1

Array

(Dizim)

1.1 Array Nedir?

Array (dizim), aynı veri tipinden çok sayıda değişkeni barındıran bir yapıdır. Array'e ait değişkenler, sıra numarası verilmiş bir liste gibidir. Listedeki öğelere sıra numarasıyla erişilir. Matematikteki sonlu dizidir. Örnekse, matematikte 100 öğesi olan bir sonlu diziyi

$$a = \{a_0, a_1, a_2, \ldots, a_{99}\} \tag{1.1}$$

biçiminde gösteririz; yani diziye ait öğeleri alt indisleriyle belirtiriz. a_7 simgesi, dizinin indisi 7 olan öğesini belirtir. Ayrıca, diziyi tanımlarken, öğelerinin hangi kümeden (veri tipi) alındığı belirtilir. Örneğin, *"tamsayı dizisi"*, dersek, öğelerin tam sayılar kümesinden seçileceği anlaşılır. Bunu ya sözle ifade ederiz, ya da

$$a_i \in \mathbb{Z},\ (i = 0, 1, 2, \ldots, 99) \tag{1.2}$$

simgesiyle belirtiriz.

1.2 Array Bildirimi

Önce array'in veri tipi belirtilir, sonra array'e bir ad verilir. Array'in adını izleyen köşeli parantez ([ ]) içine arrayin bileşen sayısı yazılır. Çok boyutlu arrayler için, her boyuta karşılık bir köşeli parantez kullanılır ve içine o boyuttaki bileşen sayısı yazılır.

Örnekler:

```
1  int intArr[ 100 ] ;       /* 100 bileşenli tek boyutlu int
    tipinden array bildirimi  */
   float floatArr[90];       /* 90 bileşenli tek boyutlu float
    tipinden array bildirimi */
   int tabloArr[5][3];       /* 5 satır ve 3 kolonu olan int
    tipinden iki boyutlu array */
```

Bilgisayar programları basit editörlerle yazılır. O editörler a_i simgesindeki gibi alt indis koyamaz. Üstelik, derleyiciler de alt indisi anlamaz. O nedenle, alt indis işlevini görmek üzere, indisler köşeli parantez içine yazılır. Dolayısıyla, arrayin öğeleri

$$a = \{a[0], a[1], a[2], \ldots, a[99]\} \tag{1.3}$$

biçiminde yazılır. Arrayin öğelerinin, örneğin *int* veri tipinden olacağını belirtmek için de

$$int \;\; a[n]; \tag{1.4}$$

deyimini yazarız. Buradaki [ ] simgesine array operatörü denilir. Görüldüğü gibi, bu operatörün iki işlevi vardır:

1. (1.4) deyiminde array bildirimi yapar.
2. (1.3) ifadesine { } içindeki a[i] terimleri, arrayin bileşenleridir. Genel olarak, array adından sonra gelen [ ] içindeki tamsayı, arrayin hangi bileşeni olduğunu gösterir.

Dikkatle incelersek, (1.1) ile (1.3) benzerdir. (1.2) ile (1.4) aynı işleve sahiptirler.

`Array` yapısı hemen her dilde çok önemli roller üstlenir. İlk yaratılan üst düzeyli dillerden itibaren her dilde bu yapı vardır. Sonra yazılan *C, C#, python, ruby* gibi diller, bu yapıya, klasik array yapısındaki kısıtları kaldıran nitelikler eklemişlerdir.

Matematikteki *sonlu dizi* kavramına benzeyen *array* kavramına her programlama dilinde gerekseme duyulması nedeni çok açıktır. Aynı veri tipinden birkaç değişken tanımlamak için genel kural izlenerek gerekli sayıda

değişken bildirimi yapılabilir. Ama, aynı veri tipinden, onlarca, yüzlerce ve hatta binlerce değişkene gerekseme duyuyorsak, onları tek tek tanımlamak, pratikte, olanaksız değilse bile çok çok zordur.

Öte yandan, array yapısı içinde yer alan çok sayıda bileşen (herbiri bir değişkendir) ile ilgili işleri yapmak için özel fonksiyonlar yazılmıştır. Söz konusu fonksiyonlar, array yaratma, arraylerle işlem yapma, array içinde bileşen arama ve array'in bileşenlerini sıralama, kopyalama gibi array ile ilgili hemen her işi yapmaya yeterlidir. Bu fonksiyonlar her array'e uygulanabilecek biçimde yazıldılar. Böylece, başka programlarda yeniden kullanılabilir (reusable) niteliğe kavuştular.

C dilinde array ile ilgili işleri yapan çok sayıda fonksiyon vardır. Özellikle, arama ve sıralama foksiyonları arrayler üzerinde etkilidirler. Ayrıca string'ler birer char arrayidir. Stringlerle yapılan işlemlerin hepsini array yapısından yararlanarak yaparız. Onların başlıcalarını ilerleyen sayfalarda ele alacağız.

C dilinde her veri tipinden array tanımlanabilir. Onların herbirisi için tek boyutlu ya da çok boyutlu arrayler yaratılabilir. Array'in bileşen türü *int, float, char, vb* ne olursa olsun, onun anabellekte konuşlanış ilkesi değişmez. Array'in adı bir referans (işaretçi, pointer) tipidir; array yapısının anabellekteki ilk bileşeninin adresini işaret eder. Array'in sonraki bileşenleri anabellekte ardışık adreslere yerleşir. Bu adresler, array'in veri tipinin büyüklüğüne göre array'in öğelerine atanacak değerlerin sığacağı büyüklükteki hücrelerdir. Daha açık bir deyişle, *char* tipi array için bileşenlere 1 byte, *int* tipi array için bileşenlere 2 byte (ya da 4 byte), *float* tipi için 4 byte, vb ayrılır. Bu özelik, array'in bileşenlerine seçkili (random) erişimi sağlar. O erişimin nasıl sağlandığını biraz sonra göreceğiz.

Array ile pointer tipleri birbirlerinden farklı iki yapısal tiptir. Ama, uygulamada array, bir pointer gibi kullanılabilir.

1.3 Tek Boyutlu Array

Çok boyutlu array'lerin nitelikleri tek boyutlu arraylerinki gibidir. O nedenle, arraylerin özeliklerini tek boyutlu arrayler üzerinde söylemek yeterlidir. Daha sonra çok boyutlu array'lere örnekler vereceğiz.

Tek boyutlu array, matematikteki tek boyutlu sonlu dizidir; onlar tek indisli (index, damga) dizimlerdir. 0-ıncı bileşenden başlayıp sonuncu bileşenine kadar indis sırasıyla dizilirler. İndisler negatif olmayan tam sayılardan (doğal sayılar) seçilir.

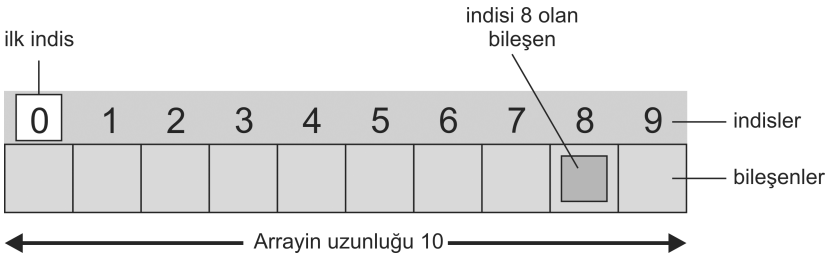
1.4 [ ] Operatörü

[ ] operatörünün iki işlevi olduğunu söylemiştik: Birincisi, (1.4) deyiminde olduğu gibi, array bildirimindeki işlevidir. İkincisi ise (1.3) gösteriminde olduğu gibi, arrayin bileşenlerinin sıra numarasını (indis) içermesidir.

(1.3) gösterimindeki $a[i]$ ($i = 0, 1, 2, \ldots$) öğelerine *arrayin öğeleri*, *arrayin terimleri* ya da *arrayin bileşenleri* denilir. Bu terimler eş anlamlıdır. [ ] operatörü, arrayin bileşenlerini (terim, öğe), indisleriyle (damga, index) belirler. $a[i]$ simgesinde köşeli parantez içindeki i tam sayısına indis *(index, damga)* denilir. Arrayde indisler daima tam sayıdır ve 0'dan başlar. n bileşeni olan arrayin indisleri ($i = 0, 1, 2, \ldots n-1$) olur.

[ ] operatörünün işlevlerini birazdan daha yakından inceleyeceğiz.

1.5 Array Yaratma



Şekil 1.1: Array

Bir array yaratmak, bir değişken yaratmak gibidir. Her değişkenin bir *adı*, ana bellekte bir *adresi* ve o adrese konulan *değeri* vardır. Değişken adı, ana bellekte ona ayrılan yeri işaret eden bir *işaretçidir* (pointer).

Array için de aynı kural geçerlidir. Her array'in bir *adı*, ana bellekte bir *adresi* ve o adrese konulan *bileşen değerleri* vardır. Array'in adı, ana bellekte arraye ayrılan yerin ilk hücresini işaret eden bir *referanstır (işaretçi, pointer)*.

Array yaratmak, bir veri tipinden değişken yaratmaya benzer. Bir değişken bildirimi yapılınca, ana bellekte ona bir adres ayrılır ve o değişken adı, kendisine ayrılan adresi işaret etmeye başlar. Değişkenin bir tek adresi (hücre) vardır ve o adrese tek bir değer konulabilir. Array bildiriminde ise, tek deyim ile aynı ad ve aynı veri tipinden çok sayıda değişken aynı anda yaratılmış olur.

Değişken bildirimi ile array bildirimi arasındaki önemli iki fark şudur:

1. Array adı, bildirimi yapılan array'in ana bellekteki ilk bileşeninin adresini işaret eder.
2. Array sonlu bir dizidir. Onun bileşenlerine ana bellekte ardışık hücreler ayrılır. Hücre sayısı bileşen sayısı kadardır. Bu hücrelere (adres), sırasıyla, array'in terimlerinin değerleri atanır.

Kısaca söylersek, C dilinde bir *array* yaratmanın iki aşaması vardır:

1. Birinci Aşama : Array bildirimi
2. İkinci Aşama : Array'in bileşenlerine değer atama

Bu iki aşama birleştirilip tek deyim haline getirilebilir. Şimdi bu iki aşamanın nasıl yapıldığını bir örnek üzerinde görelim.

