2

C Programlarının Yapısı

Bir C programı C fonksiyonlarından oluşur. Bunlar arasında `main()` adı verilen ana fonksiyon daima olmalıdır. C fonksiyonları programı oluşturan yapıtaşlarıdır.

Bir C programının, bilgisayarda çalışabilmesi için şu aşamalardan geçmesi gerekir:

1. Kaynak program: Herhangi bir programlama dilinde text olarak yazılır.
2. Derleyici: Kaynak programın yazıldığı dile ve kullanılacak makinaya uyan bir derleyici gerekir. Derleyici, kaynak programı `obj` koduna çevirir.
3. Linker: Varsa öteki obj kodları birleştirerek makina diline çevrilmiş (yani çalışabilir) dosyayı yaratır.

Bu bölümde en basitten başlayarak C programlarının yapısını örneklerle açıklayacağız.

2.1 En Basit c programı

Program 2.1.

```
1 #include <stdio.h>

  main()
```

```
{

}
```

En basit C pogramı Program 2.1 ile verilendir. Bir C programı bundan daha basit duruma getirilemez.

Bu program hiç bir iş yapmaz. Ama bir C programında olması gerekenleri gösteriyor.

#include <stdio.h> : Her C programına standart giriş/çıkış işlemlerini yapan `stdio.h` dosyası çağrılır. Çağrı eylemini `#include` anahtar sözcüğü yapar ve daima ilk satırda olduğu gibi yazılır. Bu şekilde çağrılan dosyalara başlık dosyaları (header files) denilir. Çok sayıda başlık dosyası vardır. Birbirleriyle ilgili fonksiyonlar bir başlık altında toplanır. Başlık fonksiyonları C kütüpanesindedirler. Bir programa birden çok başlık dosyası çağrılabilir. Kullanıcı isterse başka başlık dosyaları da oluşturabilir.

Çağrılan başlık dosyasındaki bütün fonsiyonlar, sanki çağıran programda yazılmış gibi programa katılmış olurlar. Bunun örneklerini ilerleyen bölümlerde bolca göreceğiz.

main() : Her C programı `main()` fonksiyonunu içermelidir. Çünkü programın yapacağı bütün işleri `main()` fonksiyonu belirler. Bir programda ancak bir tane main() fonksiyonu olabilir. Ama bir C programının içine gerektiği kadar başka fonksiyonlar konulabilir. Bir program içinde başka fonksiyonların nasıl tanımlandığını ve onların main() tarafından nasıl çağrıldığını ilerleyen bölümlerde göreceğiz.

() : `main` adının sonuna konulan () parantezi zorunludur. Aslında her fonksiyonun adının sonuna () parantezi konulmak zorundadır. Derleyici bu parantezi görmezse onun fonksiyon olduğunu anlamaz. Parantezin içine ileride parametreler yazacağız. Şimdilik `main` fonksiyonunun sonuna () yazılması gerektiğini bilmek yetecektir.

() parantezinin başka bir işlevi de operatörlerde öncelik sırasını belirlemektir. Onun nasıl olduğunu operatörleri incelerken göreceğiz.

Gövde : { } parantezleri `main()` fonksiyonunun gövdesidir. Programın yapacağı bütün işler { } parantezi içine, sırayla, yazılır. Sonra program onları yazıldığı sırayla işletir.

Blok : { } parantezi bir blok oluşturur. Bir blok içindeki deyimler bir birlik oluşturur. Program yürüyüşü (kontrolü) bir bloka girince, aksi söylenmezse, bloktaki deyimleri sırasıyla işletir.

Program 2.1'de { } parantezi (gövde) içine bir şey yazılmadı. Dolayısıyla, program hiç bir iş yapmaz. Şimdi bu programı bir iş yapar duruma getirelim. Yaptıracağımız iş, ekrana merhaba C yazdırmak olacaktır:

Program 2.2.

```
#include <stdio.h>

  main()
4 {
   printf("Merhaba C\n");
  }
```

Bu programın öncekinden tek farkı 5.satırdaki `printf()` fonksiyonunun `main()` fonksiyonunun gövdesine girmiş olmasıdır.

`printf()` bir C fonksiyonudur; () parantezi içindekileri standart çıktıya gönderir. Bizim kullandığımız sistemde standart çıktı ekrandır. Ama standart çıktı olarak, yazıcı vb başka bir çıkış birimi de tanımlanabilir.

Bir fonksiyon adının sonuna, onun fonksiyon olduğunu derleyiciye bildirmek amacıyla zorunlu olarak konulan () parantezi içine, varsa fonksiyonun parametreleri yazılır. Parametre değerlerinin çıktıda nasıl biçemleneceği (format) ayrıca belirlenebilir.

