1

Veri Tipleri

Bazı diller veri tiplerini önceden ayırır, bazıları ayırmaz.

Teknik olarak, veri tipini önceden belirleyen diller, anabelleğe yerleşecek verileri, orada kapsayacakları alanlara göre gruplara ayırır. Bu grupları pratikteki kullanımlarına göre adlandırır Örneğin, karakterler, tamsayılar, kesirli sayılar vb gibi yalın tipler yanında dizim (array) vb gibi birleşik yapılar bu ayrımı yapan her dilde vardır.

Fortran, Pascal,C/C++, Java vb diller veri tiplerini önceden ayıran dillerdir. *Python, Ruby* gibi diller bu ayrımı önceden yapmaz; önce veriyi ana belleğe yerleştirir sonra onun tipini belirler.

Bu iki yöntemin iyi ve kötü sayılabilecek yanları vardır. Verileri önceden tiplerine ayırmanın avantajları şöyle sıralanabilir:

- Her veri tipine yetecek ve ancak o kadar anabellek alanı ayrılır.
- Programcı, ana bellekte ayrılacak adresin büyüklüğünü bilerek değişkenlerini tanımlar.
- Her değişkene, anabellekte bir yer ayrılır. Değişken etkin olduğu sürece, o adrese bir başka veri yazılamaz. Programcı her istediğinde değişkenin değerini değiştirebilir, yeni değer atayabilir. Bu işlem değişkenin adresini değiştirmez.
- Her veri tipi üzerinde yapılacak işlemler farklıdır. Örneğin sayılarda yapılan işlemlerle metinlerde yapılan işlemler farklıdır. Bu fark göetilerek veri tipleri üzerinde işlem tanımlanır. Bu olgu programcıya kolaylık sağlar.

Buna karşın, Verileri önceden tiplerine ayırmayan dillerin avantajları şöyle sıralanabilir:

- Programcı veri tipleriyle uğraşmaz; gerekli verileri girer ve kullanır.
- Aynı adlı değişkene her istediğinde aynı ya da başka tipten veri atayabilir.
- Bir veri anabelleğe bir kez konulur. O değeri kullanan değişkenler o değeri ortak kullanır. Böylece anabelleğe bir veri ancak bir kez girer. Bu olgu, anabelleğin çok ekonomik kullanılmasını sağlar.

C dili verileri önceden tiplere ayıran dillerdendir.

1.1 Temel Veri Tiplri

C diline üç tane temel veri tipi vardır:

1. sayı
2. karakter
3. string

Bunlar uygulamada alt gruplarına ayrılırlar. Alt grupların oluşturulmasında ortak bir standarttan sözedilemez. Farklı derleyiciler farklı sınıflandırmalar ve farklı adlar kullanmıştır. Onları alışılmış adlarına ve bellekte kapladıkları büyüklüklere göre sınıflandırmak yeterli olacaktır.

1.2 İlkel Veri Tipleri

Tamsayılar	int
Karakterler	char
Mantıksal	false== 0, true != 0
Kesirli Sayılar	float, double
Yapısal Tipler	array, string, functions, struct, files, pointers

Tablo 1.1: İlkel Veri Tipler

İlkel veri tipleri deyince, bir dilde çok kullanıldığı için dile gömülü gelen, dolayısıyla kütüphneden çağrılması gerekmeyen veri tipleri akla gelir.

Ama C dilinde bazı ilkel tipleri standart kütüphaneden çağırmak gerekebilir. İlkel veri tipleri için bir standart yoktur. Ama her dilde karakterler, tamsayılar (integer), kesirli sayılar (float) yalın tiplerdir. Dizim (array), string, fonksiyon, dosya, işaretçi vb birleşik (yapısal) tiplerdir. Bazı diller metin (string) tipleri de ilkel veri olarak alır. C dili *string* tipini karakter dizimi olarak tanımlar.

1.2.1 Sayılar

Sayılar, *tamsayılar* ve *kesirli sayılar* diye ikiye ayrılır. Onlar da kendi içlerinde alt gruplar ayrılır. Bu gruplandırma sayıya anabellekte ayrılacak yerin büyüklüğüne göre değişir. Alt grupların adları ve sayıları için bir standart yoktur. Farklı diller farklı gruplamalar yapar.

