BLM267

Bölüm 3: Diziler

C Kullanarak Veri Yapıları, İkinci Baskı
Reema Thareja

- giriiş
- Dizilerin Bildirimi
- Dizinin Elemanlarına Erişim
- Değerleri Dizilerde Depolama
- Diziler Üzerindeki İşlemler
- Dizileri Fonksiyonlara Geçirme
- İşaretçiler ve Diziler
- İşaretçi Dizileri
- İki boyutlu diziler
- İki Boyutlu Dizilerde İşlemler
- İki Boyutlu Dizileri Fonksiyonlara Geçirme
- İşaretçiler ve İki Boyutlu Diziler

giriiş

- Dizi, benzer veri öğelerinin bir koleksiyonudur.
- Bu veri elemanlarının veri tipleri aynıdır.
- Dizinin elemanları ardışık bellek konumlarında saklanır ve bir dizinle (ayrıca dizin olarak da bilinir) referans alınır.
- Abonelik, dizinin bir elemanını tanımlamak için kullanılan bir sıra numarasıdır.

Dizilerin Bildirimi

- Her değişkenin kullanılmadan önce tanımlanması gerektiğini daha önce görmüştük.
- Aynı kavram dizi değişkenleri için de geçerlidir. Bir dizi kullanılmadan önce bildirilmelidir.
- Bir diziyi bildirmek, aşağıdakileri belirtmek anlamına gelir:
  - Veri türü — saklayabileceği değer türüdür, örneğin int, char, float, double.
  - Ad — diziyi tanımlamak için.
  - Boyut — dizinin tutabileceği maksimum değer sayısı.
- Diziler aşağıdaki sözdizimini kullanarak tanımlanır:
  tür adı[boyut];
- Veri türü int, float, double, char veya herhangi bir geçerli veri türü olabilir.
- Parantez içindeki sayı dizinin boyutunu, yani dizide saklanabilecek maksimum eleman sayısını gösterir.

Dizilerin Bildirimi

- Örneğin, int marks[10] yazarsak; ifadesi marks'ın 10 eleman içeren bir dizi olduğunu bildirir.
- C'de dizi indeksi sıfırdan başlar.
- İlk öğe marks[0]'da, ikinci öğe marks[1]'de vb. saklanacaktır.
- Bu nedenle son eleman, yani 10. eleman marks[9] içerisinde saklanacaktır.
- Köşeli parantez içinde yazılan 0, 1, 2, 3'ün alt simgeler olduğunu unutmayın. Bellekte, dizi Şekil 3.2'de gösterildiği gibi saklanacaktır.

1st element	2nd element	3rd element	4th element	5th element	6th element	7th element	8th element	9th element	10th element
marks[0]	marks[1]	marks[2]	marks[3]	marks[4]	marks[5]	marks[6]	marks[7]	marks[8]	marks[9]

Figure 3.2 Memory representation of an array of 10 elements

Dizilerin Bildirimi

- **Şekil 3.3 farklı dizi tiplerinin nasıl tanımlandığını göstermektedir.**

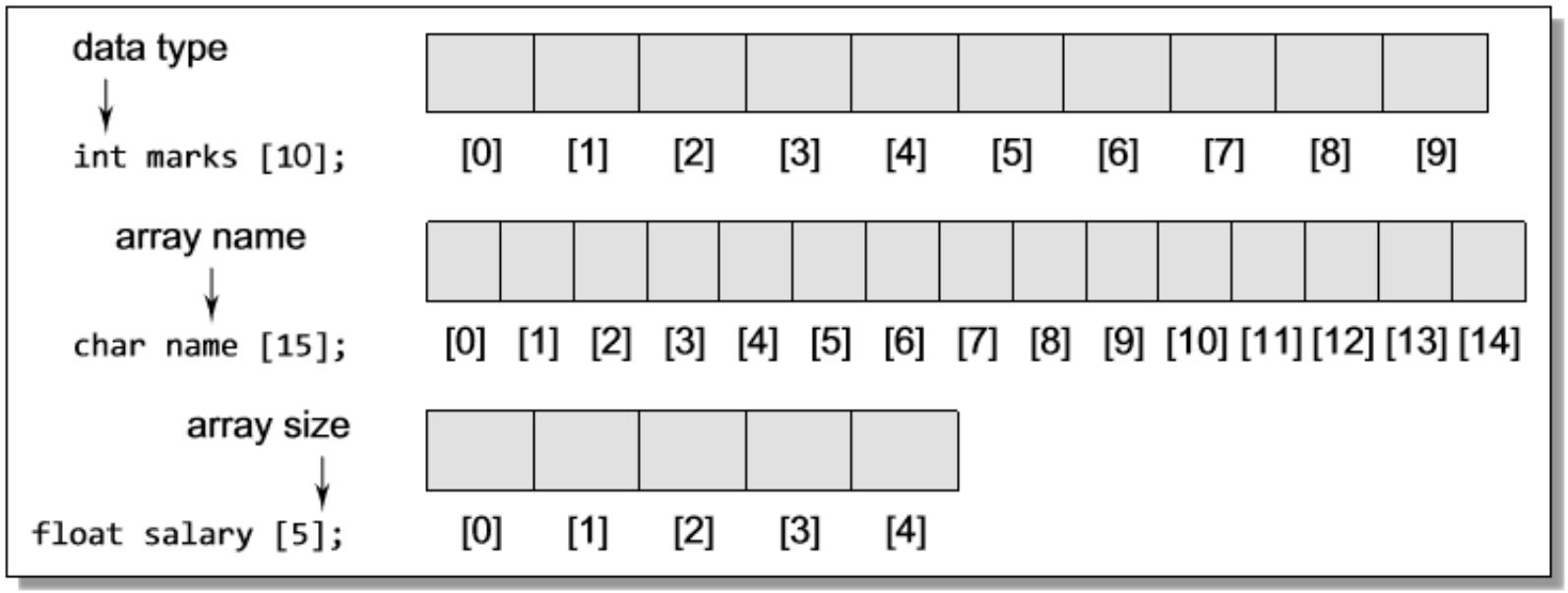


Figure 3.3 Declaring arrays of different data types and sizes

Bir Dizinin Elemanlarına Erişim

- İlgili veri öğelerinin tek bir dizide depolanması, programcıların özlü ve etkili programlar geliştirmesini sağlar.
- Ancak bir dizinin tüm elemanları üzerinde işlem yapabilen tek bir fonksiyon yoktur.
- Tüm elemanlara erişmek için bir döngü kullanmamız gerekiyor.
- Yani dizinin içindeki indis değerini değiştirerek dizinin tüm elemanlarına erişebiliriz.
- Ancak, alt dizinin bir tam sayı değeri veya tam sayı değerini veren bir ifade olması gerektiğini unutmayın.
- Şekil 3.2'de gösterildiği gibi, marks[10] dizisinin ilk elemanına marks[0] yazılarak erişilebilir.