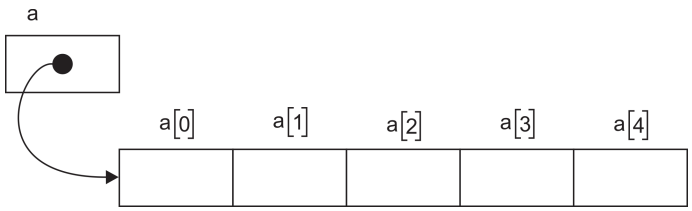
Örnekler:

Bir toptancı mağazasında mevcut malları gösteren bir stok kontrol programı düşünelim. Tabii, stok kontrol programı, çok daha elverişli yapılarla kurulmalıdır. Ama şimdilik, depodaki her eşyaya bir numara vermek için bir array yaratalım. Array'e depodaki nesneleri ifade etmek üzere *mal*, *emtia*, *eşya* gibi çağrışım yapan bir ad vermek uygun olur. Ama gösterimlerde kısalığı sağlamak için, arrayin adını a koyalım.

$$int \quad a[5]; \tag{1.5}$$

deyimi `int` tipinden `a` adlı bir array bildirimidir. (1.5) deyimi bileşenleri (terimleri, öğeleri) *int* tipi olan 5 bileşenli bir array bildirir. Her birine `int` tipi bir değer atanacağna göre, bileşenlere ayrılan bellek adreslerinin büyüklüğü 2 şer (ya da 4 er) byte olmalıdır. Beş bileşen olduğuna göre, arrayy'e ayrılan toplam adres $2 \times 5 = 10$ byte boyundadır. Bazı sistemlerde `int` veri tipine 4 byte ayrılır. O sistemler için, (1.5) arrayine anabellekte $4 \times 5 = 20$ byte boyunda yer ayrılır.

Array'in adı, C dilinde adlandırma kuralına göre verilir ve öteki değişken adlarının oynadığı role benzer bir rol üstlenir. Daha önce söylediğimiz gibi, array adı, array yapısına ana bellekte ayrılan ardışık adreslerin ilkini gösterir; yani array'in ilk bileşeninin adresini işaret eder. Böyle olduğunu görmek için, a'nın ve a[0]'ın adreslerini yazan Program 1.1'nin çıktısına bakınız. İki bellek adresinin aynı olduğunu göreceksiniz.



Şekil 1.2: Boş Array

Program 1.1.

```
#include <stdio.h>

int main() {
  int i;
  int a[5];

  printf("&a    = %d \n" , &a);
  printf("&a[0] = %d ]\n", &a[0]);

  return 0;
}
```

```
/**
&a    = 2686636
&a[0] = 2686636
*/
```

[ ] simgesi array operatörüdür; *a* ise, arrayin adıdır. Array'in adı değişken adı gibidir. Nasıl ki değişken adı, onun ana bellekteki adresini işaret ediyorsa, örneğimizdeki, (1.5) deyimi ile tanımlanan *a*, array'in ilk bileşeninin adresini işaret edecek bir referanstır. (1.5) bildirimi bu aşamasında *a* array'i için ana bellekte bir yer ayırmıştır, ama henüz array'in bileşenlerine hiç değer atanmamıştır.

(1.5) deyiminde *a* arrayin adı, [ ] ise array yaratan operatördür. [5] ifadesi *int* tipi veri tutacak 5 hücre yaratır. Tabii, 5 yerine, istenilen başka bir sayı konulabilir. O sayı arrayin bileşen (terim) sayısıdır. Benzer olarak, *int* yerine başka bir veri tipi konulabilir.

Ana bellekte arrayin bileşenleri olarak ayrılan her hücre, arrayin ait olduğu veri tipinden verilerin sığacağı büyüklüktedir. Örneğimizde *a* arrayi *int* tipi olduğundan, ona ayrılan bileşen hücrelerinin herbirisi $2 \times 8 = 16$ bit boyundadır. Bazı sistemlerde $4 \times 8 = 32$ bit boyunda olabilir. *int* yerine *double* tipten bir array olsaydı, herbir bileşen hücresi 64 bit boyunda olurdu.

Bu aşamada, (1.5) deyimi *a* tarafından işaret edilen bir array yarattı. Başka bir deyişle, ana bellekte 5 tane *int* değer tutacak bir yer (5 hücre)

ayırdı. Bunu Şekil 1.2'deki gibi gösterebiliriz. a bellekte ayrılan bu beş hücrelik yerin ilk hücresini; yani $a[0]$ bileşeninin olduğu adresi işaret eder. a'ya referans (işaretçi, pointer) denmesinin nedeni budur.

1.6 Arrayin Boyu (Uzunluğu)

Tanım 1.1. *Bir arrayin boyu (uzunluğu) onun bileşenlerinin sayısıdır.*

Program 1.1 Örneğinde, arrayin işaret ettiği adreste, *int* tipi veri tutacak şekilde yaratılan 5 tane bellek adresi (5 hücre) ana bellekte ardışıktırlar. Arrayin terimleri (bileşenleri) [ ] içine yazılan indislerle numaralanır. İlk hücrenin adresi daima 0 indisiyle başlar. $a[0]$ ilk hücreyi temsil eden değişken adıdır. Sonrakiler artan numara sırası izler.

Özetle,

Liste 1.1.

```
a[0]
a[1]
a[2]
a[3]
a[4]
```

adları, arrayin bileşenlerini gösterirler.Onların herbirisi *int* tipi bir değişkendir. Dolayısıyla, *int* tipi değişkenlerle yapılabilen her işlem, onlara da uygulanabilir.

sizeof() fonksiyonu, bir değişkene ya da yapısal bir veri tipine ana bellekte ayrılan bellek adresinin büyüklüğünü *byte* cinsinden verir. Bu demektir ki, array'e ana bellekte ayrılan adresin boyunu byte cinsinden bulabiliriz. Arrayin veri tipi bilindiğine göre, her bileşene ayrılan bellek büyüklüğü bellidir. Dolayısıyla, arraye ayrılan bellek büyüklüğünü, bir bileşenine bellekte ayrılan büyüklüğe bölersek, arrayin kaç bileşeni olduğu ortaya çıkar.

Program 1.2'in çıktısı, *char, int, float, double* veri tipleri ile *int tipinden array* ve *float tipinden array* için ana bellekte ayrılan bellek büyüklüklerini *byte* cinsinden gösteriyor.

Program 1.2.

```
#include <stdio.h>

int main() {
```

```
4   char ch;
    int i;
    float x;
7   double w;

    int a[5];
10  float b[5];

    printf("char tipine ayrılan yer    :%d byte'dir \n "  , sizeof(ch))
     ;
13  printf("int tipine ayrılan yer    :%d byte'dir \n "   , sizeof(i));
    printf("float tipine ayrılan yer  :%d byte'dir \n "   , sizeof(x));
    printf("double tipe ayrılan yer   :%d byte'dir \n "   , sizeof(w));
16
    printf("float tipinden b arrayine yer       %d byte'dir \n "  ,
     sizeof(b));
    printf("int tipinden a arrayine ayrılan yer %d byte'dir \n "  ,
     sizeof(a));
19
    return 0;
}
```

```
/**
 char tipine ayrılan yer    :1 byte'dir
3 int tipine ayrılan yer    :4 byte'dir
 float tipine ayrılan yer  :4 byte'dir
 double tipe ayrılan yer   :8 byte'dir
6 float tipinden b arrayine ayrılan yer       20 byte'dir
 int tipinden a arrayine ayrılan yer         20 byte'dir
*/
```

1.6.1 Array'in Boyunu Bulma

C dilinde arrayin boyunu bulan bir fonksiyon yoktur. Ama yukarıda söylediğimiz gibi, böyle bir fonksiyonu kolayca yazabiliriz.

```
1     boy = sizeof(arr) / sizeof(arr[0]);
```

formülü ile bulunabilir.

Array'in `a` adı ile `a[0]` ilk bileşeninin adreslerinin aynı olduğu, sonraki bileşenlere de ardışık hücrelerin ayrıldığını görmek için, Program 1.2'nin çıktısına bakmak yetecektir.

Örnekler:

Program 1.3.

```
#include <stdio.h>
2
int main()
{
5   int a[] = {0,1,2,3,4,5,6,7,8,9};
```

```
  int boy = sizeof(a)/sizeof(int);
  printf("Array'in boyu = %d "  , boy);

  return 0;
}
```

```
/**
Array'in boyu = 10
*/
```

Program 1.4, array boyunu makro olarak tanımlıyor.

Program 1.4.

```
#include <stdio.h>
#define BOY( arr ) sizeof( arr ) / sizeof( arr[0] )


int main() {
  char *kentler[] = { "Ankara", "Mardin", "Edirne" };
  printf("Array'in uzunlugu = %d  "    , BOY(kentler));
}
```

```
/**
Array'in boyu = 3
*/
```

Program 1.5 örneğinde de, array boyunu makro olarak tanımlıyor ve array bir pointer gibi kullanılıyor.

Program 1.5.

```
#include <stdio.h>
#define ARRAY_BOY(x) (sizeof(x)/sizeof(*x))


int main() {
  int tekler[] = { 1,3,5,7,9,11,13,1,5,1,7,19 };
  printf("Array'in uzunlugu = '%d  "    , ARRAY_BOY(tekler));
}
```

```
/**
Array'in boyu = 12
*/
```

1.7 Array'in Bileşenleri

int a[ ]= {0,1,2,3,4)} array bildirimi yapılınca $a[i]$ $(i = 0, 1, 2, 3, 4)$ öğelerine a arrayinin *bileşenleri* demiştik. Bileşenler, yaratılan array içinde birer değişkendir. Bu değişkenler sayesinde, array beş tane *int* tipi veriyi

bir arada tutabilme yeteneğine sahip olur. Bu değişkenlere array'in *öğeleri (terimleri)* de denilir. Bu terimleri eşanlamlı sayacağız. $0, 1, 2, 3, 4$ sayıları bileşenlerin sıra numaralarıdır; *Index* (indis, damga) adını alırlar. Sıra numaraları (indis) daima 0 dan başlar, birer artarak gider. n tane bileşeni olan bir array'in ilk bileşeninin damgası 0, son bileşeninin damgası $(n-1)$ olur. Bir array'in *boy (uzunluk, length)* onun bileşenlerinin sayısıdır.