C dilinde tamsayıların ve kesirli sayıların alt grupları Tablo 1.2 ve Tablo 1.3 ile verilmiştir.

1.3 C Dilinde Tamsayı Tipleri

Tablo 1.2'de tamsayılar gruplanırken, aynı bellek büyüklüklerine karşılık gelen farklı adlar olduğunu göreceksiniz. Bu durum, çok sayıda C derleyicisi olmasına ve donanım niteliklerine bağlı olarak oluşmuştur. Örneğin, bazı kişisel bilgisayarlarda derleyiciler `int` veri tipine 2 byte ayırırken, bazı sistemler 4 byte ayırır.

Tamsayılar için `int` anahtar sözcüğü kullanılır. Bunun iki alt grubu `long` ve `short` diye adlandırılır. Bazı derleyiciler *long int* ve *short int* terimlerini kullanır.

Derleyiciye ve donanıma bağlı olarak değişmekle birlikte, *short* tamsayılar için enaz 16 bit, *long* tamsayılar için enaz 32 bit bellek alanı ayrılır.

`int` tip, bazı sistemlerde 16 bit, bazı sistemlerde 32 bit olabilir. Ama hiç bir derleyici `short` tipe `int` tipe ayırdığından fazlasını ayırmaz.

Aksi söylenmedikçe, bütün tamsayı tipleri işaretli (signed) sayılır; yani hem negatif hem pozitif değerler alırlar.

`char` tipler, derleyiciye bağlı olarak, işaretli ve işaretsiz olabilirler. O nedenle *signed char* ve *unsigned char* diye belirtmek uygun olur.

Bildiri Örnekleri:

long int tc_kimlik_no;

short int koltuk_no;

Tablo 1.2: C Dilinde int Veri Tipleri

Tip adı	Tip	Kapsadığı Yer	Değer aralığı
char	Karakterler	1 byte	-128' den 127' ye kadar
unsigned char	char	1 byte	0' dan 255'e kadar
signed char	char	1 byte	-128' den 128'e kadar
short int	Küçük tamsayılar	2 byte	-32 768 den 32 767'ye kadar
long int	Küçük tamsayılar	4 byte	-2 147 483 648'den 12 147 483 647'ye kadar
short	Küçük tamsayılar	2 byte	-32 768 den 32 767 ye kadar
long	Büyük tamsayılar	4 byte	-2 147 483 648 den 2 147 483 647 ye kadar
unsigned char	işaretsiz karakter-ler	1 byte	0 dan 255 e kadar
unsigned int	işaretsiz tamsayılar	2 byte	0 dan 65535 e kadar
unsigned short	işaretsiz küçük tamsayılar	2 byte	0 dan 65535 e kadar
unsigned long	işaretsiz büyük tamsayılar	4 byte	0 dan 4 294 967 295 e kadar
enum	numaralanmış sayılar	2 byte	0 dan 65535 e kadar

```
signed char ch;
unsigned char harf;
unsigned int yaş;
long uzunluk;
short m, n;
unsigned uçuş_no;
unsigned long posta_kodu;
```

Bu arada, sayıların alt gruplara ayrılışının standart olmadığını bilmeliyiz. Örnekse, Java'da gruplandırma ve grup adları kesindir, donanıma ya da derleyiciye göre değişmez. Tamsayılar 8, 16, 32 ve 64 bitlik oluşlarına göre dört alt gruba ayrılır. Python'da programcı sayıları alt gruplara ayırmakla uğraşmaz. O sayıyı girer, gerisini Python derleyicisi yapar.

Aslında, C dilinde çok standart olmayan sınıflandırma işiyle uğraşmaktnsa, her veri tipini kullanarak öğrenmek daha uygun olacaktır.

