1

Program Çözümleme

Bir bilgisayar programını çözümleme eylemini, onun her satırının (deyiminin) yaptığı işi açıklamak olarak anlayacağız. Bazı kaynaklarda bu eyleme *analiz* der. Daha ileri düzeyde *"trace"* eylemi, programın her deyiminin ana bellekte yaptığı işi çözümler. Profesyınel işlerde bu iş için kullanılan özel yazılımlar vardır. Ama biz programın her satırının ne iş yaptığını açıklamakla yetineceğiz.

ANSI, C dili için çok sayıda kütüphane fonksiyonu belirlemiştir. Giriş-çıkış işlemlerini yapan *printf()* ve *scanf()* onlardandır. Bu iki fonksiyon *stdio.h* başlık dosyası içindedir. Tabii, stdio.h dosyası C dilinin önemli kütüphane fonksiyonlarından birisidir ve standart kütüphenede her zaman kullanıma hazırdır.

Aşağıdaki örneklerde printf() ve scanf() fonksiyonlarının işlevleri gösterilmektedir.

Liste 1.1.

```
1 #include <stdio.h>

  int main()
  {
      printf("C Programlama dili");
6     return 0;
  }

  /**
  C Programlama dili
3 */
```

Çözümleme:

1. *Birinci satır: stdio.h* başlık fonksiyonunu çağırıyor. printf() fonksiyonunun çalışması için gereklidir.

   stdio.h, standart giriş-çıkış işlemlerini yapan fonksiyonları içeren başlık fonksiyonudur. C dilinin standart kütüphanesi içindedir. Onu çağırmak için, programın başına #include <stdio.h> önişlem buyruğu yazılır. *printf()* fonksiyonu stdio.h başlık fonksiyonu içindedir. Başlık fonksiyonunu çağırınca, onun içindeki her şey programa girer, sanki program içinde yazılmış gibi çalışırlar.

2. *Üçüncü satır:* main() fonksiyonunu başlatıyor. Bir C programının yapacağı her iş *main()* fonksiyonu tarafından belirlenir.

   main() fonksiyonunun gövdesi ya da bloku adını alan { } parantezi içine programın yapacağı işler sırasıyla yazılır. Onlar birer C deyimidir.

   Doğal olarak, main() içinde çağrılan bir fonksiyonu derleyici bulamazsa ya da kullanılan deyim C dilinin sözdizimine (syntax) uymuyorsa derleme işini yapamaz, hata verir.

3. *Beşinci satır:* printf() fonksiyonunu çağırıyor. Çağrılan fonksiyon () parantezi içindeki parametresini ekrana yazıyor.

4. *Altıncı satır:* Gövde içine yazılan son deyim daima

   ```
   return 0;
   ```

   olmalıdır. Bu deyimin özel bir işlevi vardır. Derleyici bu deyime kadar olan deyimleri çalıştırabilirse, bu deyime erişir. Bu demektir ki, main() gövdesindeki her deyim işlevini yaptı. Dolayısıyla program kendisinden istenen her işi başardı. Ama bu satıra gelmeden önceki deyimlerden birisi yanlışsa, derleyici o satırı işleyemediğini belirten bir hata uyarısı verir.

Liste 1.2.

```
#include<stdio.h>

int main()
{
    int n = 123;
    printf("Sayı = %d",n);
    return 0;
}
```

```
/**
Sayı = 123;
*/
```

Çözümleme:

1. *Birinci ve üçüncü satır:* Program 1.1 için açıklandı.

2. *Beşinci satır:* Değişken bildirimidir. *int* tipinden n adlı bir değişken tanımlıyor ve ona ilk değer olarak 5 sayısını atıyor. C dilinde her sabit, her değişken ve her fonksiyon kullanılmak üzere çağrılmadan önce tanımlanmalıdır. Değişken bildiriminin sözdizimi (syntax)

```
değişkenin_veri_tipi   değişken_adı [ = ilk_değer] ;
```

   biçimindedir. Değikenin veri tipi, ona atanacak verilerin tipini belirler. Böylece, ana bellekte ona yetecek kadar bir yer ayrılır.