1.8 Array'in Bileşenlerine Erişim

Daha önce de söylediğimiz gibi, bir değişkene *erişmek* demek, ona değer *atamak*, atanan değeri *okumak* ve gerektiğinde değerini *değiştirmek* demektir. Array'in bileşenleri de birer değişken olduklarından, onlara istendiğinde değer atanabileceği, istenirse atanan değerlerin değiştirilebileceği açıktır. `arr` int tipinden bir array ise

```
arr[3] = 12;
```

gibi bir atamanın yapılabilmesi, arrayin her hangi bir bileşenine direk erişilebildiği anlamına gelir. Seçkili erişim, *array*'lerin üstün bir niteliğini ortaya koyar. Array'in istenen bileşenine indis (index) sayısı ile *seçkili* (doğrudan, random) erişmek mümkündür. Her bileşen bir değişken olduğu için, o bileşen tipiyle yapılabilen her işlem onlar için de yapılabilir, değişkenlerle ilgili kurallar bileşenler için de aynen geçerlidir. Gerçekten, yukarıdaki deyim, indisi 3 olan bileşene 12 değerini atamıştır.

1.8.1 Örnekler

Program 1.6.

```
#include <stdio.h>

int main() {
  int i;
  char *kent[5] = {"Van","Trabzon","Kayseri","Mardin","Adana"};
  for ( i = 0; i < 5; i++)
    printf("kent[%d] = %s \n" ,i, kent[i]);
}
```

```
/**
kent[0] = Van
kent[1] = Trabzon
kent[2] = Kayseri
kent[3] = Mardin
kent[4] = Adana
*/
```

C dilinde her veri tipinden array yaratılabilir. Örneğin, yukarıdaki

```
char kent[] = {"Van","Trabzon","Kayseri","Mardin","Adana"};
```

deyimi *char* tipinden bir array bildirmiş, onun bileşenlerine *string* tipinden değerler atamıştır. Bu atama

```
kent[0] = "Van";
kent[1] = "Trabzon" ;
kent[2] = "Kayseri" ;
kent[3] = "Mardin" ;
kent[4] = "Adana" ;
```

atamalarına denktir.

Yukarıda yaptıklarımızı özetleyelim.

```
veriTipi arrayAdı[ bileşen_sayısı ] ;    // array bildirimi
```

Array nesnesi yaratmak demek, ana bellekte array'in bileşenlerine birer yer (bileşen adresi, göze) ayırmak demektir. *array* nesnesi yaratılırken, bileşenlerine ilk değerler atanabilir. Ama ilk değerler atanmazsa, bileşenlere değerleri sonradan atanabilir. Ancak, değer atanmadan önce bileşen değerleri kullanılmamalıdır.

Arrray bileşenlerine değer atama: Array nesnesi yaratıldıktan sonra onun bileşenlerine değer atama eylemi, diğer değişkenlere değer atama gibidir. İstenen bileşen indeks sayısı ile seçilerek (seçkili, random) değer atanabilir. Örneğin, yukarıda yapıldığı gibi,

```
kent[2] = "Kayseri" // atama
```

deyimi *kent* adlı array'in 3.bileşenine *"Kayseri"* değerini atamaktadır.

1.9 Bileşenlerin Öndeğerleri

Anımsayacaksınız, *auto* depo sınıfında olan değişkenlerin öndeğerleri yoktur. Bunlar yerel değişkenlerdir. Eğer kendilerine bir değer atanmadan önce kullanılırlarsa çöp (garbage values) toplarlar. Yani, kendi adreslerinde eskiden arta kalan değerleri kullanırlar.

static, *global* ve *extern* depo sınıfında olan değişkenlerin öndeğerleri 0 dır. Kendilerine değer atanmadan kullanılırlarsa, 0 değerini kullanırlar.

Array'in bileşenleri de birer değişken olduğuna göre, aynı kurala uyarlar. Buna göre,

1. Array tanımı bir blok içinde ise, onun bütün değişkenleri yereldir; yani *auto* depo sınıfındadırlar. Dolayısıyla onların öndeğerleri yoktur. Değer atanmadan kullanılan bileşenler çöp toplar.

2. Array tanımı bütün blokların dışında ise, onun bütün bileşenleri global olur. Dolayısıyla öndeğerleri 0 dır. Array bir blok içinde ama *static* nitelemli ise, onun bütün bileşenleri *static* depo sınıfında olur. Hepsinin öndeğeri 0 dır. Array *extern* depo sınıfında ise, onun bütün bileşenleri de *extern* olacağından, öndeğerleri 0 dır.

3. Bu kurallar çok boyutlu arrayler için de geçerlidir.

Unutmayalım ki, C derleyicisi veri tipi denetimi yapmaz. Değer atanmamış bir değişken ya da array bileşeni kullanıldığında, hiçbir uyarı yapmadan, onun öndeğerini kullanır. Böyle olması ciddi hatalara neden olabilir.

Örnekler:

Program 1.7, global arraylerin bileşenlerine 0 öndeğerinin (default) olarak atandığını gösteriyor.

Program 1.7.

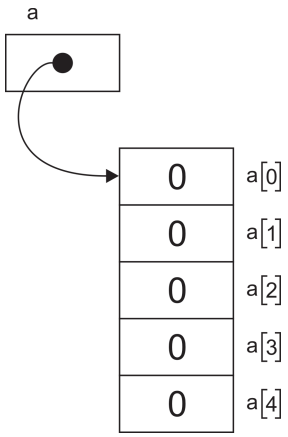
```
#include <stdio.h>

int garr[5];

int main() {
  int i;

  for (i=0; i<5; i++)
    printf("%d\t"  , garr[i]);

  return 0;
}}
```

```
/**
   0   0   0   0   0
*/
```

Açıklamalar:

Hiç bir fonksiyonun ya da blokun içinde olmayan değişken ve arrayler *global*'dır. Global değişkenlere ve arraylere, programın her tarafından erişilebilir. Program 1.7 örneğinde `garr` arrayi global'dir. `garr` arrayinin bileşenlerine değer atanmadığ halde, bileşen değerleri 0 olmuştur. Çünkü, global array'lerin bileşenlerine 0 öndeğeri (default) atanır.

Bildirim anında önüne *static* nitelemi konulan yerel değişkenler ve arrayler *static* olurlar. *static* değişkenlerin adresleri, program boyunca korunur. Değer atanmamış *static* değişkenlerin öndeğeri (default) 0 dır. Ancak yerel değişkenler static nitelemi alabilir; global ve extern değişkenler static olamazlar. Onlar zaten program süresince adreslerini korurlar.



Şekil 1.3: static, global ve extern bileşenlerin Öndeğerleri (default)

Program 1.8.

```
#include <stdio.h>

int main() {
  int i;
  static int starr[5];

  for (i=0; i<5; i++)
    printf("%d\t"  , starr[i]);

  return 0;
}
```

```
/**
   0   0   0   0   0
*/
```

Açıklamalar:

`starr` arrayinin bileşenlerine değer atanmadığ halde, 9.satırın verdiği çıktıda bileşen değerleri 0 olmuştur. Çünkü, arraylerin bileşenlerine 0 öndeğeri (default) atanır.

1.10 Seçkili (random) Erişim

Liste 1.2.

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

```
3
int main() {
  int i;
6 int a[5];
  printf("&a     = %d \n" , &a);

9 for (i=0; i<5; i++)
    printf("&a[%i] = %d \n", i, &a[i]);
  return 0;
12 }
```

```
/**
&a     = 2686632
3 &a[0] = 2686632
&a[1] = 2686636
&a[2] = 2686640
6 &a[3] = 2686644
&a[4] = 2686648
*/
```

Program 1.2, `int` tipe 4 byte ayıran bir bilgisayarda yazılmıştır. Bileşenlerin adreslerine bakarsanız, ardışık ikisi arasında 4 byte olduğunu göreceksiniz.

Array adı, anabellekte &a= 2686632 adresini gösteriyor. &a= &a[0] dır. &a[1] adresini bulmak için, a[0] adresine 4 byte ekliyor. Benzer olarak &a[2], &a[3] ve &a[4] bileşen adreslerini bulmak için, &a adresine, sırasıyla, 2×4, 3×4, 4×4 byte ekliyor. Bu işi derleyici yapıyor. Programda arrayin bileşenlerinin adreslerini belirtmeye gerek yoktur. Bu olgu, array'e özgü bir niteliktir. Bu nitelik, arrayin bileşenlerine seçkili (random) erişimi sağlar.

Doğal olarak, indis sırası kullanılırsa, arrayin bileşenlerine sıralı erişilebilir. Ama, özünde yapılan iş gene seçkili erişimdir.

1.11 Bileşenlere Değer Atama Yöntemleri

1.11.1 Seçkili (random) Atama

```
1     a[2] = 1234;
```

ataması, array'in üçüncü bileşenine 1234 değerini atar. Öteki bileşenlere de benzer yolla atamalar yapılabilir. Derleyici, bileşenlerin adreslerini yukarda söylenen yöntemle bulur. Öneğin, &a[2] adresini bulmak için a'nın adresine, bir bileşeninin büyüklüğünün iki katını ekler. &a[2]= &a+ (2 ×4 byte) olur.

1.11.2 Bildirim Anında Atama

Bildirim ve seçkili atama aşamalarını birleştirerek tek bir deyimle her ikisini bir arada yapabibiliriz. Örneğin,

$$int \quad a[\,] = \quad \{1123,\ 1125,\ 1234,\ 1256,\ 1321\}; \tag{1.6}$$

deyimi hem array bildirimini hem bileşenlere değer atamasını birlikte yapar. Bileşenlere atanacak değerler, sırasıyla, { } parantezi içine, virgül ile birbirlerinden ayrılarak yazılmıştır. Bu atama yönteminde, bütün bileşenlere sırasıyla değer atanıyor. `a` arrayinin bileşen sayısını ayrıca yazmaya gerek kalmıyor; çünkü bileşenlere atanan değerler, bileşenlerin sayısını ve sırasını belirliyor[1]. Bunu görmek için Program 1.9'i çözümleyiniz.