Örneğimizde `printf()` fonkiyonunun parametresi *Merhaba C* metnidir. C dilinde karakter dizilerine *string* denir. Biz de *metin* yerine *string* terimini kullanacağız[1]

\n

"Merhaba C" stringinin sonuna yazılan \n simgesi bir kaçış karakteridir. Ekrana *Merhaba C* yazıldıktan sonra imlecin satırbaşı yapmasını sağlar. Böylece, programın ekrana yazacağı yeni string varsa, onu yeni satırın başından başlatır.

Kaçış Karakterleri

Önüne \simgesi konulan karaktere *kaçış karakteri* (escape character) denilir. Kaçış karakteri o karakterin işlevinin değişmesi demektir. Kaçış karakteri, asıl işlevinden farklı bir işlev üstlenirler. Çok sayıda kaçış karakteri vardır. Şimdilik sık sık karşılaşacağımız bazı kaçış karakterlerini bilmek yetecektir:

[1] Bir programlama dilinin anahtar sözcüklerini Türkçe'ye çevirmenin pedagojik olmadığını, öğrenmeyi zorlaştırdığını düşünüyoruz.

\n (newline, satırbaşı)

\t (tab)

\v (vertical tab, düşey tab)

\f (form feed, new page, yeni sayfa)

\' (single quote, tek kesme işareti)

\" (double quote, çift kesme işareti)

\ (backslash, ters bölü işareti)

\b (backspace, bir karakter geri git)

\r (carriage return, satırbaşına git)

\xdd (hexadecimal olarak ascii kodu ['A' için \x41])

\ddd (octal olarak ascii kodu ['A' için \101])

Deyimin sonu

Derleyiciye bir deyimin sona erdiğini bildirmek için, sonuna noktalı virgül (;) konulur.

Uyarı 2.1. *Bazı derleyiciler fonksiyon adının önüne* int *yazılmasını ister. Buradaki int* 0 *ya da başka bir tamsayı değer alır. Bu değerleri derleyici mantıksal değerler olarak algılar.* 0 *değeri, hiç bir hata olmadan fonksiyonun tamamlandığı anlamına gelir.* 0*'dan farklı değerler, fonksiyonda hata oluştuğu anlamına gelir.*

Sözdiziminde hata oluştuğunda, derleme işlemi yapılamaz. Başka bir deyişle derleme hatası *oluşur. İyi derleyiciler, hatanın hangi satırda olduğunu bildirir. Böylece, kaynak programdaki hata ayıklanabilir (debug).*

Bizim kullandığımız derleyici fonksiyonların önüne *int* konulmasını gerektirmiyor.

2.2 Kaynak Programın Biçemi

C dilinde deyimlerin bitişini (;) ile belirttiğimizi söylemiştik. Öyleyse, satırların derleyici açısından bir önemi olmaz. Dolayısıyla, bir satıra birden çok deyim yazılabilir. Ama her deyimin sonunai (;) konulur. Örneğin,

int a,b,c; float gelir, vergi;

deyimi

int a,b,c;

float gelir, vergi;

deyimlerine denktir. Algılamanın kolaylaşması için her deyimin ayrı bir satıra yazılması tercih edilir. Benzer olarak,farklı blokları biraz içerlek (indent) olarak düşey hizada yazmak, bloklar arasına boş satırlar koymak, kaynak programın kolay okunmasını ve algılanmasını sağlar. Ama kaynak programın yazılış biçemi (format) derleyiciyi etkilemez. Hangi biçemde yazılırsa yazılsın, sözdizimi doğru olan kaynak programı derleyici derler. Ne var ki, sözdizimi yanlışı olduğu zaman, hata ayıklama zorlaşır.

2.2.1 Fonksiyon Bildirimi

C fonksiyonlarının kullanılışını, ayrı bölümde, daha ayrıntılı olarak ele alacağız. Şimdilik, basit program yapısını açıklayabilmek için gerekseme dyduğumuz fonksiyon bildirimini ve fonksiyon çağrısını kısaca tanımlayacağız.

Bir işi yapmak için belirli bir sırayla işlenmek üzere bir araya getirilen deyimlerin oluşturduğu öğedir. Bir C programı en az bir fonksiyon içerir; o da main() fonksiyonudur. main() dışında istenildiği kadar fonksiyon içerebilir.