   Veri tipinden sonra enaz bir boşluk bırakılarak değişkenin adı yazılır. Adlandırma kuralına uyulmak koşuluyla, değişkene istenilen bir ad verilebilir. Derleyici verilen adın anlamlı ya da anlamsız olduğunu anlamaz. Ancak, büyük programlarda her değişkene onun işlevini çağrıştıran bir ad verilmesi, kaynak programın algılanmasını kolaylaştırır.

   Teknik olarak değişkenin adı, ana bellekte kendisine ayrılan yerin adresini gösteren bir işaretçidir (pointer).

   Değişkene, bildirim anında değer vermek gerekli değildir. İsteğe bağlıdır (optional). İsteğe bağlı olanları [ ] parantezi içine yazacağız. Örneğimizde, n adlı değişkene ilk değer olarak 123 sayısı atanmıştır. Bildirim anında atanan değerlere *ilk_değer* diyoruz. İlk değer, istenildiğinde, aynı veri tipinden olmak koşuluyla, başka bir değerle değiştirilebilir. İlk değer atanmamışsa, kullanılmadan önce, değişkene bir değer atanmalıdır.

   Değişken bildirimini yapan deyimin sonuna noktalı virgül (;) konulmalıdır. Aslında her deyimin sonuna (;) konulur. Bu karakter derleyiciye deyimin sona erdiğini bildirir. Aksi halde, derleyici deyimin sona erdiğini anlayamaz.

3. *Altıncı satır:* Bu satırda printf() fonksiyonu çağrılıyor. Çağrılan fonksiyon parametresinde olan metni (string) ekrana yazıyor. Bunu yazarken " " içinde yer alan %d biçem belirteci (format specifier), n değişkeni için yer tutuyor ve onu onlu (decimal) sisteme göre yazıyor.

4. *Yedinci satır:* Program 1.1 için açıklandığı gibidir.

Program 1.1.

```
#include<stdio.h>

int main()
{
    int n;
    printf("Bir tamsayı giriniz\n");
    scanf("%d" ,&n);
    printf("Girdiğiniz sayı %d dir." ,n);
    return 0;
}
```

```
/**
Bir tamsayı giriniz
45
Girdiğiniz sayı 45 dir.
*/
```

Çözümleme:

1. *Birinci, üçüncü ve altıncı satırlar:* Önceki çözümlemelerde açıklandı.

2. *Yedinci satır:* scanf() fonksiyonu çağrılıyor. Bu fonksiyon, kullanıcının klavyeden gireceği sayıyı n değişkeni için anabelllekte ayrılan yere yazıyor. Girilen verinin tipini derleyiciye bildirmek için %d belirtecini kullanıyor. Girilen verinin yazılacağı adres $\&n$ ile belirleniyor. Buradaki d karakteri tamsayı (decimal) içindir (bkz. Bölüm **??**).

Program 1.2.

```
#include <stdio.h>

int main(){
    float a;
    printf("Bir kesirli sayı gir: ");
    scanf("%f",&a);
    printf("Girdiğiniz sayı  %f dir.\n" ,a);
    return 0;
}
```

```
/**
Bir kesirli sayı gir:  23.45
Girdiğiniz sayı 23.450001 dir.
*/
```

Kesirli sayıların (float) çoğunlukla yaklaşık değerleri yazılır. Örnekteki çıktı gcc derleyicisinden alınmadır. FArklı derleyiciler farklı yazabilir. Hepsinin yaklaşık değerler olduğunu bilmeliyiz.

Çözümleme:

1. *Bir, üç, beş ve sekizinci satırlar:* Önceki çözümlemelerde açıklandı.