Bir Dizinin Elemanlarına Erişim

- Şimdi dizinin tüm elemanlarını işlemek için Şekil 3.4'te gösterildiği gibi bir döngü kullanırız.
- Şekil 3.5, Şekil 3.4'te gösterilen kodun sonucunu göstermektedir.
- Kod, dizinin her bir öğesine erişir ve değerini -1 olarak ayarlar.
- For döngüsünde, önce marks[0] değeri –1 olarak ayarlanır, ardından indeks (i) değeri artırılır ve bir sonraki değer, yani marks[1] –1 olarak ayarlanır.
- Dizideki 10 elemanın tamamı -1 olana kadar işlem devam eder.

```
// Set each element of the array to -1
int i, marks[10];
for(i=0;i<10;i++)
       marks[i] = -1;
```

Figure 3.4 Code to initialize each element of the array to –1

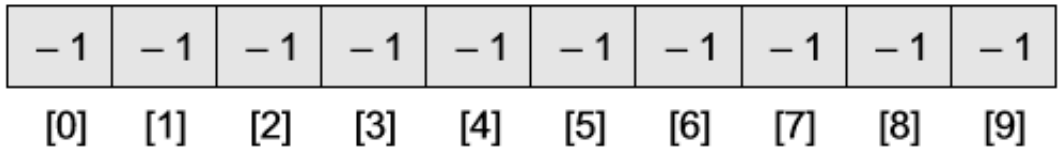


Figure 3.5 Array `marks` after executing the code given in Fig. 3.4

Dizi Elemanlarının Adresinin Hesaplanması

- C'nin bir dizinin herhangi bir elemanının bellekte nerede olduğunu nasıl bildiğini merak ediyor olmalısınız.
- Cevap, dizi adının dizinin ilk baytının adresine sembolik bir referans olmasıdır.
- Dizi adını kullandığımızda aslında dizinin ilk baytına atıfta bulunuyoruz.
- Dizin veya indeks, dizinin başlangıcından başvurulan öğeye kadar olan uzaklığı temsil eder.
- Yani, C sadece dizi adı ve indeksi kullanarak dizideki herhangi bir elemanın adresini hesaplayabilir.
- Bir dizi tüm veri elemanlarını ardışık bellek konumlarında sakladığından, yalnızca taban adresini, yani dizideki ilk elemanın adresini saklamak yeterlidir.

Dizi Elemanlarının Adresinin Hesaplanması

- **Diğer veri elemanlarının adresi, taban adresi kullanılarak kolayca hesaplanabilir.**
- **Bu hesaplamayı gerçekleştirmek için formül şudur: Veri öğesinin adresi, A[k] = BA(A) + w(k – alt_sınır)**
- **Burada A dizi, k adresini hesaplamamız gereken elemanın indeksi, BA A dizisinin taban adresi ve w bellekteki bir elemanın boyutu, örneğin int'in boyutu 2'dir.**

Example 3.1 Given an array `int marks[]={99,67,78,56,88,90,34,85}`, calculate the address of `marks[4]` if the `base address = 1000`.

Solution

99	67	78	56	**88**	90	34	85
marks[0]	marks[1]	marks[2]	marks[3]	**marks[4]**	marks[5]	marks[6]	marks[7]
1000	1002	1004	1006	**1008**	1010	1012	1014

We know that storing an integer value requires 2 bytes, therefore, its size is 2 bytes.

```
marks[4] = 1000 + 2(4 - 0)
         = 1000 + 2(4) = 1008
```

Bir Dizinin Uzunluğunun Hesaplanması

- **Bir dizinin uzunluğu, içinde saklanan eleman sayısıyla belirlenir.**
- **Bir dizinin uzunluğunu hesaplamak için kullanılan genel formül Uzunluk = üst_sınır – alt_sınır + 1'dir. Burada üst_sınır dizideki son elemanın indeksi, alt_sınır ise dizideki ilk elemanın indeksidir.**

Example 3.2 Let `Age[5]` be an array of integers such that

```
Age[0] = 2, Age[1] = 5, Age[2] = 3, Age[3] = 1, Age[4] = 7
```

Show the memory representation of the array and calculate its length.

Solution

The memory representation of the array `Age[5]` is given as below.


```
Length = upper_bound - lower_bound + 1
```

Here, `lower_bound = 0, upper_bound = 4`

Therefore, `length = 4 - 0 + 1 = 5`

DEĞERLERİ DİZİLERDE DEPOLAMA

- Bir diziyi tanımladığımızda, yalnızca elemanları için yer ayırırız; dizide hiçbir değer saklanmaz.
- Bir dizide değerleri saklamanın üç yolu vardır.
- Birincisi, bildirim sırasında dizi elemanlarını başlatmak; ikincisi, klavyeden tek tek elemanlar için değerler girmek; üçüncüsü, tek tek elemanlara değerler atamak.
- Bu durum Şekil 3.6'da gösterilmiştir.

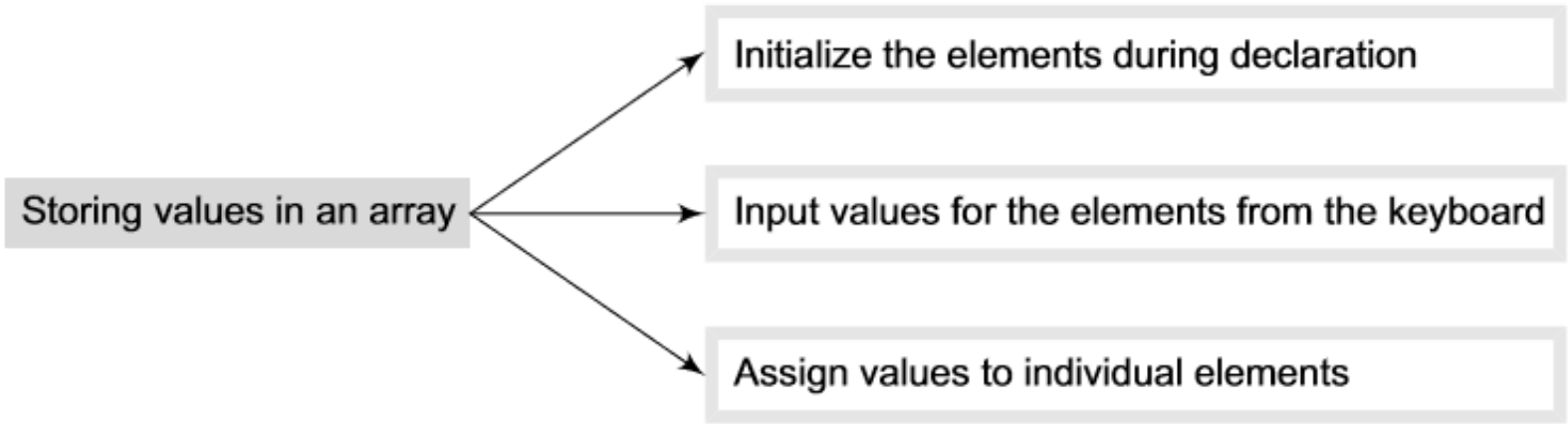


Figure 3.6 Storing values in an array

Bildirim Sırasında Dizileri Başlatma

- Bir dizinin elemanları, diğer değişkenler gibi, bildirim anında başlatılabilir.
- Bir dizi başlatıldığında, dizideki her eleman için bir değer sağlamamız gerekir.
- Diziler, dizi_adı[boyut]={değerler listesi} türünde yazılarak başlatılır;
- Değerlerin süslü parantez içinde yazıldığını ve her değerin virgülle ayrıldığını unutmayın.
- Dizideki eleman sayısından daha fazla değer belirtmek bir derleyici hatasıdır.
- int marks[5]={90, 82, 78, 95, 88}; yazdığımızda; marks adında, beş elemanı saklayacak kadar alana sahip bir dizi bildirilir.
- İlk öğeye, yani marks[0]'a 90 değeri atanır.
- Benzer şekilde, dizinin ikinci elemanı, yani marks[1], 82 olarak atanır, vb. Bu, Şekil 3.7'de gösterilmiştir.

