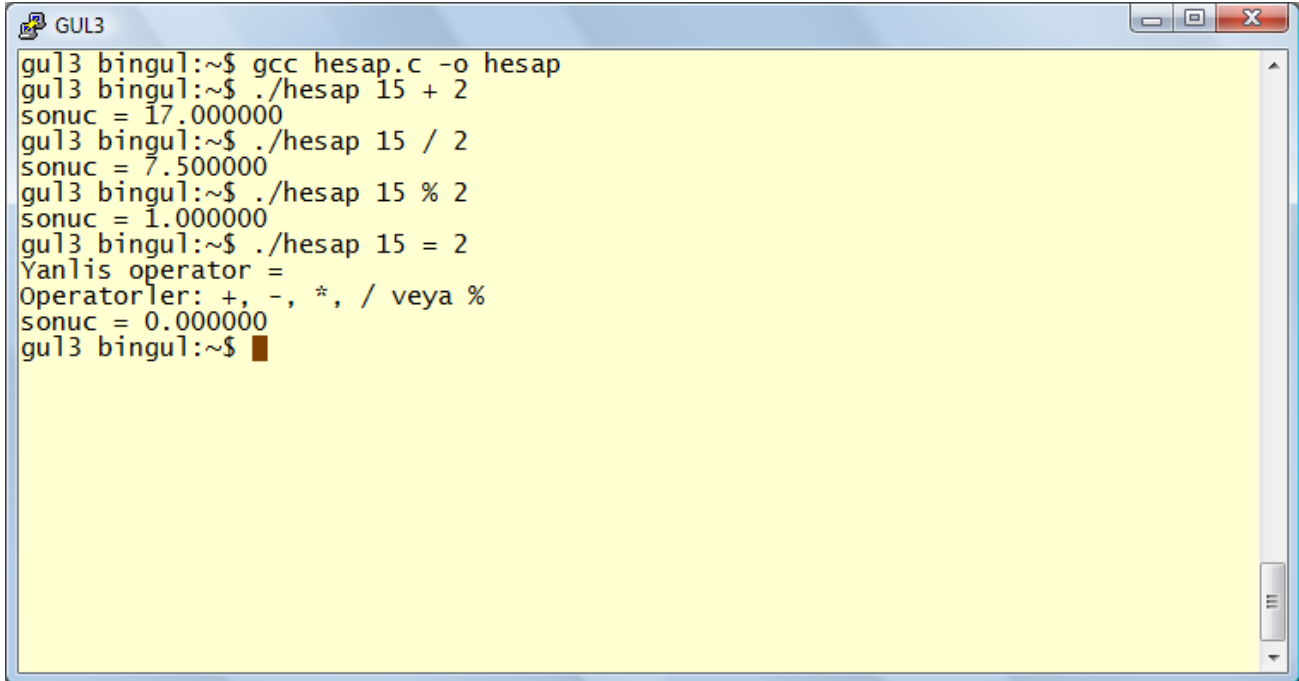
```

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

```

```
38:         printf("Yanlis operator %c\n",op);
39:         printf("Operatorler: +, -, *, / veya %%\n");
40:     }
41:
42:     printf("sonuc = %f\n",sonuc);
43:
44:     return 0;
45: }
```

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`
[cp komutunun kaynak kodları](#)
- `wc` (word count) komutu, bir dosyanın kaç karakter, kelime satırdan oluştuğunu bulup ekrana yazar.
Kullanımı: `wc dosya_adı`
[wc komutunun kaynak kodları](#)