Program 1.9.

```
#include <stdio.h>

int main() {
  int i;
  int    a[]  =  {1123, 1125, 1234, 1256, 1321 };
  printf("%d\n" , &a);

  for (i=0; i<5; i++)
    printf("a[%i] = %d \n"  , i, a[i]);
  return 0;
}
```

```
/**
&a    = 2686632
a[0] = 1123
a[1] = 1125
a[2] = 1234
a[3] = 1256
a[4] = 1321
*/
```

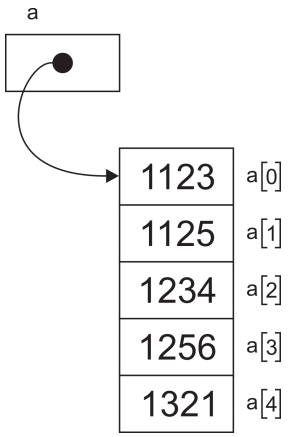
deyimi ile bildirim aşaması (ilk aşama) tamamlanır, bileşenlere değer atama işi sonra seçkili atama ile yapılabilir.

Aşağıdaki iki yöntem aynı işi yapar.

1.Yöntem: [ ] operatörü ile array yaratırken boyunu belirleyip, sonradan bileşenlerine seçkili (random) değer atama.

```
  int =   arr[10];
  arr[6]=70; arr[1]=20;  arr[2]=30; arr[7]=80;  arr[4]=50;
  arr[3]=40; arr[0]=10;  arr[8]=90; arr[9]=100; arr[5]=60;
```

[1] C_{99} sürümünde bileşenlere sıralı olmayan atamalar yapılabilir



Şekil 1.4: Arayin Bileşenlerinin Hepsine Değer Atama

deyimi 10 bileşenli `arr` adlı array yaratır sonra bileşenlerine seçkili (random) değer atar.

2.Yöntem: [ ] operatörü ile array'i yaratırken, bileşenlerine sırayla ilk değerlerini atama.

```
int arr[] = { 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 };
```

deyimi 10 bileşenli bir array yaratır ve bileşenlerine sırayla değerler atar. `arr[]` ifadesi array'in boyunu belirlemez; ancak { } bloku içine sırayla yazılan değerler array boyunu ve her bileşene sırayla atanan değeri kesinlikle belirler. { } içindeki değerlerin yazılış sırası ile bileşenlerin indis sırası uyumludur. Örneğin, { } içindeki 7.değer 7.bileşene aittir. Tabii, 7.bileşenin indisinin 6 olduğunu biliyoruz; çünkü damgalama işlemi 1 den değil 0 dan başlar. O nedenle `arr[6] = 70` dir. Array'lerde bu özeliği daima anımsamalıyız.

İstenirse,

1.11.3 Örnekler

Liste 1.3.

```
int derslik[3];
int derslik[3] = {0, 1, 2};

string kent[] = {"Erzurum","Van","Samsun","Adana","Muğla"};

```

```
string kent[5] = {"Erzurum","Van","Samsun","Adana","Muğla"};

string kent[5]; kent[4] ="Muğla"; kent[1] = "Van"; kent[3] = "
 Adana"; kent[2] = "Samsun"; kent[0] ="Erzurum";
```

Açıklamalar:

1. İlk satır: *int* tipinden *derslik* adlı 3 bileşenli array bildirimini yapıyor. Bileşenlere değer atamıyor.
2. İkinci satır: önce *int* tipinden *derslik* adlı 3 bileşenli array bildirimini yapıyor. Bileşenlere sırayla değer atıyor.
3. Dördüncü satır: *string* tipinden *kent* adlı beş bileşenli bir array bildiriyor. Ama atanan değerlerin sayısı bileşen sayısını ve sırasını belirliyor. Bileşenlere yapılan atama, arrayin boyunu dinamik olarak, belirliyor. Tabii, bu atama deyiminden sonra yeni atamalar yapılarak, *kent* arrayinin boyu değiştirilemez.
4. Altıncı satır: *String* tipinden *kent* adlı beş bileşenli bir array bildiriyor. Sonra bileşenlere ilk değerlerini sırayla atıyor.
5. Sekizinci satır: *String* tipinden beş bileşeni olan *kent* adlı bir array bildiriyor. Bildirim anında array yaratılmış olur. Sonra bileşenlere ilk değerleri seçkili (random) atanıyor.

1.12 Bileşenlerle İşlem

Array aynı veri tipinden çok sayıda değişkenin bir deyimle bildirilmesini sağlayan yapısal bir tiptir. Arrayin her öğesi bir değişkendir. Ona indisiyle her zaman erişilebilir. O halde, onlarla, ait olduğu veri tipinde yapılabilen bütün işler yapılabilir. Örneğin *int* tipinden bir arrayin öğeleri arasında int tipinden değişkenler için yapılabilen bütün işlemler yapılabilir. Ayrıca, aynı tipten olan arraylerin bileşenleri arasında da işlemler yapılabilir.

1.12.1 Atama Yöntemiyle Değer Aktarma

Bir arrayin bileşenlerinin değerleri aynı veri tipinden başka değişkenlere aktarılabilir. Doğal olarak, bu işin tersi de yapılabilir. Aslında, aktarma işlemi, atama deyimi ile yapılan kopyalama işlemidir. Burada atama ile yapılabilen kopyalama eylemlerine örnekler vereceğiz.

x değişkeninin veri tipi a arrayi ile aynı olsun. Örneğin,

```
1    int x;
     int a[5];
```

bildirimleri yapılmışsa $a[2] = 1234$; atamasından sonra

```
1    x = a[2];
```

ataması, `a[2]` bileşeninin değerini x değişkenine aktarır; öyleyse, bu atama deyimi

```
     x = 1234 ;
```

atamasına denktir.

Tersine olarak, y aynı tipten bir değişken ise, yani *int y;* bildirimi yapılmışsa,

```
     int  y = 567;
2    a[2] = y;
```

ataması geçerlidir. Bu atama y'nin değerini `a[2]` bileşenine aktarır; dolayısıyla,

```
1    a[2] = 567;
```

atamasına denktir.

Array'in bir bileşeninin değeri başka bir bileşenine aktarılabilir. Örneğin,

```
     a[3] = a[2];
```

ataması geçerlidir.

1.13 Array Üzerinde Döngü

Array yapısının çok işe yaradığı yerlerden birisi döngülerdir. Daha önceleri genel *for* döngüsünü kullanmayı öğrendik. For döngüsünde sayaç olarak arrayin indisleri alınırsa, arrayin bileşenleri üzerinde for döngüsü kurulmuş olur. Array üzerinde for döngüsü kullanılarak array kopyalanabilir, arrayin öğeleri toplanabilir, sıralanabilir, array içinde bir öğenin olup olmadığı araştırılabilir, arrayin en büyük ve en küçük öğeleri bulunabilir. Bu eylemlerin yapılışını örneklerle göreceğiz.

1.14 Array Öğelerini Toplama

Program 1.10 arrayin öğelerini topluyor.

Program 1.10.

```
#include <stdio.h>

int main() {
  int i,toplam = 0;
  int a[10] = { 10,20,30,40,50,60,70,80,90,100};
  for (i=0; i<10; i++)
    toplam = toplam + a[i];
  printf("%d" ,toplam);
}
```

```
/**
  550
*/
```

1.15 Array Kopyalama

C dilinde bir arrayi başkasına bir bütün olarak kopyalayan deyim yoktur. O nedenle, bir döngü yardımıyla kaynak arrayin bileşenleri başka bir arrayin bileşenlerine kopyalanabilir. Ancak, array bir string ise, *strcopy()* fonksiyonu kullanılabilir.

Program 1.11 bir stringi (char arrayi) *strcpy()* fonksiyonu ile başka bir array üzerine kopyalıyor.

1.15.1 strcpy() ile *char* array kopyalama

Program 1.11.

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

int main()
{
   char kaynak[1000], hedef[1000];

   printf("Input a string\n");
   gets(kaynak);

   strcpy(hedef, kaynak);

   printf("Kaynak string:      \"%s\"\n", kaynak);
   printf("Hedef string: \"%s\"\n", hedef);

   return 0;
}
```

```
/**
Input a string
C dilini öğreniyorum
Kaynak string:       "C dilini öğreniyorum"
Hedef string: "C dilini öğreniyorum"
*/
```

strcopy() fonksiyonu Bölüm **??**'de yeniden ele alınacaktır.

1.15.2 Döngü ile array kopyalama

Program 1.12 *int* arrayini döngü ile başka bir array üzerine kopyalıyor.

Program 1.12.

```
#include <stdio.h>

int main() {
  short a[] = {21,32,24,35};
  short b[4];

  int i=0;
  for (i=0; i<4; i++)
    b[i] =a[i];


  for (i=3; i>=0; i--)
  printf("%d\t", b[i]);


  return 0;
}
```

Program 1.13 *char* arrayini döngü ile başka bir array üzerine kopyalıyor.

Program 1.13.

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

void copy_string(char [], char []);

int main() {
  char s[1000], d[1000];

  printf("Input a string\n");
  gets(s);

  copy_string(d, s);

  printf("Source string:      \"%s\"\n", s);
  printf("Destination string: \"%s\"\n", d);
```

```
16
    return 0;
}
19
void copy_string(char d[], char s[]) {
    int c = 0;
22
    while (s[c] != '\0') {
      d[c] = s[c];
25    c++;
    }
    d[c] = '\0';
28  }
```

1.15.3 Arrayler Üzerinde İstatistik

Program 1.14 örneği, bir arrayin bileşen değerlerinin aritmetik ortalamasını buluyor.

Program 1.14.

```
#include <stdio.h>
2 #define BOY 5

int main() {
5   int ortalama = 0;
    int toplam =0;
    int x=0;
8
    int arr[BOY];

11  /* arrayin bileşen */
    for (x=0; x < BOY; x++) {
      printf("sayıları giriniz %d\n", x);
14    scanf("%d", &arr[x]);
    }
    for (x = 0; x < BOY; x++) {
17    toplam  += arr[x];
    }

20  ortalama = toplam/BOY;
    printf("Ortalama : %d", ortalama);
    return 0;
23  }
```

```
1 /**
  sayıları giriniz 0
  56
4 sayıları giriniz 1
  45
  sayıları giriniz 2
7 23
  sayıları giriniz 3
  -23
```

```
sayıları giriniz 4
64
Ortalama: 33
*/
```

1.15.4 Array Sıralama

Bir veri topluluğunu artan ya da azalan sıraya dizmek, bütün programlama dillerinde önem taşır. Programlama tarihi boyunca, çok farklı sıralama algoritmaları geliştirilmiştir. Bu algoritmalar arasında en iyisi yoktur. Sıraya konulacak veri topluluğuna göre bazen birisi bazen ötekisi etkin olur. Sıralama algoritmalarının hemen hepsi, topluluktan seçtiği iki öğeyi mukayese eder, artan sıraya dizecekse, küçük olanı öne alır. Bu bir takas (swap) eylemidir.