2. *Dördüncü satır:* Veri tipi *float* olan *a* adlı değişkenin bildirimidir. *float* tip, kesirli sayılar içindir.

3. *Altıncı satır:* scanf() fonksiyonu çağrılıyor. Bu fonksiyon, kullanıcının klavyeden gireceği sayıyı *a* değişkeni için anabelllekte ayrılan yere yazıyor. Girilen verinin tipini derleyiciye bildirmek için %f belirtecini kullanıyor. Girilen verinin yazılacağı adres &*a* ile belirleniyor. Buradaki *f* karakteri float sayı tipleri içindir (bkz. Bölüm **??**)

   %f biçem belirtecidir. () parantezi içindeki parametreyi standart çıkış birimine (bizim kullandığımız sistemde ekran) gönderir. İlk printf() fonksiyonunun parametresi *"Bir kesirli sayı gir"* string'idir. Bu metin kullanıcıya ne yapacağını söyleyen bir işarettir (prompt). İkinci printf() fonksiyonunun parametresi *"Girdiğiniz sayı %f dir.\n"*,*a* sözdizimidir. " " içindeki ifade bir metindir. Bu metin içindeki %f belirteci çıktının sözkonusu metin içinde nereye konuşlanacağını belirtiyor ve sayının kesirli sayı olarak yazılmasının söylüyor. Bu nedenle, %f belirtecine bazen *yer tutucu* da denilir.

   Parametre olarak yazılan " " içindreki string'in sonuna konulan \n simgesine kaçış (escape) karakteri denilir. Bu karakter, imleci satırbaşına gönderir. Sonra gelecek çıktıları yeni satırın başından başlatır.

   %f yer tutucusunun metinde ayırdığı yere yazılacak veri, " " den sonra konulan virgülü izleyen *a* değişkeninin değeridir.

Program 1.3.

```
1 #include <stdio.h>

  int main(){
      char ch;
      printf("Bir harf giriniz : ");
6     scanf("%c" ,&ch);
      printf("%c  harfini girdiniz.\n" ,ch);
      return 0;
  }
```

```
1 /**
  Bir harf giriniz :  m
  m  harfini girdiniz.
  */
```

Çözümleme:

1. *Bir, üç, beş ve sekizinci satırlar:* Önceki çözümlemelerde açıklandı.

2. *Dördüncü satır:* char c; decimi, veri tipi karakter clan c adlı bir değişken bildirimidir.

3. *Altıncı satır:* scanf() fonksiyonunu çağırıyor. Bu fonksiyon kullanıcının klavyeden girdiği veriyi karakter olarak &c adresine yazıyor.

4. *Yedinci satır:* %c yer tutucusunun metinde ayırdığı yere, ch değişkeninin değerini yazıyor. Burada yer tutucunun kime yer ayırdığı önceki çözümlemelerde açıklandığı gibidir.

   Burada %c yer tutucusu, aynı zamanda bir dönüştürücüdür. String içinde %c dönüştürücüsü kullanılmazsa, C derleyicisi değişken değerini int olarak yazar. Çünkü, C dilinde *int* öntanımlı veri tipidir. Derleyiciye veri tipi bildirilmediğinde onu *int* sayar.

Program 1.4.

```
#include <stdio.h>

int main(){
    char ch;
    printf("Bir karakter giriniz : ");
    scanf("%c" ,&ch);
    printf("Girdiğiniz karakter %c dir.\n" ,ch);
    printf("Girdiğiniz karakterin ASCII kodu %d dir.\n" ,ch);
    return 0;
}
```

```
/**
Bir karakter giriniz : h
Girdiğiniz karakter h dir.
Girdiğiniz karakterin ASCII kodu 104 dir.
*/
```

Çözümleme:

1. *Sekizinci satır:* (Öteki satırların işlevleri önceki çözümlemelerde açıklandı.) Bu satır prinf() fonksiyonunu çağırıyor. Bu fonksiyon, parametresindeki %d yer tutusunun olduğu yere, ch değişkeninin değerini decimal (tamsayı) olarak yazıyor. Aslında *ch* bir karakterdir. Ama yer tutucu, aynı zamanda bir dönüştürücüdür. &ch adresindeki değeri %d cinsinden; yani tamsayı olarak yazıyor. Tabii, &ch adresinden gelen değer, kullanıcının girdiği karakterin ascii kodudur. Örneğimizde bu karakter *h* harfidir ve ascii kodu 104 dir.