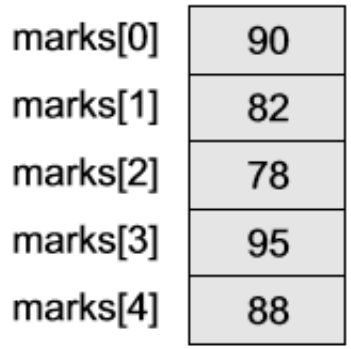


Figure 3.7 Initialization of array `marks[5]`

Bildirim Sırasında Dizileri Başlatma

- Programcı, diziyi bildirim sırasında başlatırken dizinin boyutunu atlayabilir.
- Örneğin, int marks[]= {98, 97, 90}; Yukarıdaki ifade kesinlikle yasaldır.
- Burada derleyici, başlatılan tüm elemanlar için yeterli alanı tahsis edecektir.
- Verilen değerlerin sayısının dizideki eleman sayısından az olması durumunda, atanmamış elemanların sıfırlarla doldurulacağını unutmayın.

r.

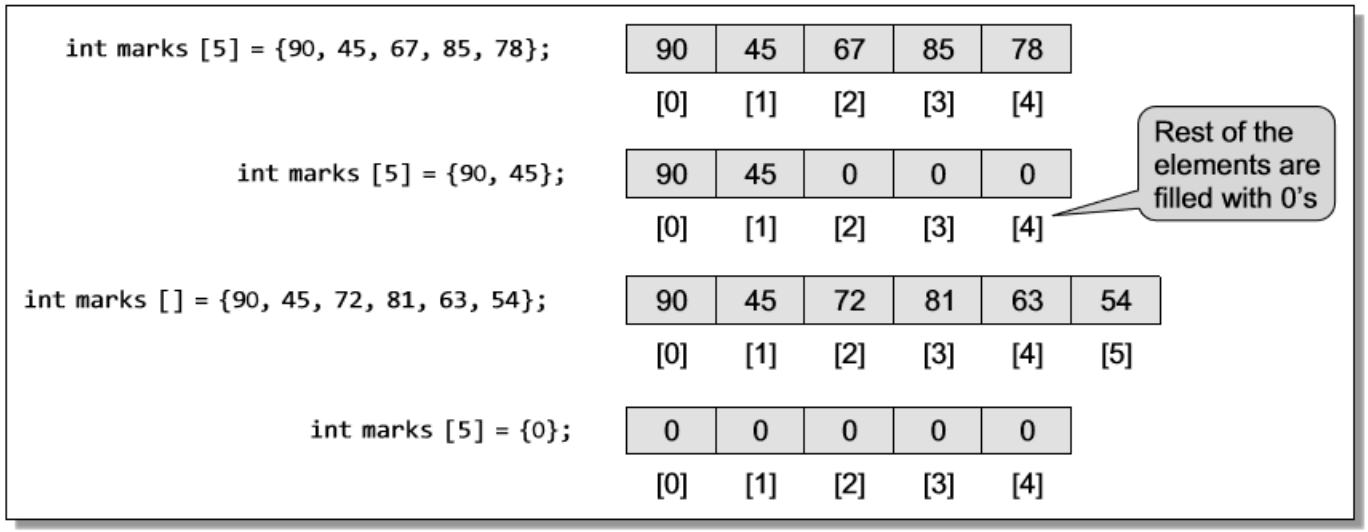


Figure 3.8 Initialization of array elements

Klavyeden Değer Girme

- Bir dizi, klavyeden değerler girilerek başlatılabilir.
- Bu yöntemde, dizinin her bir elemanı için değer girmek üzere while/do–while veya for döngüsü çalıştırılır.
- Örneğin, Şekil 3.9'da gösterilen koda bakın.
- Kodda, 0'daki i indeksinden başlıyoruz ve dizinin ilk elemanının değerini giriyoruz.
- Dizi 10 elemandan oluştuğu için indeksi 0 ile 9 arasında değişen elemanlar için değerler girmeliyiz.

```
int i, marks[10];
for(i=0;i<10;i++)
        scanf("%d", &marks[i]);
```

Figure 3.9 Code for inputting each element of the array

Bireysel Elemanlara Değer Atama

- Üçüncü yol, atama operatörünü kullanarak dizinin bireysel elemanlarına değer atamaktır.
- Dizinin veri türüne denk gelen herhangi bir değer, ayrı dizi elemanlarına atanabilir.
- Basit bir atama ifadesi marks[3] = 100 olarak yazılabilir; Burada, 100, marks[3] olarak belirtilen dizinin dördüncü elemanına atanır.
- İki dizi aynı tipte ve boyutta olsa bile, bir diziyi başka bir diziye atayamayacağımızı unutmayın.
- Bir diziyi kopyalamak için, ilk dizinin her öğesinin değerini ikinci dizinin öğelerine kopyalamanız gerekir. Şekil 3.10, bir diziyi kopyalamak için kullanılan kodu göstermektedir.

```
int i, arr1[10], arr2[10];
arr1[10] = {0,1,2,3,4,5,6,7,8,9};
for(i=0;i<10;i++)
        arr2[i] = arr1[i];
```

Figure 3.10 Code to copy an array at the individual element level

Bireysel Elemanlara Değer Atama

- Şekil 3.10'da döngü, ilk dizinin her bir elemanına erişir ve aynı anda değerini ikinci dizinin karşılık gelen elemanına atar.
- Sıradaki öğeye erişmek için dizin değeri i artırılır. Bu nedenle, bu kod yürütüldüğünde arr2[0] = arr1[0], arr2[1] = arr1[1], arr2[2] = arr1[2], vb.
- Dizi elemanlarına bir değer örüntüsü atamak için bir döngü de kullanabiliriz.
- Örneğin, bir diziyi çift sayılarla (0'dan başlayarak) doldurmak istiyorsak, Şekil 3.11'deki gibi bir kod yazacağız.
- Kodda, her bir elemana indeksinin iki katına eşit bir değer atıyoruz ve indeks 0'dan başlıyor. Bu nedenle, bu kodu çalıştırdıktan sonra arr[0] = 0, arr[1] = 2, arr[2] = 4, vb. olacak.

```
// Fill an array with even numbers
int i,arr[10];
for(i=0;i<10;i++)
        arr[i] = i*2;
```

Figure 3.11 Code for filling an array with even numbers

DİZİLER ÜZERİNDEKİ İŞLEMLER

- Diziler üzerinde gerçekleştirilebilecek bir takım işlemler vardır.
- Bu işlemler şunları içerir:
  - Bir diziyi dolaşma
  - Bir diziye eleman ekleme
  - Bir dizideki bir öğeyi arama
  - Bir diziden bir öğeyi silme
  - İki diziyi birleştirme
  - Bir diziyi artan veya azalan düzende sıralama

- Bu bölümde, 14. Bölümde ele alınacak olan arama ve sıralama hariç, tüm bu işlemleri ayrıntılı olarak ele alacağız.