Programın yapacağı işleri yaptıran deyimlerin hepsini, sırasıyla main() fonksiyonunun gövdesine yazmak mümkündür. derleyici buna itiraz etmez. Kaynak programı okuyup değiştirmek, güncellemek isteyenler için böyle yapılması, 5-10 deyim içeren küçük programlarda bir sorun yaratmaz. Ama yüzlerce ve hatta binlerce deyim içeren programlarda, bütün deyimleri main() içine yazılınca, onları düezeltmek, güncellemek çok zorlaşır. Dolayısıyla, belirli işleri yapan deyimleri bir araya getirip main() gövdesi dışında bir yere bir ad vererek yazmak, sonra gerektiğinde onu main() içinde adıyla çağırmak daha uygundur. Bu biçimde yazılan öğelere *fonksiyon* denilir. Bazı kaynaklarad fonksiyon yerine *procedure*, *subroutine* gibi adlar verilir.

C dilinde iyi programlar, belirli bir işi yapan deyimleri bir araya getirip onu bir fonksiyon olarak tanımlar.

C dilinde iki tür fonksiyon vardır:

1. Kütüphane fonksiyonları
2. Kullanıcı taımlı fonksiyonlar

Çok sayıda kütüphane fonksiyonu vardır. Birbirleriyle ilişkili olanlar bir araya getirilip paketler haline getirilmiştir. Örneğin, giriş-çıkış işlemlerini

yapan fonksiyonlar stdio.h içindedir. Bu pakatlerde olan bir fonksiyonu kullanmak için, programın başına

```
#include <stdio.h>
```

yazmayı öğrendik. Kütüphaneye konulan bu paketlere *başlık* fonksiyonları diyoruz. Her başlık fonksiyonu *#include* önişlemcisi ile çağrılabilir. Çağrılan başlık fonksiyonunda olan her şey, çağıran programa girer, sanki o programda yeni yazılmış gibi işlev yaparlar.

Kullanıcı tanımlı fonsiyonlar, kütüphanede olmayan, ama programcının gerçekleştirmek için yazdığı deyimlerden oluşan bir birimdir. Fonksiyon bildiriminde belirli bir sözdizimi vardır. Ona uyulmalıdır.

Liste 2.1.

```
veri\_tipi  ad(parametreler )
            {
               fonksiyonun gövdesi
            }
```

Bir fonksiyon bildiriminde (tanımında) belirtilmesi zorunlu olan öğeler şunlardır.

Veri_tipi Her fonksiyonun verdiği bir değer ve o değerin bir veri tipi vardır. Ona verdiği (döndürdüğü, return) veri tipi diyoruz. Hiç değer vermeyen fonksiyonların veri tipi *void*'dir. Void boş demektir. Burada, *fonksiyon bir değer vermiyor* anlamındadır. Bazı dillerde bir iş yapan ama bir değer vermeyen fonksiyonlara *procedure* denilir. Fonksiyonun bize vereceği değer, fonksiyon gövdesi içine son deyim olarak yazılır. Sözdizimi,

```
return değer;
```

biçimindedir.

fonksiyonun adı Her fonksiyona bir ad verilir. Adlandırma kuralına uyularak istenilen bir ad verilebilir. Ama çok sayıda fonksiyon içeren programlarda fonksiyon adları yaptıkları işi çağrıştırı nitelkte olursa, kaynak program daha kolay algılanır. Örneğin, gelir vergisi hesabı yapan bir programa *gelir_vergisi_hesapla* gibi bir ad verilirse, fonksiyonun ne iş yptığı herkes tarafından anlaşılır. Ama aynı fonksiyona *hesap* gibi bir ad berilirse, neyin hesabını yaptığı adından anlaşılamaz.

Matematikte olduu gibi, C fonksiyonları da bazı değişken değerlerine göre değer verebilirler. Örneğin bir dikdörtgenin alanını bulan alan() fonksiyonu eni 3 birim, boyu 5 birim olan bir tek dikdörtgen için

Program 2.3.

```
1           int alan( )
                {
                    return 3*5;
                }
```

diye tanımlanabilir. Ama bunun yerine verilen her dikdörtgenin alanının hesaplayan

Program 2.4.

```
1           int alan(en,boy )
                {
                    return en*boy;
                }
```

diye tanımlanırsa, boyutları verilen her dikdörtgenin alanını hesaplayabilir. Bunun main() içine nasıl çağrılacağını biraz sonra göreceğiz. Program 2.4'den görüldüğü gibi, parametreler () içine yazılır. Birden çok parametre varsa, aralarına (,) konulur.

fonksiyonun gövdesi İkinci satırda başlayan { ile başlayan fonksiyon gövdesi, istenen işi yapabilmesi için gerekli olan bütün deyimleri içerir. Son deyim olarak vereceği değer return anahtar sözcüğü ile belirtilir ve fonksiyon gövdesi } ile kapatılır.