1.4 C Dilinde Kesirli Sayı Tipleri

C dilinde kesirli sayılar üç alt gruba ayrılır:

1. float
2. doube
3. long double

Kesirli sayılar, yüzer-gezer nokta (floating point) gösterimi dediğimiz yöntemle farklı biçemlerde (format) yazılabilir. Örneğin,

$$123.4567 = 1.234567 \times 10^{2} = 12345.67 \times 10^{-2}$$

sayıları aynıdır. Kesir ayracını istediğimiz yere kaydırabiliriz. Matematikte, kesirli olsun olmasın her sayı bir gerçel sayıdır (*real number*). Öte yandan, bilgisayardaki sayıların büyüklükleri sonludur. Oysa derçel sayılar (*real number*) sınırsız büyük olabilir. Dolayısıyla kesirli sayılara *real number* demek doğru bir adlandırma olmaz.

Buradan programcıların çıkaracağı sonuç şu olmalıdır. Bilimin her dalında olduğu gibi, bir bulguya, bir varlığa bir nesneye ad verilirken, çoğunlukla, o adın öteki dallardaki anlamı gözetilmez ve etimolojisi araştırılmaz. Bilgisayar dillerinde de bu olgu vardır. Bir bilgisayar dilini öğrenirken, o dilin sözdizimindeki terimlerin, o dildeki anlamlarını bilmek yetecektir.

Tablo 1.3: C Dilinde float Veri Tipleri

Tip adı	Tip	Kapsadığı Yer	Değer aralığı
float	Küçük kesirli sayılar (real)	4 byte	3.4E +/-38 (7 haneli)
double	Büyük kesirli sayılar (real)	8 byte	3.4E +/- 308 (15 haneli)
long double	Çok Büyük kesirli sayılar (real)	10 byte	1.7E +/- 4932 (19 haneli)

1.5 Adlandırma

Genellersek, C dilinde sabit, değişken, fonksiyon, array, struct, dosya, pointer gibi programın her öğesine bir ad verilir. Buna kaynak programın kendisi de dahildir. Kaynak programa nasıl ad verildiğini biliyoruz. Şimdi kaynak program içindeki öteki öğelere nasıl ad verildiğini görelim.

Bildirim

C dilinde, programda kullanılacak bütün sabitlerin, değişkenlerin ve fonksiyonların, kullanılmadan önce bildirilmeleri (declare) gerekir. Her sabitin, değişkenin ve fonksiyonun bir adı (identifier) vardır.

1.5.1 Adlandırma Kuralları

Adlandırma kuralları basittir. Adlar;

- harf, sayak (digit) ve alt-çizgi (_) karakterlerinden oluşur. Anlamlı ya da anlamsız oluşu önemli değildir. Örnekse, `ght8_se` bir ad olur.
- sayak ile başlayamaz. Karakter ya da _ ile başlayabilir. Sonra sayak içerebilir. Örneğin 8ay ad olamaz, ama ay8, _8ay ad olabilir.
- bir ad'da kullanılan karakter sayısı en çok 32 olmalıdır. Daha çok karakter kullanılabilir, ama ilk 32 'şer karakteri aynı olan iki adı derleyici birbirinden ayıramaz.
- C dili küçük/büyük harfe duyarlıdır. Dolayısıyla adlandırmada da bu kural geçerlidir. Örneğin, *Kent* ve *kent* farklı adlardır.

- C dili yazıldığında *unicode* yoktu. İngiliz alfabesi dışındaki alfabelerin de kullanılacağı düşünülmüyordu ve karakter kodlama sistemleri henüz ortaya çıkmamıştı. Dolayısıyla, C dili 128 karakter içeren ascii karakterleri için yazıldı. Sonradan bazı derleyiclere farklı kodlama sistemlerini yapan modüller eklendi. Ancak, kaynak programın taşınabilirliğini (yani her C derleyicisinde de derlenebilirliğini) sağlamak için, adlar ascii karakterleriyle yazılmalıdır[1].

Geçerli ve geçersiz ad örnekleri

```
212                /* geçersiz */
215u               /* geçersiz */
_ucret             /* geçerli */
gelirTemmuz2015    /* geçerli */
t_2a7b_            /* geçerli*/
```

1.5.2 Değişken Bildirimi

Teknik açıdan, değişken bidirimi (declaration), bir veriye anabellekte bir yer ayırma eylemidir. Ayrılan yere bir veri yazılsın ya da yazılmasın, değişken etkin kaldığı sürece, ayrılan anabellek adresine başka bir şey yazılmaz. Benzetmek gerekirse, o yer değişkene geçici olarak kiralanmış olur.