DİZİLER ÜZERİNDEKİ İŞLEMLER

- **Bir diziyi dolaşma**
- **Bir diziyi dolaşmak, belirli bir amaç için dizinin her bir öğesine erişmek anlamına gelir.**
- **Bir dizi A'nın veri elemanları arasında gezinmek, her elemanı yazdırmayı, toplam eleman sayısını saymayı veya bu elemanlar üzerinde herhangi bir işlem yapmayı içerebilir.**
- **Dizi doğrusal bir veri yapısı olduğundan (çünkü tüm elemanları bir dizi oluşturur), elemanları arasında geçiş yapmak çok basit ve kolaydır.**
- **Dizi geçişi için algoritma Şekil 3.12'de verilmiştir.**

DİZİLER ÜZERİNDEKİ İŞLEMLER

- **Bir diziyi dolaşma**
- **1. Adımda dizinin alt sınırına indeksi başlatıyoruz.**
- **2. Adımda while döngüsü çalıştırılır.**
- **Adım 3, dizi adı ve dizin değeri tarafından belirtilen bireysel dizi öğelerini işler.**
- **4. Adım, bir sonraki dizi öğesinin işlenebilmesi için dizin değerini artırır.**
- **Adım 2'deki while döngüsü, dizideki tüm elemanlar işlenene kadar, yani I, dizinin üst sınırından küçük veya ona eşit olana kadar yürütülür.**

```
Step 1: [INITIALIZATION] SET I = lower_bound
Step 2: Repeat Steps 3 to 4 while I <= upper_bound
Step 3:       Apply Process to A[I]
Step 4:       SET  I = I + 1
        [END OF LOOP]
Step 5: EXIT
```

Figure 3.12 Algorithm for array traversal

DİZİLER ÜZERİNDEKİ İŞLEMLER

```
#include <stdio.h>
#include <conio.h>
int main()
{
        int i, n, arr[20], small, pos;
        clrscr();
        printf("\n Enter the number of elements in the array : ");
        scanf("%d", &n);
        printf("\n Enter the elements : ");
        for(i=0;i<n;i++)
                scanf("%d",&arr[i]);
        small = arr[0]
        pos =0;
        for(i=1;i<n;i++)
        {
                if(arr[i]<small)
                {
                        small = arr[i];
                        pos = i;
                }
        }
        printf("\n The smallest element is : %d", small);
        printf("\n The position of the smallest element in the array is : %d", pos);
        return 0;
```

Output

```
Enter the number of elements in the array : 5
Enter the elements : 7 6 5 14 3

   The smallest element is : 3
   The position of the smallest element in the array is : 4
```

DİZİLER ÜZERİNDEKİ İŞLEMLER

Write a program to enter *n* number of digits. Form a number using these digits.

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
int main()
{
        int number=0, digit[10], numofdigits,i;
        clrscr();
        printf("\n Enter the number of digits : ");
        scanf("%d", &numofdigits);
        for(i=0;i<numofdigits;i++)
        {
                printf("\n Enter the digit at position %d", i+1);
                scanf("%d", &digit[i]);
        }
        i=0;
        while(i<numofdigits)
        {
                number = number + digit[i] * pow(10,i);
                i++;
        }
        printf("\n The number is : %d", number);
        return 0;
}
```

Output

```
Enter the number of digits : 4
Enter the digit at position 1: 2
Enter the digit at position 2 : 3
Enter the digit at position 3 : 0
Enter the digit at position 4 : 9
The number is : 9032
```

DİZİLER ÜZERİNDEKİ İŞLEMLER

Bir diziye eleman ekleme

- Mevcut bir dizinin sonuna bir eleman eklenecekse, ekleme işlemi oldukça basittir.
- Üst sınıra sadece 1 ekleyip değeri atamamız gerekiyor.
- Burada dizi için ayrılan bellek alanının hala kullanılabilir olduğunu varsayıyoruz.
- Örneğin, bir dizi 10 eleman içerecek şekilde tanımlanmışsa, ancak şu anda yalnızca 8 elemana sahipse, o zaman açıkça iki eleman daha barındıracak yer vardır.
- Ama eğer zaten 10 eleman varsa, o zaman ona bir eleman daha ekleyemeyiz.
- Şekil 3.13 bir dizinin sonuna yeni bir eleman eklemek için bir algoritmayı göstermektedir.
- 1. Adımda üst sınırın değerini artırıyoruz.
- 2. Adımda yeni değer, üst sınırın işaret ettiği konuma kaydedilir.
- Örneğin, bir dizinin int marks[60] olarak bildirildiğini varsayalım;