Şimdilik çok ayrıntıya inmeden, sıralamanın nasıl yapıldığını göstermek için, köpük sıralaması *(buble sort)* denilen algoritmaya bir örnek vereceğiz. Hemen belirtelim ki, köpük sıralaması, sıralama algoritmaları arasında en yavaş olanıdır. Ama sıralamanın nasıl olduğunu göstermesi bakımından, öğretici değeri vardır.

Program 1.15.

```
#include <stdio.h>
#define BOY   10

int main() {
  int a[BOY] = {43,5,14,59,0,27,78,-5,65,35};
  int i, adim, yedek;
  printf("Kaynak array\n\n");
  // Kaynak arrayi göster
  for(i = 0; i < BOY; i++)
    printf("%d  ", a[i]);
  // ------sıralama----------
  // ardışık öğe çiftlerini karşılaştır; küçük olanı öne al
  for(adim = 1; adim < BOY; adim++)

    for(i = 0; i < BOY; i++)
      // set the condition...
      if(a[i] > a[i + 1]) {
        // takas
        yedek = a[i];
        // put the a[i + 1] in a[i]
        a[i] = a[i + 1];

        a[i + 1] = yedek;
      }
  printf("\n\narrayin artan sırada dizilmiş hali:\n\n");

  for (i = 0; i < BOY; i++)
```

```
    printf("%4d", a[i]);
29  printf("\n\n");
    return 0;
}

/**
2   Kaynak array:
    43  5  14  59  0  27  78  -5  65  35

5   arrayin artan sırada dizilmiş hali:
      -5   0   5  14  27  35  43  59  65  78
*/
```

Açıklamalar:

5.satır: Sıralanacak arrayin bildirimidir.

8.satır: for döngüsü ile kaynak arrayin öğeleri yazılıyor.

15.satırdaki içteki for döngüsü ardışık iki öğeyi mukayese ediyor; küçük olanı öne alıyor (takas).

13.satır: dıştaki for döngüsü arrayin ilk öğesinden başlayarak, her öğeyi arrayin öteki öğeleriyle mukayese etmesi için içteki for döngüsüne yolluyor.

27.satır: Sıralanmış arrayi yazıyor.

1.16 Arama

Bir verinin array içinde olup olmadığını bulmak için, arrayin her öğesi aranan öğe ile karşılaştırılır. Bir bileşen değeri aranan değere eşitse, onun indisi yazılır. Böylece aranan öğenin hem array içinde olduğu hem arrayin kaçıncı bileşeni olduğu belirlenmiş olur.

Program 1.16.

```
#include <stdio.h>
2
int main()
{
5   int x= 3;
    int a[] = {1,5,8,7,2,3};
    printf("%d"  ,bul(a, 3));
8
}
int bul(int a[], int aranan) {
11  int i;
    int bayrak=0;
    for(i=0; i<6; i++) {
14    if(a[i]==aranan) {
        bayrak=1;
```

```
      return i+1;
      break;
    }
  }
  if(bayrak==0) {
    return 0;
  }
}
```

```
/**
6
*/
```

Burada a[ ] arrayinin *bul()* fonksiyonuna parametre oluşuna dikkat ediniz. İzleyen kesimde bunu konu edineceğiz.

Arrayler fonksiyonlarda paramatre olarak kullanılırken izlenen yöntemlerden birisi şudur: Çağrılan fonksiyonun argümanı olarak arrayin adı yazılır. Arrayin kaç öğesinin kullanılacağını belirtmek için, ikinci bir argümana gerek vardır. Program **??** o yöntemi kullanıyor.

Program 1.17.

```
#include <stdio.h>

/* fonksiyon bildirgesi (prototype)*/
double ortalamaHesapla(int arr[], int boy);

int main () {
  /* 5 ogeli int arrayi*/
  int puan[5] = {97, 38, 34, 75, 89};
  double ortalama;

  /* arrayi argument olarak gecir */
  ortalama = ortalamaHesapla( puan, 5 ) ;

  /* ciktiyi yazdir */
  printf( "Ortalama : %lf "  ,ortalama);

  return 0;
}

double ortalamaHesapla(int arr[], int boy) {
  int    i;
  double ort;
  double toplam;

  for (i = 0; i < boy; ++i) {
    toplam += arr[i];
  }

  ort = toplam / boy;

  return ort;
}
```

```
1 /**
    ortalama :   67.800000
*/
```

Açıklamalar:

20-32.satır: *ortalamaHesapla()* fonksiyonunu tanımlıyor.

3.satır: bildirge.

12.satır: *ortalamaHesapla()* fonksiyonunu çağırıyor. Çağrıda, 6.satırda tanımlanan arrayi parametre olarak kullanıyor. Fonksiyonun ikinci parametresi yerine arrayin boyu olan 5 sayısı konuluyor. Fonksiyonun verdiği *double* tipinden olan değer *ortalama* değişkenine atanıyor. 15.satır bu değeri yazıyor.

26.satır: arrayin bileşen değerlerini topluyor. Bileşen değerleri *int* tipinden olduğu için *toplam* int olur.

29.satır: toplamı bileşen sayısına bölerek ortalamayı buluyor. Bulunan ortalama iki tamsayının bölümü olarak *int* tipinden olmalıdır. Ama 23.satırda *ort* değişkeni *double* olarak bildirildiği için, *ort* sayısı otomatik olarak *double* tipe dönüştürülüyor. İstenirse,

```
ort = toplam / boy;
```

tip dönüşümü yapılabilir.

Uyarı 1.1. *Arrayin bileşenleri sayısal veri tipleri arasındaki genel dönüşüm kuralına uyar (bkz.* **??***).*

Buna göre, *int* tipinden *float* ve *double* tiplere dönüşümü C derleyicisi gerektiğinde otomatik olarak yapar. Buna istençsiz dönüşüm (*implicit conversion*) diyoruz. Ama, daha büyük bellek adresi olan sayısal verileri daha küçük bellek adreseri olan sayısal veri tiplerine dönüştürürken veri kaybı oluşur.

1.17 Çok Boyutlu Arrayler

Çok boyutlu arrayler, bileşenleri birden çok indise (damga,index) bağlı olan arrayler'dir. Çok boyutlu arraylerin bildirimi, yaratılması ve bileşenlerine değer atanması eylemleri, bir boyutlu arraylerde yapılanlar gibidir.

Aşağıdaki bildirimler, sırasıyla 2, 3 ve 4 boyutlu birer array bildirimidir.

Liste 1.4.

```
int[][]          arr2Boyut ;   // 2 boyutlu array bildirimi
float[][][]      arr3Boyut ;   // 3 boyutlu array bildirimi
float[][][][]    arr4Boyut ;   // 4 boyutlu array bildirimi
```

Bir boyutlu arrayler için yaptığımız gibi, çok boyutlu arraylerin bileşenlerine de bildirim anında değer atayabiliriz. Liste 1.5 örneğindeki array, bileşenleri *int* tipi olan 2-boyutlu bir arraydir.

Liste 1.5.

```
int ikilSayı[][] = new int[][] { {1, 2}, {3, 4}, {5, 6} };
```

İki boyutlu arrayleri birer matris gibi düşünebiliriz. Örneğin, *ikilSayı* adlı arrayin bileşenleri 3x2 tipi bir matris gibi dizilebilir. `ikilSayi[][]` arrayinde, köşeli parantezlerinden ilki temsil ettiği matrisin *satır* numaralarını, ikincisi ise *kolon* (sütun) numaralarını belirler.

1.17.1 Çok Boyutlu Arraylerin Bileşenlerine Erişim

Çok boyutlu arraylerin bileşenlerine erişim, tek boyutlularda olduğu gibidir. Değer atama ve atanan değeri okumak için seçkili (direk, random) erişim yapılabilir. Değer atarken seçkili ya da sıralı atama yapılabilir. Program 1.6 örneği, iki boyutlu arrayin bileşenlerine seçkili atama ayapıyor, sonra onları bir döngü ile sırayla yazıyor.

Liste 1.6.

```
#include <stdio.h>

int main() {
  int i,j;
  int arr[2][3] ;
  arr[0][0] = 1 ;
  arr[0][1] = 2 ;
  arr[0][2] = 3 ;
  arr[1][0] = 4 ;
  arr[1][1] = 5 ;
  arr[1][2] = 6 ;
  // arrayi bir tablo halinde yaz
  for (i=0; i < 2; i++) {
    for (j =0; j < 3; j++)
      printf("%d " , arr[i][j]);
    printf("\n");
  }
}
```

```
/**
   1  2  3
   4  5  6
*/
```

Tek boyutlu arrayler için söylediğimiz gibi, C dilinde *çok boyutlu array* yapıları üzerinde topluca işlem yapan hazır fonksiyonlar yoktur. Ancak, çok boyutlu arrayleri kurmaya ve onlar üzerinde işlem yapacak fonksiyonların serbestçe tanımlanmasına izin verir. Örneğin, *veri_tipi* C dilinde bir veri tipi olsun.

```
b1, b2,        b3, ... , bN
```

birer pozitif tamsayı olmak üzere,

Liste 1.7.

```
veri_tipi  arr[b1][b2][b3]...[bN]
```

bildirimi N boyutlu bir array tanımlar. Bu bildirimde geçen terimlerin anlamları Tablo 1.1'de yazılıdır.

veri tipi	arayin veri tipi; bileşenlerinin alacağı değer tipi
arr	arrayin adı
N	arrayin boyut sayısı
bK	$K = 1, 2, 3, ..., N$ olmak üzere, her boyuttaki bileşen sayısı

Tablo 1.1: Dizim Bildirgesi

Daha somuta indirgersek,

```
double  puan[4][10][5]
```

bildirimi *puan* adlı, üç boyutlu ve bileşenleri *double* tipten olan bir dizim (array) bildirimidir.