C de değişken bildiriminde değişkenin v*eri_tipi* ve *adı* (identifier) belirtilir. Veri tipi belirtilince, anabellekte ona ne kadar yer ayrılacağı belirlenmiş olur. Örneğin *float* tipinden değişkenlere 4 byte'lık yer ayrılır. Her değişkene bir ad verilir. Adlar, genel adlandırma kuralına uymalıdır (bkz. 1.5.1). Örneğin,

```
float  toplam;
```

bir değişken bildirimidir. Bu bildirim değişkene atanabilecek veri tipinin *float* olduğunu ve değişken adının *toplam* olduğunu bildiriyor. Bunu genelleştirirsek,

```
veri_tipi  değişken_adı;
```

bir değişken bildiriminin genel sözdizimidir (syntax).

Anımsarsanız, değişken adı harf, sayak (digit) ve _ karakterlerinden oluşur, ama sayak ile başlayamaz. C dili 32 karaktere kadar olan adları birbirlerinden ayırabilir. Ama ilk 32 karakterleri aynı ise, daha çok karakterden oluşan iki adı birbirinden ayıramaz. Zaten, uygulamada adlar için 32 karakter fazlasıyla yeterli olur. Unutmayınız ki, bildirimde uzun adlar

[1] Kodlama sistemleri geliştikten sonra yazılan dillerde bu sorun kolay aşılmıştır. Java, C#, Python, Ruby dillerinde dünyadaki bütün alfabeler kullanılabilir.

kullandığınızda, onları kullanırken de o uzun adları aynen yazmak zorundasınız. Gerekmediği halde çok uzun adlar vermek hem zaman alır hem de hata yapma olasığını artırır.

Bir değişken bildirimi yapıldığında, anabellekte onun veri tipine yetecek kadar bir yer ayrılır. Değişken adı, aslında bellekte o adresi işaret eden bir işaretçidir (pointer).

Program 1.1.

```
#include <stdio.h>

  main()
  {
5   int katsayı;
    float toplam;
    char kent[];
  }
```

Program 1.1, sırasıyla *int, float ve string* tipinden *katsayı, toplam ve kent* adları verilen üç değişken bildirimi yapıyor. Sonuncu `kent[]` veri tipi karakterlerden oluşan bir dizidir; yani bir string'dir. Array operatörü denilen [ ] parantezinin işlevini daha sonra açıklayacağız.

Aynı veri tipinden birden çok değişken bildirimi, değişkenlerin arasına (,) konularak tek bir deyim hainde de yaplabilir. Örneğin,

```
int a,b,c;
```

bildirimi *int* tipinden a,b,c adlı üç değişkeni bildiriyor. Ancak, *int* tipinden olmayan değişkenlerin bu şekilde tek deyimle bildirimi bazen sorun yaratabilir. Örneğin

```
float x,y,z;
```

bildiriminde, bazı dereleyiciler x değişkenini *float*, y ve z değişkenlerini *int* olarak algılar. Çünkü, C dilinde *int* öntanımlı (default) veri tipidir. Veri tipi apaçık belirli değilse, C derleyicisi onu *int* tipinden sayar.

1.5.3 İlk Değer Atama

Bazen değişken bildiriminde, ona ilk değerini atamak gerekebilir. Bunun için atama operatörü (=) kullanılır:

int sokak_no= 17;

deyimi, *int* tipinden s*okak_no* adlı bir değişken bildiriyor ve ona 17 ilk değerini atıyor. İlk değer atama zorunluğu yoktur. Program çalışırken değer atanabilir ya da atanmış değer değiştirilebilir. Böyle olması değişkenin

özniteliğidir. Ama değişkene hiç değer atanmadan kullanıldığında, beklenmedik hatalar doğabilir[2]. Her durumda, blok içindeki değişkenlere (yerel değişken), kullanılmadan önce değer atanmalıdır

1.6 Sabitler

Program boyunca değeri değişmeyecek değişkene sabit denilir. Bunlara *literal* denilir. Her veri tipinden *sabit* tanımlanabilir. Sabitler değişkenler gibidir; aralarındaki tek fark, sabit değerlerin program boyunca değişememesidir.