```
Step 1: Set upper_bound = upper_bound + 1
Step 2: Set A[upper_bound] = VAL
Step 3: EXIT
```

Figure 3.13 Algorithm to append a new element to an existing array

DİZİLER ÜZERİNDEKİ İŞLEMLER

- **Bir diziye eleman ekleme**
- Dizi, bir sınıftaki tüm öğrencilerin notlarını saklamak için tanımlanmıştır.
- Şimdi diyelim ki 54 öğrenci var ve yeni bir öğrenci gelip aynı sınava girmesi isteniyor.
- Bu yeni öğrencinin notları, notlar[55]'da saklanacaktır. Öğrencinin 68 not aldığını varsayarak, değeri notlar[55] = 68 olarak atayacağız;
- Ancak dizinin ortasına bir eleman eklememiz gerekiyorsa, bu o kadar da kolay bir iş değildir.
- Ortalama olarak, yeni elemana yer açmak için elemanların yarısını yerlerinden oynatmamız gerekebilir.
- Örneğin, elemanları artan düzende düzenlenmiş bir diziyi ele alalım.
- Şimdi, eğer yeni bir eleman eklenecekse, muhtemelen dizinin ortasına bir yere eklenmesi gerekecektir.
- Bunu yapmak için öncelikle yeni elemanın ekleneceği yeri bulmalı, sonra da yeni elemanın değerinden büyük değere sahip olan tüm elemanları bir konum sağa taşımalıyız ki yeni değeri depolamak için alan yaratılabilsin.

DİZİLER ÜZERİNDEKİ İŞLEMLER

- Bir Dizinin Ortasına Bir Eleman Eklemek İçin Algoritma
- INSERT algoritması INSERT (A, N, POS, VAL) şeklinde tanımlanacaktır.
- Argümanlar şunlardır:
  - (a) A, elemanın eklenmesi gereken dizi
  - (b) N, dizideki eleman sayısı
  - (c) POS, elemanın yerleştirilmesi gereken konum
  - (d) VAL, girilmesi gereken değer
- Şekil 3.14'te verilen algoritmada, Adım 1'de ilk olarak I'yi dizideki toplam eleman sayısı ile başlatıyoruz.

DİZİLER ÜZERİNDEKİ İŞLEMLER

- **Bir Dizinin Ortasına Bir Eleman Eklemek İçin Algoritma**
- **Adım 2'de, POS'tan büyük indekse sahip tüm elemanları yeni eleman için yer açmak amacıyla bir konum sağa doğru taşıyacak bir while döngüsü yürütülür.**
- **Adım 5'te dizideki toplam eleman sayısını 1 artırıyoruz ve son olarak Adım 6'da istenilen pozisyona yeni değer ek**

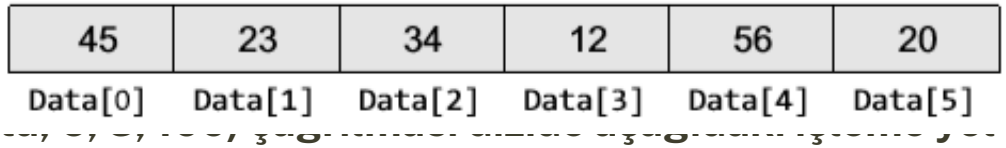
```
Step 1: [INITIALIZATION] SET I = N
Step 2: Repeat Steps 3 and 4 while I >= POS
Step 3:          SET A[I + 1] = A[I]
Step 4:          SET I = I - 1
        [END OF LOOP]
Step 5: SET N = N + 1
Step 6: SET A[POS] = VAL
Step 7: EXIT
```

Figure 3.14 Algorithm to insert an element in the middle of an array.

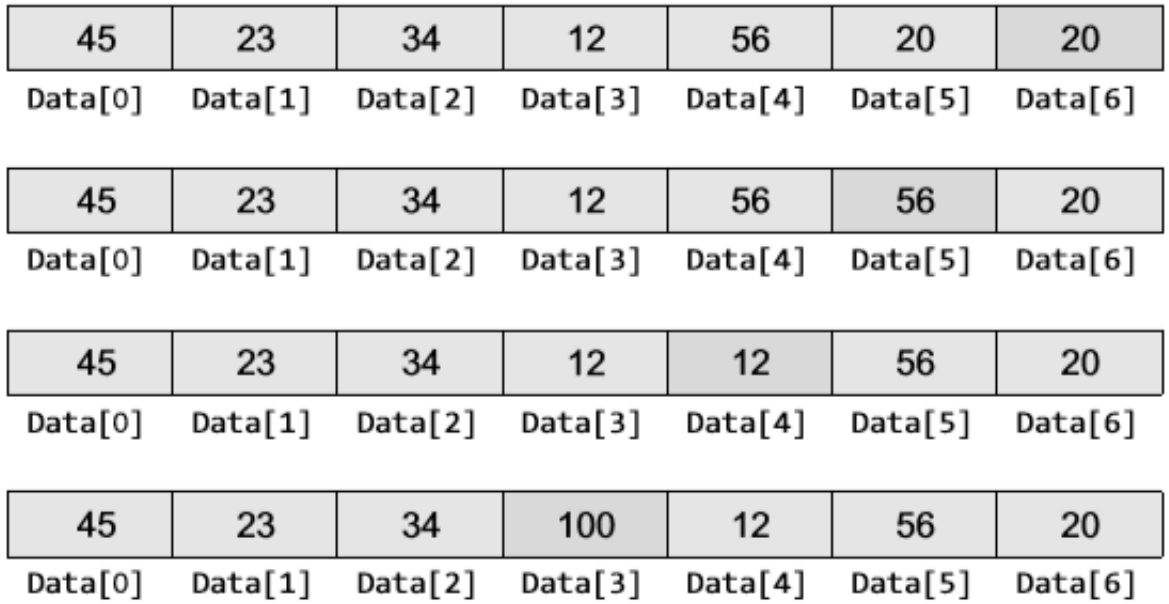
DİZİLER ÜZERİNDEKİ İŞLEMLER

• Bir Dizinin Ortasına Bir Eleman Eklemek İçin Algoritma

- **Şimdi bu algoritmayı bir örnek üzerinden görselleştirelim. Initial Data[] aşağıdaki gibi verilmiştir.**



- **INSERT (Da... ...açacaktır:**



DİZİLER ÜZERİNDEKİ İŞLEMLER

Bir diziden bir öğeyi silme

- Bir diziden bir elemanı silmek, var olan bir diziden bir veri elemanını kaldırmak anlamına gelir.
- Eğer eleman mevcut dizinin sonundan silinecekse, silme işlemi oldukça basittir.
- Üst sınırdan sadece 1'i çıkarmamız gerekiyor.
- Şekil 3.15 bir dizinin sonundan bir elemanı silmek için bir algoritmayı göstermektedir.
- Örneğin, int marks[60] olarak tanımlanmış bir dizimiz varsa; Dizi, sınıftaki tüm öğrencilerin notlarını saklamak için tanımlanmıştır.
- Şimdi, 54 öğrenci olduğunu ve 54 numaralı öğrencinin dersten ayrıldığını varsayalım.
- Bu öğrencinin puanı notlara kaydedildi[54].
- Sadece üst sınırı azaltmamız gerekiyor. Üst sınırdan 1 çıkarmak dizide 53 geçerli veri olduğunu gösterecektir.

```
Step 1: SET upper_bound = upper_bound - 1
Step 2: EXIT
```

Figure 3.15 Algorithm to delete the last element of an array

DİZİLER ÜZERİNDEKİ İŞLEMLER

- Bir diziden bir öğeyi silme
- Ancak bir dizinin ortasından bir elemanı silmemiz gerekiyorsa, bu o kadar da kolay bir iş değildir.
- Silinen öğenin alanını doldurabilmek için ortalama olarak öğelerin yarısını yerlerinden oynatmamız gerekebilir.
- Örneğin, elemanları artan düzende düzenlenmiş bir diziyi ele alalım.
- Şimdi, dizinin ortasından bir elemanın silinmesi gerektiğini varsayalım.
- Bunu yapmak için önce elemanın silineceği yeri bulmalı, sonra da silinen elemanın değerinden büyük değere sahip olan elemanları bir konum sola doğru kaydırmalıyız ki silinen elemanın boşalttığı yer diğer elemanlar tarafından doldurulabilsin.

DİZİLER ÜZERİNDEKİ İŞLEMLER

- Bir Dizinin Ortasından Bir Elemanı Silme Algoritması
- DELETE algoritması DELETE(A, N, POS) olarak bildirilecektir. Argümanlar şunlardır:
  - (a) A, öğenin silinmesi gereken dizi
  - (b) N, dizideki eleman sayısı
  - (c) POS, öğenin silinmesi gereken konum
- Şekil 3.16, öncelikle elemanın silineceği pozisyondan I'yi başlattığımız algoritmayı göstermektedir.
- Adım 2'de, silinen elemanın boşalttığı alanı doldurmak için POS'tan büyük indekse sahip tüm elemanları bir boşluk sola doğru hareket ettirecek bir while döngüsü yürütülür.
- Bir elemanı sildiğimizde aslında ardışık elemanın değerini o elemanın üzerine yazıyoruz.
- 5. Adımda dizideki toplam eleman sayısını 1 azaltıyoruz.
- Şimdi bu algoritmayı Şekil 3.17'de verilen bir örnek üzerinden görselleştirelim.
- DELETE (Data, 6, 2) çağrılması dizide aşağıdaki işleme yol açacaktır.

DİZİLER ÜZERİNDEKİ İŞLEMLER

- **Bir Dizinin Ortasından Bir Elemanı Silme Algoritması**