Liste 1.3.

```
#include <stdio.h>

int main(){
    int n = 71;
    printf("%d ascii koduna karşılık gelen karakter %c dir.n" , n,n);
    return 0;
}
```

```
/**
  ASCII kodu %d olan olan karakter G dir.
*/
```

Çözümleme:

1. *Beşinci satır:* (Öteki satırların işlevleri önceki çözümlemelerde açıklandı.) Bu satır printf() fonksiyonunu çağırıyor. Parametre içinde iki yer tutucu var. Birincisi olan %d dönüştürücüsü, birinci değişkenin değerini decimale (tamsayı) dönüştürüp yazacak. İkinci %c yer tutucusu ise, ikinci değişkeni karakter cinsinden yazacak. String'den sonraki virgülü izleyen ilk değişken n, ikinci değişken gene n dir. Birinci değişkenin değeri, ilk yer tutucunun olduğu yere *decimal* olarak yazılıyor. İkinci değişkenin değeri, ikinci yer tutucunun olduğu yere *char* olarak yazılıyor. Her iki değişken aynı olduğu halde, birincide değeri tamsayı olarak, ikincide ise karakter olarak yazılıyor.

   Bu örnek, dönüştürücülerin işlevini anlatmaya yeterlidir.

Program 1.5.

```
#include<stdio.h>

int main(){
    printf("SATIR 0 : |123456789012345| \n");

    printf("SATIR 1 : |---------------| \n");

    printf("SATIR 2 : |%6d|\n" ,1234);

    printf("SATIR 3 : |%3d|\n" ,1234);

    printf("SATIR 4 : |%.2f|\n",1234.6382);

    printf("SATIR 5 : |%10.2f|\n",1234.6382);

    printf("SATIR 6 : |%.f|\n" ,1234.6382);

    printf("SATIR 7 : |%e|\n",1234.6382);

    return 0;
}
```

```
/**
SATIR 0 : |123456789012345|
SATIR 1 : |---------------|
SATIR 2 : |  1234|
SATIR 3 : |1234|
SATIR 4 : |1234.64|
SATIR 5 : |   1235.64|
SATIR 6 : |1235|
SATIR 7 : |1.234638e+003|
*/
```

Tamsayı ve kesirli sayıların çıktıları farklı biçemlerde yazılabilir. Ekrana ya da yazıcıya yazılanların birer metin (resim) olduğunu aklımızdan çıkarmamalıyız.

Kesirli sayıların (float) çoğunlukla yaklaşık değerleri yazılır. Ondalık hane sayısı belirtilerek kesirli sayı yazdırılırken, kullanılan skalada atılan ondalık kısım yarım ya da daha büyükse, atılmadan kalan kısım, sözkonusu skalada bir üst değere yükseltilir.

Çözümleme:

1. *SATIR 0 ve SATIR 1 :* Hane sayısını saymayı kolaylaştırmak için yazılmıştır.

2. *SATIR 2 :* Çıktıyı 6 haneye sağa yanaşık yazar. Solda 2 hane boş kalır : |_ _ 1 2 3 4|

3. *SATIR 3 :* Çıktıyı 3 haneye sağa yanaşık yaz komutuna uymaz; çünkü tamsayı verilen hanelere sığmaz. Sayının tam kısmı için gerektiği kadar hane ayırır : | 1 2 3 4|

4. *SATIR 4 :* Çıktıyı 2 ondalıklı olarak yaz komutuna uyar. Tam kısmı aynen yazar. Kesir kısmını 2 ondalık hane ayırır. Dört ondalık haneden son iki haneyi atar. Attığı kısmın ilk basamağı >= 5 olduğundan, kalan kısmın son basamağını 1 arttırır. Onlu ayracı (.) dahil sayının tamamının sığacağı 7 haneyi ayırır, oraya sayıyı sağa yanaşık yazar. Çıktı |1224.64| olur.