Tek boyutlu arraylerde olduğu gibi, çok boyutlu arrayin her boyutundaki bileşenler sıra numarasıyla indislenir. İndisler 0 dan başlar, o boyuttaki bileşen sayısının 1 eksiğine kadar gider. Bu bildirimde arrayin boyutlarının indisleri,

Birinci boyut indisleri	0,1,2,3
İkinci boyut indisleri	0,1,2,3,4,5,6,7,8,9
Üçüncü boyut indisleri	0,1,2,3,4

Tablo 1.2: İndisler

olur.

Çok boytlu dizimlerin her buyutu bir boyutlu array gibi kullanılabilir. Dolayısyla, konuyu yalınlaştırmak için iki boyutlu dizimlerin kullanımlarına örnekler vermekle yetineceğiz.

İki boyutlu dizimler, günlük yaşamda her zaman karşımıza çıkan tablolardır. Ayrıca bilimsel ve teknik uygulamalarda da onlarla çok sık karşılaşırız. Örneğin, bir sınavda öğrencilerin aldığı puanlar, adlarının karşılarına yazılarak düzenlenen Tablo 1.3 listesi iki boyutlu bir araydir.

Ad	*puanlar*
Can	86
Hacer	97
Yunus	89
Melih	46
Mehmet	54
Emre	39
Batıhan	76
Yaşar	75

Tablo 1.3: İki Boyutlu Dizim

Bu arrayin bildirimi

Liste 1.8.

```
int puan[8][2]
```

olacaktır. Bu bildirimde *Can 86* satırı gibi yatay yazılanlara arrayin satırları, *Ad* ve *puanlar* sütunları gibi düşey olanlara da arrayin kolonları (sütun) denilir. iki boyutlu arrayde ilk boyut [b1] satır sayısını , ikinci boyut [b2] kolon sayısını belirtir.

1.17.2 Çok Boyutlu Arraylerin Öndeğerleri

Çok boyutlu arraylerin öndeğerleri, Kesim 1.9'de açıklanan kurallara uyar.

Çok boyutlu arrayin bileşenleri birer değişken olduğundan, öndeğerleri (default values) değişkenlerin öndeğerleri gibidir. *auto* depo sınıfında olanlar (yerel değişkenler) çöp toplar. *static*, *global* ve *extern* depo sınıfında

olanların öndeğerleri 0 olur. Buna göre, Program 1.18 örneğinde iki boyutlu *arr* arrayi *main()* içinde yerel değişkendir; yani bütün bileşenleri *auto* depo sınıfına aittir. Dolayısıyla, bileşenlere değer atanmazsa, onlar çöp toplar.

Program 1.18.

```
#include <stdio.h>

int main() {
  int i,j;
  int  arr[10][5];

  // arrayi bir tablo halinde yaz
  for (i=0; i < 10; i++) {
    for (j =0; j < 5; j++)
      printf("%d " , arr[i][j]);
    printf("\n");
  }
}
```

Program 1.18 örneğinin aksine, array tanımını *global* yaparsak, bütün bileşenleri *global* depo sınıfına ait olur. Dolayısıyla, bütün bileşenlerin öndeğerleri 0 olur. Böyle olduğunu Program 1.19 örneğindeki arrayin bileşenlerini yazdırarak görebiliriz.

Program 1.19.

```
#include <stdio.h>

int  arr[10][5];
int main() {
  int i,j;

  // arrayi bir tablo halinde yaz
  for (i=0; i < 10; i++) {
    for (j =0; j < 5; j++)
      printf("%d " , arr[i][j]);
    printf("\n");
  }
}
```

```
/**
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
*/
```

1.18 Array'in parametre olarak kullanılması

Arrayler fonksiyonlarda parametre olarak kullanılabilir. Program 1.16 buna basit bir örnektir. Bu konuyu ilerleyen kesimlerde daha ayrıntılı olarak ele alacağız.

1.19 Matris ve Array

İki boyutlu array'lerin bileşenlerinin matris biçiminde yazılışı birer dikdörtgen görüntüsü veriyor. O nedenle, bazı kaynaklar bir ve birden çok boyutlu array'lere *dikdörtgensel array* derler.

m satır ve n kolonu olan $m \times n$ tipinden $A = (a_{m,n})$ matrisi

$$A_{m,n} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}$$

biçiminde bir tablodur.

Matematiksel uygulamalarda matrislerin toplanması, çarpılması gibi işlemlerle çok karşılaşılır. Daha önce de söylediğimiz gibi, kaynak programda $a_{i,j}$ ya da $a^{i,j}$ alt ve üst indislerini yazamayacağımız için $a_{i,j}$ yerine $a[i][j]$ yazacağız. Buna göre 4 satırı ve 5 kolonu olan 4×5 tipi bir matrisi

$$A_{4,5} = \begin{pmatrix} a[0][0] & a[0][1] & a[0][2] & a[0][3] & a[0][4] \\ a[1][0] & a[1][1] & a[1][2] & a[1][3] & a[1][4] \\ a[2][0] & a[2][1] & a[2][2] & a[2][3] & a[2][4] \\ a[3][0] & a[3][1] & a[3][2] & a[3][3] & a[3][4] \end{pmatrix}$$

biçiminde bir tablo ile gösterebiliriz. Bunun bileşenlerine değer atama eylemi, tek boyutlu arraylerde yaptığımız gibidir. Her bileşenine seçkili (random) değer atayabiliriz.

a[0][0]= -3	a[0][1]= 6	a[0][2]= 0	a[0][3]= 6	a[0][4]= 0
a[1][0]= 7	a[1][1]= 9	a[1][2]= 7	a[1][3]= 4	a[1][4]=-4
a[2][0]= 0	a[2][1]= 2	a[2][2]= 3	a[2][3]= 2	a[2][4]= 5
a[3][0]= 2	a[3][1]= 0	a[3][2]= 8	a[3][3]=-2	a[3][4]= 0

Tablo 1.4: Matrisin bileşenlere değer atama

A ve B matrislerinin çarpılabilmesi için A nın kolon sayısı B nin satır sayısına eşit olmalıdır. İki matrisin toplanabilmesi için satır ve kolon sayıları karşılıklı eşit olmalıdır.

Aşağıda 2×2 tipinden iki matrisin çarpımı ve toplamı görülüyor.

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 2 & 6 \\ -1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 6 \\ 2 & 18 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 6 \\ 2 & 18 \end{pmatrix} + \begin{pmatrix} 2 & 6 \\ -1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 12 \\ 1 & 18 \end{pmatrix}$$

Program 1.20 örneği, 3×5 tipi bir matrisin bileşen değerlerinin klavyeden girilmesini sağlıyor.

Program 1.20.

```
#include<stdio.h>
int main() {
  /* 2 boyutlu matris*/
  int a[3][5];

  /*Bileşen değerlerini girdiren döngü*/
  int i, j;

  for(i=0; i<=2; i++) {
    for(j=0; j<=4; j++) {
      printf("Matrisin a[%d][%d] bileseninin degerini giriniz: ", i,
        j);
      scanf("%d", &a[i][j]);
    }
  }
  return 0;
}
```

1.19.1 Matris Çarpımı

Program 1.21, çarpılacak matrislerin satır ve kolon sayısı ile bileşen değerlerinin klavyeden girilmesini sağlıyor ve çarpılabilen matrisler için çarpma işlemini yapıyor.

Program 1.21.

```
include <stdio.h>

int main() {
  int m, n, p, q, c, d, k, toplam = 0;
```

```
  int ilk[10][10], ikinci[10][10], carp[10][10];

  printf("İlk matrisin satır ve kolon sayılarını giriniz \n");
  scanf("%d%d", &m, &n);
  printf("İlk matrisin bileşen değerlerini giriniz\n");

  for (c = 0; c < m; c++)
    for (d = 0; d < n; d++)
      scanf("%d", &ilk[c][d]);

  printf("ikinci matrisin satır ve kolon sayılarını giriniz \n");
  scanf("%d%d", &p, &q);

  if (n != p)
    printf("Bu matrisler çarpılamaz.\n");
  else {
    printf("ikinci matrisin bileşen değerlerini giriniz\n");

    for (c = 0; c < p; c++)
      for (d = 0; d < q; d++)
        scanf("%d", &ikinci[c][d]);

    for (c = 0; c < m; c++) {
      for (d = 0; d < q; d++) {
        for (k = 0; k < p; k++) {
          toplam = toplam + ilk[c][k]*ikinci[k][d];
        }

        carp[c][d] = toplam;
        toplam = 0;
      }
    }

    printf("Matrislerin çarpımı:-\n");

    for (c = 0; c < m; c++) {
      for (d = 0; d < q; d++)
        printf("%d\t", carp[c][d]);

      printf("\n");
    }
  }

  return 0;
}
```

```
/**
ilk matrisin satır ve kolon sayılarını giriniz
2
2
ilk matrisin bileşen değerlerini giriniz
1
2
3
4
ikinci matrisin satır ve kolon sayılarını giriniz
2
2
```

```
ikinci matrisin bileşen de erlerini giriniz
2
6
-1
0
Matrislerin çarpımı :
0       6
2       18
*/
```

1.20 Array'e Eleştiri

Array Yapısının Kısıtları:

Procedural dillerde olduğu gibi, C dili de elbette array yaratmaya ve array üzerinde işlem yapacak metotları tanımlama yeteneğine sahiptir.

Array yapısı, hemen her dilde veri koleksiyonlarını işlemek için vazgeçilemeyen kullanışlı bir yapıdır. Ama bu yapının iki kısıtı vardır:

1. *Array'in boyu değiştirilemez.*

   Yukarıda yaptığımız gibi, array'in boyunu (boyutunu, bileşenlerinin sayısını) ya bildiriminde belirtiriz ya da bileşenlerine ilk değerlerini atayarak arrayin boyutunu belirlemiş oluruz. Her durumda, arrayin boyu başlangıçta kesinkes belirlenmiş olur.