```
Step 1: [INITIALIZATION] SET I = POS
Step 2: Repeat Steps 3 and 4 while I <= N - 1
Step 3:          SET A[I] = A[I + 1]
Step 4:          SET I = I + 1
        [END OF LOOP]
Step 5: SET N = N - 1
Step 6: EXIT
```

Figure 3.16 Algorithm to delete an element from the middle of an array

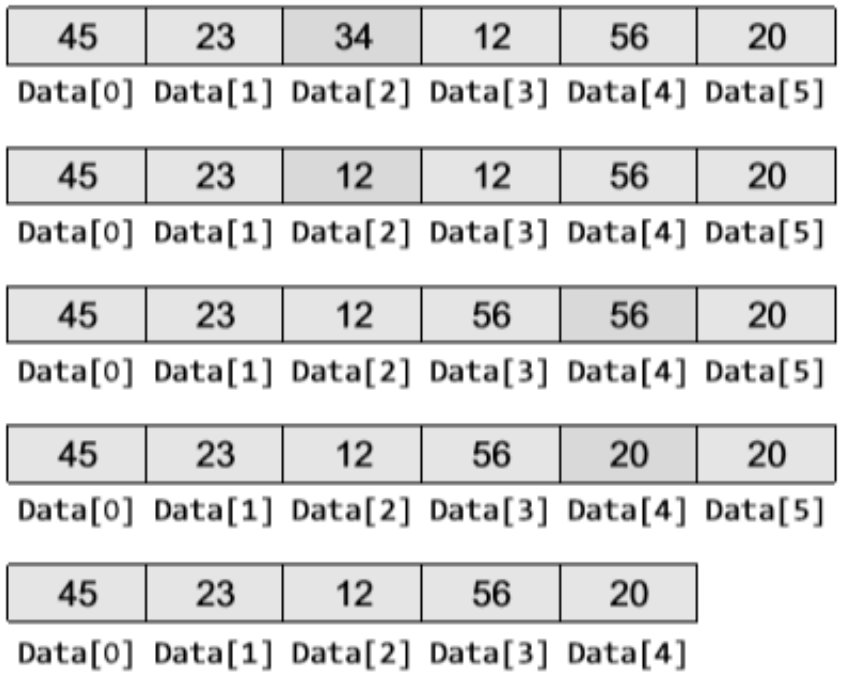


Figure 3.17 Deleting elements from an array

Write a program to delete a number from a given location in an array.

```
#include <stdio.h>
#include <conio.h>
int main()
{
        int i, n, pos, arr[10];
        clrscr();
        printf("\n Enter the number of elements in the array : ");
        scanf("%d", &n);
        for(i=0;i<n;i++)
        {
                printf("\n arr[%d] = ", i);
                scanf("%d", &arr[i]);
        }
        printf("\nEnter the position from which the number has to be deleted : ");
        scanf("%d", &pos);
        for(i=pos; i<n-1;i++)
                arr[i] = arr[i+1];
        n--;
        printf("\n The array after deletion is : ");
        for(i=0;i<n;i++)
                printf("\n arr[%d] = %d", i, arr[i]);
        getch();
        return 0;
}
```

Output