Sabit sayılar

Her sayı bir sabittir. Örneğin 7, 3.4, 17.3e9, 123.4567 sayıları birer sbittir.

Sabit sayıları *decimal, octal, hexadecimal* ya da *büyük sabit* (long constant) sayı olarak belirleyebiliriz.

- `Decimal` (on tabanlı) sayılar olduğu gibi yazılır. Solda sıfır olmaz; yani decimal sayı 0 ile başlamaz.
- `Octal` (8 tabanlı) sabitlerin önüne sıfır (0) konulur; yani 0 ile başlar. Octal sistemde sayaklar 0,1,2,3,4,5,6 dır. Örneğin 063 bir octal sayısıdır , ama 073 bir octal sayı değildir. Çünkü 7 sayağı *octal* sistemde yoktur.
- `Hexadecimal` (16 tabanlı) sabitlerin önüne 0x ya da 0X konulur. Hexadecimal sistemde sayaklar 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F dir. Burada A=10, B=11, C=12, D=13, E=14, F=15 sayılarını temsil ederler. Örneğin 0x1B3 bir hexadecimal sayıdır, ama 0x1G3 bir hexadecimal sayı değildir. Çünkü G, hexadecimal sistemde bir sayak değildir. *Hexadecimal* terimi yerine bazen, kısaca `hex` denilir. Hex sayı 0X ile başlarsa A,B,C,D,E,F sayakları büyük harfle yazılır ama 0x ile başlarsa küçük harfle yazılır.
- Büyük tamsayıların sonuna L konulur: 372L

Bazı tamsayı literalleri:

[2]Bazı diller değer atanmadığında değişkene bir öndeğer (default value) atar. Ama C derleyicilerinin hepsi bunu yapmayabilir. Değişkene değer atanmadan kullanılırsa, derleyici değişkenin adresinde önceden kalmış veriyi kullanabilir ve büyük hatalar oluşabilir.

```
532         /* decimal */
0123        /* octal */
0xc6        /* hexadecimal */
43          /* int */
12u         /* unsigned int */
123l        /* long */
123ul       /* unsigned long */
```

Sabit karakterler

Her karakter bir sabittir. C dilinde karakterler 'a' ve 'A' gibi tek tırnak arasında yazılır.

Kaçış karakteri

Programlama dillerinin hemen hepsinde '\' karakterine kaçış (escape) karakteri denilir. Bir karakterin önüne kaçış karakteri gelirse, onun anlamı değişir; genellikle ikisi birlikte yeni bir komut oluşturur. Kaçış karakterleri ya da kontrol karakterleri diye adlandırılan bu çiftlerin sayısı çoktur. Şimdilik çok karşılaşacağımız bir kaçını bilmek yeterlidir.

\n satırbaşı

\t tab

\\ters bölü simgesi

\' single quote

\0 null (Bir stringin sonuna, bittiğini göstermek için konulur.)

1.6.1 Sabit Bildirimi

Program boyunca değeri değişmeyecek değişkene sabit denilir. C 'nin önceki sürümlerind sabit bildirimi yoktu. Değeri değiştirilmeyen değişken sabit olurdu. Sabit bildirimi ilk kez ANSI C ile gelmiştir. Sabit bildirimi de değişken bildirimine benzer. Değişken adının önüne *const* yazılır.

int const a= 10;

const int b= 10;

const anahtar sözcüğü İngilizcedeki *constant* (sabit) teriminin kısaltmasıdır. Başka değer almayacağına göre, sabitin ilk değeri bildirim anında verilmelidir.

1.6.2 Önişlemci Sabitleri

C dilinde sabitler *önişlemci* olarak etkin kullanılır. Değeri program boyunca değişmeyecek olan değişkenlerin sabit olarak bildirimi, programın ileride değiştirilmesini ve güncellelenmesini kolaylaştırır.

```
#define TRUE 1
#define FALSE 0
#define KATSAYI 20
```

Bu bildirimden sonra TRUE, FALSE ve KATSAYI sabit olur.