   Ancak, bazı durumlarda, arrayi yarattıktan sonra, array'in bileşen sayısını artırmak ya da azaltmak gerekebilir. Ne var ki, aynı bellek adresinde o iş yapılamaz. O zaman yeni uzunluğa göre yeni bir array tanımlayıp, eski arrayin öğelerini yeni arraye taşımak gerekir. Bu işi yapan fonksiyonları yazmak mümkündür. Ama bu iş yapılırken ana bellekte hem yeni, hem eski array bileşenleri yer alır. Eski arrayin öğelerini yenisine taşımak için fonksiyonlar yazılır. Bütün bunlar, arrayin boyunu değiştirmenin bedelidir. Bu işler, eski array'e ayrılan bellek bölgesinde yapılamadığı için, *array boyu değiştirilemez (immutable)*, denilir. Bunun anlamı, array'in boyunun, ilk tanımlandığı bellek bölgesinde değiştirilemeyeceğidir.

2. *Array'in bileşen değerleri aynı veri tipindendir.*

   Bazı durumlarda farklı veri tiplerinden oluşan veri koleksiyonlarını işlemek gerekebilir. O zaman, bileşenleri aynı veri tipinden olan array yapısı işimize yaramaz. *Java, C#, Python, Ruby* gibi yeni diller farklı veri tiplerinden oluşan koleksiyonlarla iş yapmayı sağlayan yapılar getirmişlerdir.

1.20.1 Array Yerine Başka Yapılar

Prosedural dillerde array yapısı çok önemli bir yapıdır; ondan vazgeçilemez. C dili *array* yapısı, array ile yapılabilecek bütün işlerle ilgili metotları tanımlama yeteneğine sahiptir. Ayrıca, her veri tipindn array kurulabilir. Ancak, array'in değişemez (immutable) oluşu, boyu sık sık değişen arraylerin kullanılmasında programcıya zorluklar çıkarır. Bu sorunu aşmak için, *C Collections Framework (JCF)* (bkz. Kaynak [**?**]) içinde yer alan *Vector, Arraylist, LinkedList, TreeSet* gibi veri yapılarının kullanılması tercih edilir. Tabii, JCF içinde array'den başka yapılara ve başka yapılardan array yapısına dönüşümler yapan metotlar vardır. Özellikle yeni yazılan programlarda *array* yapısı yerine *ArryList* yapısının kullanılması, programcının hayatını çok kolaylaştıracaktır.

1.21 Paramatre Olarak Array Kullanımı

Bir arrayi fonksiyon parametresi olarak kullanmak için, fonksiyonun aşağıdaki üç biçimden birisine benzer olarak tanımlanması gerekir. Başka bir deyişle, fonksiyonlarda array'leri üç farklı biçimde parametre olarak kullanabiliriz. Parametre olarak kullanılacak arrayi *param[ ]* diyelim.

Program 1.22.

```
#include <stdio.h>
disp( char ch)
{
    printf("%c ", ch);
}
int main()
{
    char arr[] = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'I', 'j'};
    int i;
    for (i=0; i<=10; i++)
    {
        /* arrayin her öğesi tek tek disp() fonksiyonuna parametre
         olarak geçiyor*/
        disp (arr[i]);
    }

    return 0;
}

/**
   a  b  c  d  e  f  g  h  i  j
*/
```

1. Parametre olarak pointer kullanmak:

Liste 1.9.

```
void denemeA(int *param)
{
.
.
.
}
```

Array adı pointer olarak kullanılabilir. Pointer konusu Bölüm **??**'de işlenecektir.

2. Uzunluğu belirli array

   Liste 1.10.

```
void denemeA(int param[12])
{
.
.
.
}
```

3. Uzunluğu belirsiz array

   Liste 1.11.

```
void denemeB(int param[])
{
.
.
.
}
```

4. Kaç terim alınacağı belirlenerek

   Liste 1.12.

```
void denemeA(int param[], int uzunluk)
{
.
.
.
}
```

Örnekler:

Daha Az Bileşeni Geçirme Program 1.23 örneğinde, dikkat edilise, arrayin boyunun 10 olduğu, ama *ortalamaBul()* fonksiyonu çağrılırken parametre olarak 5 yazıldığı görülür. Böylece, arrayin bütün terimlerinin değil, ilk beş teriminin aritmetik ortalaması hesaplanır.

Program 1.23.

```
#include <stdio.h>

double ortalamaBul(int arr[], int boy);
int main(){

  int a[] = {13,2,31,24,45,63,777,89,94,10};
  printf("%f"    ,ortalamaBul(a,5));
  return 0;
}

double ortalamaBul(int arr[], int boy)
{
  int     i;
  double ort;
  double toplam;

  for (i = 0; i < boy; ++i)
  {
    toplam += arr[i];
  }

  ort = toplam / boy;

  return ort;
}
```

```
/**
    31.800000
*/
```

Bütün Bileşenleri Geçirme: Program 1.24 örneğinde, arrayin tamamı fonksiyona paramatre olarak geçiyor.

Program 1.24.

```
#include <stdio.h>

/* bildirge (prototype)*/
double getAverage(int arr[], int size);

int main ()
{
   /* 5 bileşenli array tanımı */
   int a[5] = {100, 27, -3, 172, 321};
   double ort;
```

```
    /* Array argümanı olarak pointer geçişi */
    ort = ortalamaBul( a, 5 ) ;

    /* Bulunan ortalamayı yaz */
    printf( "Ortalama değer: %f ", ort );

    return 0;
}
```

```
Ortalama değer. 123.400000
```

Tek Bileşenin Geçmesi Program 1.25 örneğinde, arrayin bir tek bileşeni fonksiyona parametre olarak geçiyor.

Program 1.25.

```
#include <stdio.h>
void display(int a)
   {
   printf("%d",a);
   }
int main(){
   int c[]={2,3,4};
   display(c[2]);  //Passing array element c[2] only.
   return 0;
}
```

```
4
```

Geometrik Ortalama

Program 1.26.

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

double geometrikOrt(double arr[], int);
int main(){


  double a[]= {1,2,3,4,5};
      printf("Geometrik Ortalama: %lf"  ,geometrikOrt(a,5));
}

double geometrikOrt(double arr[], int boy){
int i;
double geoOrt;
    double carpim = 1.0;

    for (i = 0; i < boy; i++)
```

```
    carpim *= arr[i];

    geoOrt = pow(carpim, (1.0/boy));


    return geoOrt;
}

/**
  2.605171
*/
```

1.22 Çok Boyutlu Arraylerin Parametre Olması

Program 1.27.

```
#include <stdio.h>

void goster(int arr[2][2]);
int main(){
   int a[2][2],i,j;
   printf("4 sayi giriniz:\n");
   for(i=0;i<2;++i)
      for(j=0;j<2;++j){
           scanf("%d",&a[i][j]);
      }
      /* çok boyutlu arrayin fonksiyona parametre olarak geçmesi */
   goster(a);
   return 0;
}
void goster(int arr[2][2]){
   int i,j;
   printf("Gösteriliyor:\n");
   for(i=0;i<2;++i)
      for(j=0;j<2;++j)
           printf("\n%d\t"  ,arr[i][j]);
           return;
}
```

```
4 sayı giriniz:
  3  5  7  9

Gösteriliyor:
  3
  5
  7
  9
```

Parametrenin referans olarak geçmesi (Pass by Reference): Program 1.28 örneğinde,

- arrayin adı fonksiyona paramatre olarak geçiyor (pointer)
- Ad, taban adrestir [0-ıncı bileşenin adresi]
- Arrayin bileşen değerleri fonksiyon içinde güncelleniyor
- main() içinde bileşenler görünüyor

Program 1.28.

```
#include<stdio.h>

void dizim(int arr[]){
int i;
for(i=0;i< 5;i++)
 arr[i] = arr[i] + 10;
}

int main()
{
int arr[5],i;

printf("\nArrayin bileşen değerlerini giriniz : ");
for(i=0;i< 5;i++)
 scanf("%d",&arr[i]);

printf("\nBütün arrayi geçir...");
dizim(arr);  // Yalnızca arrayin adı geçsin (pointer)

for(i=0;i< 5;i++)
 printf("\nFonksiyon çağrısından sonra: a[%d] : %d",i,arr[i]);

return 0;
}
```

```
/**

Arrayin bilşen değerlerini giriniz : 1 2 3 4 5

Bütün arrayi geçir...
Fonksiyon çağrısından sonra a[0] : 11
Fonksiyon çağrısından sonra a[1] : 12
Fonksiyon çağrısından sonra a[2] : 13
Fonksiyon çağrısından sonra a[3] : 14
Fonksiyon çağrısından sonra a[4] : 15
*/
```

1.23 Bileşenler Arasında Aktarma

Array'in bileşenlerden başka değişkenlere ya da başka değişkenlerden array'in bileşenlerine veri aktarmalarında istençli (explicit) ve istençsiz (implicit) döküm kuralları aynen geçerlidir (bkz. **??**).

Program 1.29 örneği *short* tipinden *long* tipine istençsiz (implicit) döküm yapılabileceğini göstermektedir.

Program 1.29.

```
#include <stdio.h>

int main() {
  int i,j;
  short a[] = { 8, 7, 6, 5 };
  long b[] = {4567678, 3046732, 32468765, 12765987, 2547321 };

  for (i=0; i <4; i++) {
    b[i] = a[i];       /* veri kaybı oluşmaz */
    printf("b[%d] = a[%d]  \n" , i,i);
  }
  printf("*************\n");
  for (j=0; j <4; j++) {
    printf("\nb[%d] = %d\n" , j, b[j]);
  }
}
```

```
/**
b[0] = a[0]
b[1] = a[1]
b[2] = a[2]
b[3] = a[3]
*************

b[0] = 8

b[1] = 7

b[2] = 6

b[3] = 5
*/
```

Bu dönüşümde, kaynak verilerde bir kayıp oluşmuyor.

Program 1.30 örneği, *long* tip verileri *short* tipe dönüştürüyor. Kaynak değerler ile hedef değerleri karşılaştırdığımızda, çok ciddi hatalar oluştuğu görüyoruz. Bunun nedeni, C derleyicisinin veri denetimi yapmaması ve daha büyük bellek adresinden daha küçük bir bellek adresine aktarma (döküm, cast) yapmamızdır.

Program 1.31 örneği *long* tipten *short* tipe istençsiz döküm (implicit casting) yapıyor ve ciddi veri kayıplarına yol açıyor. Programcı bu tür yanılgılara düşmemelidir.

Program 1.30.