```
Enter the number of elements in the array : 5
arr[0] = 1
arr[1] = 2
arr[2] = 3
arr[3] = 4
arr[4] = 5
Enter the position from which the number has to be deleted : 3
The array after deletion is :
arr[0] = 1
arr[1] = 2
arr[2] = 3
arr[3] = 5
```

DİZİLER ÜZERİNDEKİ İŞLEMLER

İki Diziyi Birleştirme

- İki diziyi üçüncü bir dizide birleştirmek, önce birinci dizinin içeriğini üçüncü diziye kopyalamak ve ardından ikinci dizinin içeriğini üçüncü diziye kopyalamak anlamına gelir.
- Dolayısıyla birleştirilmiş dizi, ilk dizinin içeriğini ve ardından ikinci dizinin içeriğini içerir.
- Eğer diziler sıralanmamışsa, dizileri birleştirmek çok basittir; tek yapmanız gereken bir dizinin içeriğini diğerine kopyalamaktır.
- Ancak iki dizi sıralandığında ve birleştirilen dizinin de sıralanması gerektiğinde birleştirme basit bir iş değildir.
- Öncelikle sıralanmamış dizilerdeki birleştirme işlemini ele alalım. Bu işlem Şekil 3.18'de gösterilmiştir.

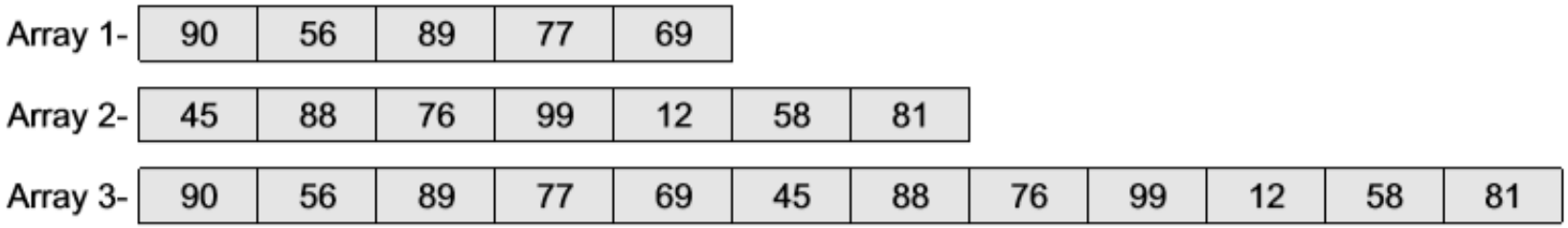


Figure 3.18 Merging of two unsorted arrays

DİZİLER ÜZERİNDEKİ İŞLEMLER

- **İki Diziyi Birleştirme**
- **Eğer iki sıralı dizimiz varsa ve ortaya çıkan birleştirilmiş dizinin de sıralı olması gerekiyorsa, dizileri birleştirme işi biraz zorlaşır.**
- **Birleştirme görevi Şekil 3.19 kullanılarak açıklanabilir.**
- **Şekil 3.19, iki sıralı dizi kullanılarak birleştirilmiş dizinin nasıl oluşturulduğunu göstermektedir.**
- **Burada ilk önce array1'in 1. elemanını array2'nin 1. elemanıyla karşılaştırıyoruz ve daha sonra daha küçük elemanı birleştirilmiş diziye koyuyoruz.**
- **20 > 15 olduğundan, birleştirilen dizinin ilk elemanı olarak 15'i koyduk.**

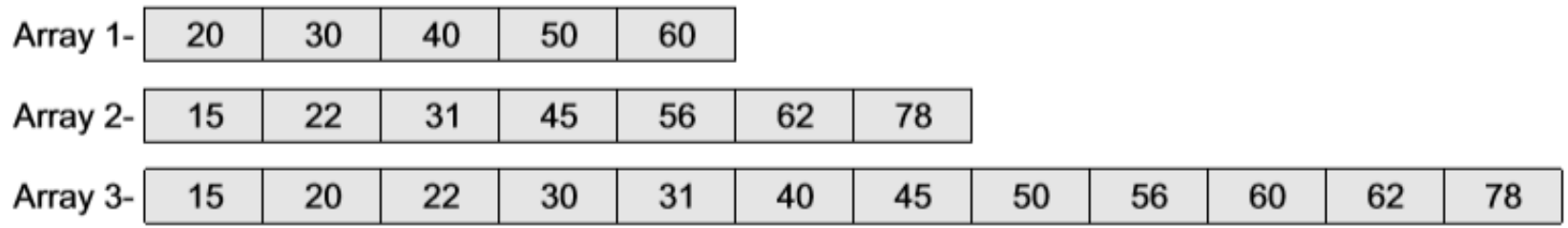


Figure 3.19 Merging of two sorted arrays

DİZİLER ÜZERİNDEKİ İŞLEMLER

- **İki Diziyi Birleştirme**

- **Daha sonra ikinci dizinin 2. elemanını birinci dizinin 1. elemanıyla karşılaştırırız.**
- **20 < 22 olduğundan, artık 20, birleştirilen dizinin ikinci elemanı olarak saklanır.**
- **Daha sonra ilk dizinin 2. elemanı, ikinci dizinin 2. elemanı ile karşılaştırılır.**
- **30 > 22 olduğundan, birleştirilmiş dizinin üçüncü elemanı olarak 22'yi saklıyoruz.**
- **Şimdi birinci dizinin 2. elemanını ikinci dizinin 3. elemanıyla karşılaştıracağız.**
- **30 < 31 olduğundan, 30'u birleştirilmiş dizinin 4. elemanı olarak saklıyoruz. Bu prosedür, her iki dizinin elemanları birleştirilmiş dizide doğru yere yerleştirilene kadar tekrarlanacaktır.**

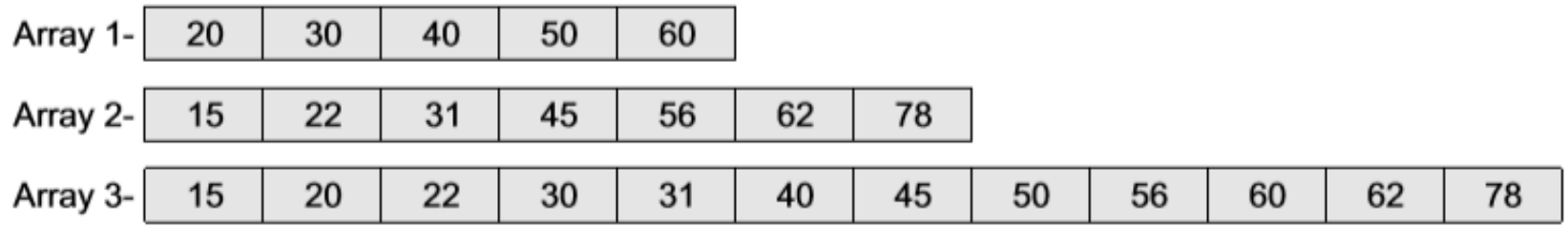


Figure 3.19 Merging of two sorted arrays

DİZİLERİ FONKSİYONLARA AKTARMA

- Diğer veri tiplerindeki değişkenler gibi bir diziyi de bir fonksiyona geçirebiliriz.
- Bazı durumlarda dizinin tek tek elemanlarını geçirmek isteyebilirsiniz; diğer durumlarda ise dizinin tamamını geçirmek isteyebilirsiniz.
- Bu bölümde her iki durumu da ele alacağız.
- Konsepti anlamanıza yardımcı olacak Şekil 3.20'ye bakın.

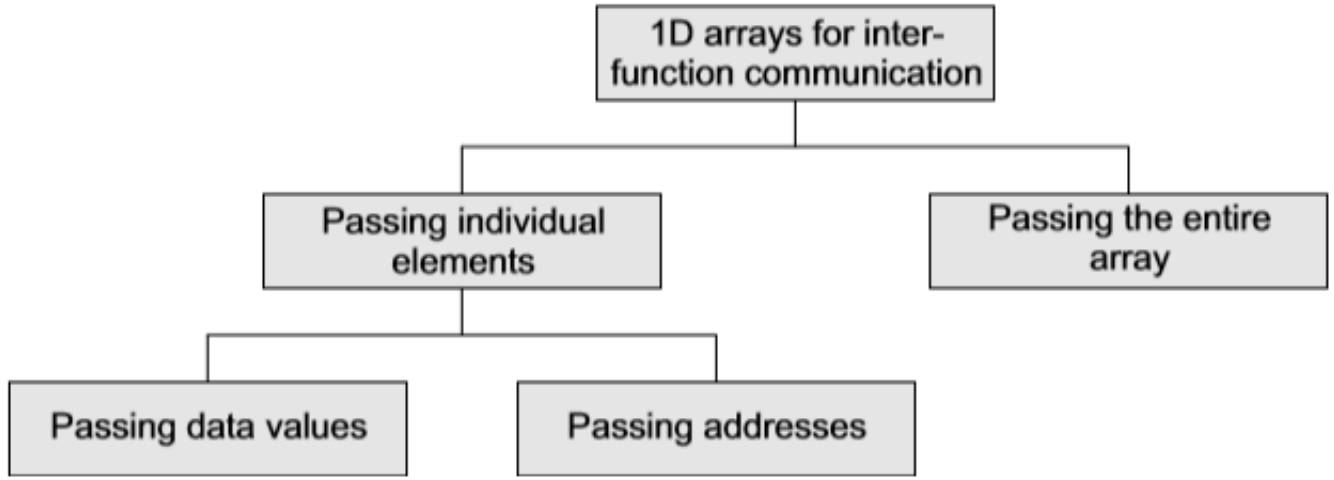


Figure 3.20 One dimensional arrays for inter-function communication

DİZİLERİ FONKSİYONLARA AKTARMA

- **Bireysel elemanların geçişi**
- **Bir dizinin bireysel elemanları, veri değerleri veya adresleri geçirilerek bir fonksiyona geçirilebilir.**
- **Veri Değerlerini Geçirme**
- **Bireysel elemanlar, diğer veri tiplerindeki değişkenleri geçirdiğimiz şekilde aynı şekilde geçirilebilir.**
- **Koşul sadece dizi elemanının veri türünün fonksiyon parametresinin türüyle uyuşmasıdır.**
- **Veri değerini geçirerek bireysel bir dizi elemanını geçirmek için kodu gösteren Şekil 3.21(a)'ya bakın.**

DİZİLERİ FONKSİYONLARA AKTARMA

Calling function

```
main()
{
        int arr[5] ={1, 2, 3, 4, 5};
        func(arr[3]);
}
```

Called function