Uyarı 1.1. *Sabitleri büyük harfle yazmak zorunlu değildir; ama böyle yapılırsa, kaynak programı okuyan herkes onların sabit olduğunu kolayca algılar.*

Sabitin fonksiyon parametresi olarak kullanılması da uygulamada sık geçer. Bir fonksiyonun parametresi sabit olunca fonksiyon o değeri değiştiremez.

1.7 enum

enum anahtar sözcüğü İngilizcedeki *enumerate* sözcüğünün kısaltmasıdır. Bir grup sabiti sıralayıp sıra numarası verme eylemini yapar. Örneğin,

```
enum meyveler {ELMA, ARMUT, KAYISI, NAR }
```

deyimi { } içindeki dört meyveye sıra numarası verir. Özel sıra numarası verilmemişse ilk sabite 0 numarası verilir. Sonrakiler, yazılış sırasına göre 1 er artarak numara alır. Buna göre ELMA=0, ARMUT=1, KAYISI=2, NAR=3 olur. Sıralanan nesneleri büyük harflerle yazmak zorunlu değildir, ama sabitler için söylediğimiz gibi, iyi bir alışkanlıktır.

Bildirimde, istersek sıra numaralarını kendimiz belirleyebiliriz:

```
enum meyveler {ELMA=5, ARMUT, KAYISI=4, NAR }
```

dersek ELMA=5, ARMUT=6, KAYISI=4, NAR=5 olur. Kural gereği, bu sıralamada ELMA ve NAR aynı numarayı alır. { } içindeki sabitleri belirten *meyveler* adı yerine, adlandırma kuralına uyrak, istediğiniz başka bir ad verebilirsiniz.

Sabitleri istediniz sırada numaralayabileceğinizi görmek için Program 1.2'e benzer programlar yazıp deneyiniz.

Program 1.2.

```
#include <stdio.h>

int main()
{
  enum meyveler {ELMA=5, ARMUT , KAYISI=2, NAR};

  printf("ELMA = %d\n", ELMA);
  printf("ARMUT = %d\n", ARMUT);
  printf("KAYISI = %d\n", KAYISI);
  printf("NAR = %d\n", NAR);
  return 0;
}
```

```
ELMA = 5
ARMUT = 6
KAYISI = 2
NAR = 3
```

1.8 Mantıksal Tipler

Mantıksal değerler, ikili mantık sisteminde yanlış (false) ve doğru (true) değerleridir. Bu değerlere, mantık konusunu matematiksel yapı olarak tanımlayan İngiliz matematkçi Geoge Boole'ün anısına *boolean* değerler denilir. Bilgisayarın mantıksal deyimleri işleyip karar verebilmesi için boolean değerler her dilde önem taşır. Bazı derleyicilerde *false* ve *true* ilkel veri tipi olarak vardır. C89 da boolean değerler yoktur. Onun yerine 0 değeri *false*, sıfırdan farklı değerler *true* kabul edilir:

1. false== 0
2. true != 0

Programcı isterse yanlış ve doğru boolean değerleri önişlemci olarak tanımlayabilir (bkz. **??**).

1.9 Öznitelikler

global/yerel

C dilinde kullanılan değişkenler, ana belleğe konuşlanmaları ve bellekte kalma süreleri açısından ikiye ayrılabilir. Hiç bir blok içinde olmayan değişkenlere programın *global* değişkenleri, bir blok içinde olanlara da o blokun *yerel* değişkenleri denilir.

Yaşam Süresi

Büyük programlarda zamanı gelen öğe ana belleğe konulur. İşi biten öğe ana bellekten silinir. Bir öğenin yaşam süresi (duration), onun ana bellekte kalış süresidir. Bir programda bazı öğelerin yaşam süresi kısadır. Bazı öğeler tekrar tekrar yaratılıp bellekten silinirler. Bazı öğeler de program boyunca bellekte kalırlar.