```c
#include <stdio.h>

int main() {
  int i,j;
  short a[] = { 8, 7, 6, 5 };
  long b[] = {4567678, 3046732, 32468765, 12765987, 2547321 };

  for (i=0; i <4; i++) {
    a[i] = b[i];      /* hata: veri kaybı oluşuyor */
    printf("a[%d] = b[%d]  \n" , i,i);
  }
  printf("*************\n");
  for (j=0; j <4; j++) {
    printf("\na[%d] = %d\n" , j, a[j]);
  }
}
```

```
/**
a[0] = b[0]
a[1] = b[1]
a[2] = b[2]
a[3] = b[3]
*************

a[0] = -19842

a[1] = 32076

a[2] = 28445

a[3] = -13533
*/
```

Buradaki veri kaybı ciddi hatalara yol açacaktır. Böyle bir dönüşüm yapılacaksa, programcı meydana gelebilecek veri kaybını göze alarak istemli döküm (casting) yapmalıdır.

1.24 İstençli Döküm

Program 1.31 örneği float tipten int tipe istençli döküm (explicit casting) yapıyor.

Program 1.31.

```c
#include <stdio.h>

int main() {
  int i,j;
  int a[] = { 8, 7, 6, 5 };
  float b[] = {4567.678, 304.6732, 32.468765, 0.12765987 };

  for (i=0; i <4; i++) {
```

```
9       a[i] = (short)b[i];       /* istençli döküm */
        printf("a[%d] = b[%d]   \n" , i,i);
    }
12  printf("*************\n");
    for (j=0; j <4; j++) {
        printf("\na[%d] = %d\n" , j, a[j]);
15  }
}
```

```
/**
a[0] = b[0]
a[1] = b[1]
a[2] = b[2]
a[3] = b[3]
*************

a[0] = 4567

a[1] = 304

a[2] = 32

a[3] = 0
```

Burada sayıların kesir kısımları yok oluyor. Eğer, programda, sayıların kesir kısımlarının atılıp, tamsayı kısımlarının elde edilmesi amaçlanıyorsa, 9.satırda olduğu gibi, istençli döküm *(explicit casting)* yapılabilir.

Program 1.32 örneğinde, iki boyutlu arrayin bileşenlerine içiçe bir döngü ile değer atanıyor ve atanan değerler yazılıyor.

Program 1.32.

```
#include <stdio.h>

//int  arr[5][4];
int main() {
    int i,j;
    int  arr[5][4];
    for (j = 0; j < 5; j++) {   // satır döngüsü
        for ( i = 0; i < 4; i++) { //kolon döngüsü
            arr[j][ i] = i*j ;        // bileşenlere değer atar
            printf("%d\t"  ,  arr[j][i]);
        }
        printf("\n");
    }
}
```

```
/**
0       0       0       0
0       1       2       3
0       2       4       6
0       3       6       9
0       4       8       12

*/
```

1.25 Alıştırmalar

1. Program 1.33 örneği, bir metotla array kurmakta, başka bir metotla arrayin bileşen değerlerini okutmaktadır. Metotların parametreleri kurulan array nesnedir. *arrKur()* metodunun parametresi yaratacağı arraydir. *arrOku()* metodunun parametresi yaratılan arraydir. Her iki fonksiyon, arrayin parametre olarak kullanılışını gösteriyor. Burada array adı *arr* pointer işlevini görüyor.

   Program 1.33.

```
#include <stdio.h>

void arrKur(int p[]);
void arrOku(int r[]);
int arr[10] ;
int n = 10, u;


int main() {
  arrKur(arr);
  arrOku(arr);
  return 0;
}

void arrKur(int p[]) {
  int i;
  for (i = 0; i < 10; i++)
    p[i] = i*3;
}

void arrOku(int r[]) {
  int i;
  for (i = 0; i < 10; i++)
    printf("%d\t" , r[i]);
}
```

```
/**
0   3   6   9   12   15   18   21   24   27
*/
```

2. Bir kurumda çalışanların aylık ücretlerini bir array ile gösteriniz. Sonra ücretlerden kesilen %30 gelir vergisi miktarını başka bir array ile gösteren C programı yazınız.

   Program 1.34.

```
#include <stdio.h>

int main() {
  int i;
```

```
  float aylikUcret[3] ;
  aylikUcret[0] = 3456.76f;
  aylikUcret[1] = 8765.37f;
  aylikUcret[2] = 21347.72f;

  float gelirVergisi[3] ;
  for (i = 0; i < 3; i++) {
    gelirVergisi[i] = aylikUcret[i] * 30 / 100;
    printf("%f    TL ücretin gelir vergisi = %f dir\n" ,
     aylikUcret[i], gelirVergisi[i]);
  }
}
```

```
/**
3456.760010     TL ücretin gelir vergisi = 1037.027954 dir
8765.370117     TL ücretin gelir vergisi = 2629.610840 dir
21347.720703    TL ücretin gelir vergisi = 6404.316406 dir
*/
```

3. Program 1.35 örneğinde arrayin fonksiyona parametre olarak geçişi Liste 1.11'deki gibidir.

Program 1.35.

```
#include <stdio.h>
float ortalama(float a[]);
int main() {
  float ort, c[]= {32.4, 43, 32.6, 4, 41.5, 17};
  ort=ortalama(c);   /* Only name of array is passed as argument
   . */
  printf("Average age=%.2f",ort);
  return 0;
}
float ortalama(float a[]) {
  int i;
  float ort, sum=0.0;
  for(i=0; i<6; ++i) {
    sum+=a[i];
  }
  ort =(sum/6);
  return ort;
}
```

```
  Ortalama yaş :  28.42
```

4. Program 1.36 örneğinde arrayin fonksiyona parametre olarak geçişi 1.12'deki gibidir.

Program 1.36.

```
#include<stdio.h>

void degistir(int b[3]);
```

```
int main(){
int i;
int arr[3] = {1,2,3};
degistir(arr);
for(i = 0;i < 3; i++)
  printf("%d\t", arr[i]);

return 0;
}

void degistir(int a[3])
{
int i;
for(i=0;i<3;i++)
  a[i] = a[i]*a[i];
}
```

```
/**
   1   4   9
*/
```

5. Program **??** örneğinde arrayin fonksiyona parametre olarak geçişi, Liste 1.10'deki gibi yapılıyor.

Program 1.37.

```
#include <stdio.h>

int maksimum( int [] );  /* prototype */

main()
{
  int arr[5], i, max;

  printf("Beş sayi giriniz\n");
  for( i = 0; i < 5; ++i )
    scanf("%d", &arr[i] );

  max = maksimum( arr );
  printf("\nGirdiğiniz sayilarin maksimumu: %d\n", max );
}

int  maksimum( int arr[5] )
{
  int  max_deger, i;

  max_deger = arr[0];
  for( i = 0; i < 5; ++i )
    if( arr[i] > max_deger )
      max_deger = arr[i];

  return max_deger;
}
```

```
/**
   5 sayı sayi giriniz
   34 -56 98 31 45

   Girdiğiniz sayilarin maksimumu: 98
*/
```

6. Bir zar 1000 kez atılıyor. Her yüzün kaç kez geldiğini bulan bir C fonksiyonu yazınız. [Zar gelişi rasgele sayı olacak].

Program 1.38.

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

void zarAt();
int zar[6];
int main() {
  zarAt();
  return 0;
}

void zarAt() {
  int i;
  srand(time(NULL));
  for (i = 0 ; i <= 1000; i++)
    switch(rand()%6+1) {
      case 1:
        zar[1]++;
        break;
      case 2:
        zar[2]++;
        break;
      case 3:
        zar[3]++;
        break;
      case 4:
        zar[4]++;
        break;
      case 5:
        zar[5]++;
        break;
      case 6:
        zar[6]++;
        break;
    }

  for (i = 1; i <= 6; i++)
    printf("\n%d    "   , zar[i]);

  return;
}
```

```
/**
175
180
```

```
170
149
165
161
*/
```

7.

Program 1.39.

```
#define SIZE 100
#include<stdio.h>

float medianBul(float[],int);

int main() {

  int i,n,choice;
  float array[SIZE],mean,median,mode;
  printf("Kaç bileşene değer atayacaksiniz?\n");
  scanf("%d",&n);
  printf("%d float sayi giriz\n"  , n);
  for(i=0; i<n; i++)
    scanf("%f",&array[i]);
  median=medianBul(array,n);
  printf("\n\tMedian = %f\n",median);

}

float medianBul(float a[],int n) {

  float temp;
  int i,j;

  //array artan sıraya konuluyor
  for(i=0; i<n; i++)

    for(j=i+1; j<n; j++) {
      if(a[i]>a[j]) {
        temp=a[j];
        a[j]=a[i];
        a[i]=temp;
      }
    }
  if(n%2==0)
    return (a[n/2]+a[n/2-1])/2;
  else
    return a[n/2];
}
```

```
/**
Kaç bileşene değer atayacaksınız?
    6
    -30004608 float sayi giriz
    23 -43 98 56 12 3 -1

    Median = 17.500000
*/
```

8. Klavyeden girien sayıların modunu bulan bir fonksiyon yazınız. [*Yol gösterme:* Bir sayı kümesinde en çok tekrar edene o kümenin modu denilir. Girilen sayılarıdan bir array oluşturunuz. Arrayin hangi bileşen değerinin en çok tekrar ettiğini bulunuz.]

Program 1.40.

```
#include<stdio.h>

int main() {
  int i,j,k=1,p,a[20],b[20],n,cnt=1,big;
  printf("Kaç sayı gireceksiniz\n");
  scanf("%d",&n);
  printf("%d tane tam sayı giriniz\n"    ,n);
  for(i=1; i<=n; i++)
    scanf("%d",&a[i]);
  for(i=1; i<=n; i++) {
    for(j=i+1; j<=n; j++) {
      if(a[i]==a[j])
        cnt++;
    }
    b[k]=cnt;
    k++;
    cnt=1;
  }

  big=b[1];
  p=1;
  for(i=2; i<=n; i++) {
    if(big<b[i]) {
      big=b[i];
      p=i;
    }
  }
  printf("En çok tekrarlanan sayı: %d\n",a[p]);
  printf("Bu sayı %d kez tekrarlandı\n",b[p]);
  return 0;
}
```

```
/**
   Kaç sayı gireceksiniz
   5
   5 tane tam sayı giriniz
   34 65 7 7 8 13 25
   En çok tekrarlanan sayı: 7
   Bu sayı 2 kez tekrarlandı
*/
```