5. *SATIR 5 :* Çıktıyı 10 haneye 2 ondalıklı olarak sağa yanaşık yaz. Kesir kısmını önceki gibi ayarlar. Sayının çıktısına, onluk ayracı (.) dahil, toplam 10 hane ayırır. Ayrılan yere sayıyı sağa yanaşık yazar. Solda 3 hane boş kalır. |____ 1 2 3 4 . 6 4|

6. *SATIR 6 :* Çıktıyı 0 ondalıklı olarak sağa yanaşık yaz. Sayının ondalık kısmı tamamen atılıyor. Atılan kısmın ilk basamağı >= 5 olduğundan, kalan kısmın son basamağı 1 artar : | 1 2 3 5|

7. *SATIR 7 :* Çıktıyı üstel olarak yaz (e kullan). Çıktıya yetecek 13 haneyi ayırır : |9.876543e+002|

Bazen bir deyimle (aynı satırda) birden çok sayı girmek isteyebiliriz. Program 1.6 onu gösteriyor.

Program 1.6.

```c
#include <stdio.h>

int main(){
    int a,b;
    float c,d;
    printf("iki tamsayi giriniz: \n");
    scanf("%d%d" ,&a,&b);
    printf("Girdiginiz ilk sayi %d ve ikinci sayı %d dir.\n"  , a,b);

    printf("Bir tamsayı ile bir kesirli sayi giriniz : \n");
    scanf("%d%f",&a,&c);
    printf("Girdiginiz ilk sayi %d ve ikinci sayı %f dir.\n"  , c,d);

    return 0;
}
```

```
/**
   iki tamsayi giriniz:
   123
   4546
   Girdiginiz ilk sayi 123 ve ikinci sayi 4546 dir.

   Bir tamsayi ile bir kesirli sayi giriniz :
   123
   34.784
   Girdiginiz ilk sayi 536870912 ve ikinci sayi 0.000000 dir.
```

Çözümleme:

1. *Yedinci satır:* Aynı deyimle birden çok tamsayı okunabilir.

2. *Onbirinci satır:* Kuramsal olarak, aynı deyimle aynı anda bir `int` ile bir `float` okunabilir. Ancak bazı derleyiciler bunu yapmaz. Kullanıcının derleyicinin bu işi yapıp yapmadığını denetlemesi gerekir. Çoğunlukla *int* olmayan birden çok değişken aynı deyimle okunamaz. Örneğin, `gcc` derleyicisi üzerine kurulu Dev-C++ tümleşik yazılımı, çıktıda yazılan 10.satırı verir. Bu demektir ki `gcc` aynı deyimle bir *int* ile bir *float* sayıyı okuyamıyor.

   Değişken bildiriminde de aynı kurala uyarak, `int` olmayan değişkenlerin her birisini ayrı ayrı deyimlerle bildirmeliyiz. Çünkü,

   ```c
   float x, y ;
   ```

   gibi bir bildirim yapılınca, derleyici x değişkenini float olarak algılar, ama y değişkenini tipi belirtilmemiş olarak algılar ve ona öntanımlı (default) veri tipi olan `int` tipini verebilir.

   O nedenle, *int* olmayan birden çok değişken bildiriminde her birisi için ayrı bildirim deyimi yazılmalıdır. Değişken değerleri okunurken de aynı kurala uyularak, *int* olmayan birden çok değişkenin her birisi için ayrı bir `scanf()` fonksiyonu kullanılmalıdır.