Kapsanma Alanı

Bir öğeye erişilebilmesi demek, o öğenin bir deyim ile kullanılabilmesi demektir. Örneğin bir değişkene erişmek demek, ona değer atayabilmek ve atanan değeri adresinden alıp kullanabilmek demektir. Fonksiyona erişmek ise, onu çağırıp çalıştırmak demektir. Program içindeki öğelere programın bazı yerlerinden erişilebilir, bazı yerlerinden erişilemez. Bir öğeye erişilebildiği yerlerin tamamına onun kapsanma alanı (scope) denilir. Bazı öğelere programın her yerinden erişilebilir. Bazılarına ancak programın belirli yerlerinden erişilebilir.

Genel kural olarak, bir blok içinden blok dışındaki öğelere erişilebilir. Ama blok dışından, blok içindeki öğelere erişilemez. Benzetmek için bir örnek verebiliriz. Bazı pencereler içeriyi göstermeyen ince bir tabaka ile kaplanır. İçeriden dışarısı görülebilir. Ama dışarıdan içerisi görülemez. Böyle olan otomobil pencerelerine çok rastlarız. Programlama dillerinde de benzer olgu olur. Bloklar, pencerelere kaplı ince tabakaların rolüne benzer bir rol oynarlar. Blok içindeki deyimler dışarıdaki öğelere erişebilir, ama dışarıdaki deyimler blok içindeki öğeleri kullanamaz.

Bağlanabilme

Birden çok kaynak kullananıldığında, bir öğe yalnızca o andaki kaynak tarafından biliniyor olabileceği gibi, onu öteki kaynaklar da biliyor olabilir. Bu konu ileride daha ayrıntılı incelenecektir.

1.10 Depolama Türleri

C dilinde bildirimi yapılan öğeler, yaşam süreleri, kapsanma alanları (scope) ve bağlanma (linkage) yetilerine göre dört sınıfa ayrılır:

1. auto

2. register
3. extern
4. static

Yaşam süreleri açısından, değişkenler için bu dört depo sınıfı aslında iki sınıfa iner:

geçici depolama: auto (automatic) ve *register* depo sınıflarına konulan değişkenler, ait oldukları blokun işi bitince bellekten silinirler. auto depo sınıfına konulacak değişkenlerin bildirimi

```
1|    auto float x, y;
```

gibidir. Bir blok içinde bildirimi yapılan değişkenler *auto* depo sınıfındandır. Onları ayrıca auto diye nitelemeye gerek olmaz. O nedenle, bir blok içinde yukarıdaki bildirimi

```
|    float x, y;
```

gibi de yazabiliriz.

Register'ler (yazmaç), MİB (CPU) içinde veri tutmaya yarayan küçük depolardır. Makine diline dönüşen kodlar çalışırken, çok kullanılan veriler bir registere alınır; böylece işlemlerin daha hızlı yapılması sağlanır. Register'e konulması istenen değişkenlerin bildirimi

```
|    register int m, n;
```

biçiminde yapılır. Ne var ki, boş register yoksa, derleyici bu isteğe uyamaz. Aslında, gelişkin derleyiciler çok kullanılan verileri kendiliğinden register'e koyabilme yetisine sahiptir. O tür derleyicilerde, programcının register'e konulacak değişkeni belirtmesine gerek kalmaz.

Geçici depolama, ana belleğin ekonomik kullanımı açısından önemlidir. Büyük programlarda çok sayıda değişken ve blok bulunur. Bazısı çok kısa süre için kullanılır, bazısı program boyunca kullanılır. İşi biten değişkenlerin ve blokların ana bellekten silinmeleri, belleğin o alanlarının başka işler için kullanılabilmesine olanak sağlar. O nedenle, iyi bir program, işi biten değişkenleri bellekten yokedilecek biçimde tanımlar.

kalıcı depolama: *extern* ve *static* depo sınıflarına konulan öğeler, program süresince silinmeden bellekte kalırlar. Global değişkenler ve fonksiyonlar öntanımlı (default) olarak *extern* öğe sayılırlar. Onlar program boyunca, ana bellekte kalırlar. O nedenle, onarı *extern* diye ni-

telemeye gerek yoktur. Ancak, bazı durumlarda bir blok içindeki değişkenin program boyunca bellekte kalması gerekebilir. O zaman değişken *static* diye nitelenir.