```
void func(int num)
{
        printf("%d", num);
}
```

Figure 3.21(a) Passing values of individual array elements to a function

- **Veri Değerlerini Geçirme**
- **Yukarıdaki örnekte, çağrılan fonksiyona dizinin yalnızca bir elemanı geçirilir.**
- **Bu, indeks ifadesi kullanılarak yapılır.**
- **Burada, arr[3] tek bir tamsayı değerine değerlendirilir.**
- **Çağrılan fonksiyon, kendisine normal bir tamsayı değişkeni mi yoksa bir dizi değeri mi geçirildiğini bilmez.**

DİZİLERİ FONKSİYONLARA AKTARMA

- Adresleri Geçmek
- Sıradan değişkenler gibi, dizinli dizi öğesinin önüne adres operatörünü ekleyerek tek bir dizi öğesinin adresini geçirebiliriz.
- Bu nedenle, dizinin dördüncü öğesinin adresini çağrılan işleve geçirmek için &arr[3] yazacağız.
- Ancak çağrılan fonksiyonda dizi elemanının değerine dolaylı (*) operatörü kullanılarak erişilmesi gerekmektedir.
- Şekil 3.21(b)'de gösterilen koda bakın.

Calling function

```
main()
{
        int arr[5] ={1, 2, 3, 4, 5};
        func(&arr[3]);
}
```

Called function

```
void func(int *num)
{
        printf("%d", *num);
}
```

Figure 3.21(b) Passing addresses of individual array elements to a function

DİZİLERİ FONKSİYONLARA AKTARMA

- **Tüm Diziyi Geçmek**
- **C'de dizi adının, bellekteki dizinin ilk baytını ifade ettiğini tartışmıştık.**
- **Dizi içerisindeki kalan elemanların adresi, dizi adı ve elemanın indeks değeri kullanılarak hesaplanabilir.**
- **Bu nedenle, bir fonksiyona tüm bir diziyi geçirmemiz gerektiğinde, sadece dizinin adını geçirmemiz yeterli olacaktır.**
- **Şekil 3.22, çağrılan fonksiyona tüm diziyi geçiren kodu**

Calling function

```
main()
{
        int arr[5] ={1, 2, 3, 4, 5};
        func(arr);
}
```

Called function

```
void func(int arr[5])
{
        int i;
        for(i=0;i<5;i++)
                printf("%d", arr[i]);
}
```

Figure 3.22 Passing entire array to a function

DİZİLERİ FONKSİYONLARA AKTARMA

- **Tüm Diziyi Geçmek**
- Bir dizi kabul eden bir fonksiyon, resmi parametreyi aşağıdaki iki yoldan biriyle bildirebilir.
  func(int dizi[]); veya func(int *dizi);
- Bir dizinin ismini bir fonksiyona verdiğimizde, dizinin sıfırıncı elemanının adresi fonksiyondaki yerel işaretçi değişkenine kopyalanır.
- Bir fonksiyon başlığında resmi bir parametre dizi olarak bildirildiğinde, bir dizi olarak değil, bir değişkene işaret eden bir işaretçi olarak yorumlanır.
- Bu işaretçi değişkeni ile dizinin tüm elemanlarına şu ifadeyi kullanarak erişebilirsiniz: dizi_adı + indeks.
- Dizi boyutunu da fonksiyona başka bir parametre olarak geçirebilirsiniz.
- Yani parametre olarak dizi kabul eden bir fonksiyon için bildirim şu şekilde olmalıdır.
  func(int dizi[], int n); veya func(int *dizi, int n);

DİZİLERİ FONKSİYONLARA AKTARMA

- **Tüm Diziyi Geçmek**
- **Dizinin tamamını bir fonksiyona geçirmek gerekli değildir.**
- **Dizinin bir kısmını alt dizi olarak da geçirebiliriz.**
- **Bir alt dizinin işaretçisi aynı zamanda bir dizi işaretçisidir.**
- **Örneğin, diziyi üçüncü elemandan başlayarak göndermek istiyorsak, üçüncü elemanın adresini ve alt dizinin boyutunu geçirebiliriz, yani dizide 10 eleman varsa ve diziyi üçüncü elemandan başlayarak göndermek istiyorsak, o zaman yalnızca sekiz eleman alt dizinin parçası olacaktır.**
- **Bu nedenle fonksiyon çağrısı func(&arr[2], 8); şeklinde yazılabilir.**
- **Çağrılan fonksiyonun dizide hiçbir değişiklik yapmamasını istiyorsak, dizinin çağrılan fonksiyon tarafından sabit dizi olarak alınması gerektiğini unutmamalıyız.**
- **Bu, dizi elemanlarında herhangi bir türde istem dışı değişiklik yapılmasını önler.**
- **Bir diziyi sabit dizi olarak bildirmek için, dizinin veri türünden önce const anahtar sözcüğünü eklemeniz yeterlidir.**

İŞARETÇİLER VE DİZİLER

- Dizi kavramı işaretçi kavramına çok bağlıdır.
- Şekil 3.23'ü ele alalım. Örneğin, int arr[] = {1, 2, 3, 4, 5} şeklinde bildirilen bir dizimiz varsa; bu durumda bellekte Şekil 3.23'te gösterildiği gibi depolanacaktır.
- Dizi gösterimi, işaretçi gösteriminin bir biçimidir. Dizinin adı, dizinin bellekteki başlangıç adresidir.
- Aynı zamanda taban adresi olarak da bilinir.
- Başka bir deyişle, temel adres dizideki ilk elemanın adresi veya arr[0] adresidir.
- Şimdi aşağıdaki ifadede verilen işaretçi değişkenini kullanalım. int *ptr; ptr = &arr[0];
- Burada ptr'nin dizinin ilk elemanını göstermesi sağlanmıştır.

1	2	3	4	5
arr[0]	arr[1]	arr[2]	arr[3]	arr[4]
1000	1002	1004	1006	1008

Figure 3.23 Memory representation of `arr[]`

Programming Tip

The name of an array is actually a pointer that points to the first element of the array.

İŞARETÇİLER VE DİZİLER

- Aşağıda verilen kodu çalıştırıp çıktıyı inceleyin, bu çıktı size konuyu daha açık hale getirecektir.

```
ana()
{
   int dizi[]={1,2,3,4,5};
   printf("\n Dizi adresi = %p %p %p", arr, &arr[0], &arr);
}
```

- Benzer şekilde, ptr = &arr[2] yazmak, ptr'nin dizinin 2 indeksine sahip üçüncü elemanını işaret etmesini sağlar. Şekil 3.24, ptr'nin dizinin üçüncü elemanını i r.

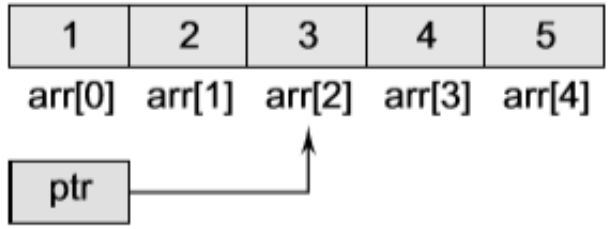


Figure 3.24 Pointer pointing to the third element of the array

Programming Tip

An error is generated if an attempt is made to change the address of the array.

İŞARETÇİLER VE DİZİLER

- Eğer ptr işaretçi değişkeni dizideki ilk elemanın adresini tutuyorsa, ptr++ yazılarak ardışık elemanların adresi hesaplanabilir.

```
int *ptr = &arr[0];
ptr++;
printf("\n Dizinin ikinci elemanının değeri %d'dir",
*ptr);
```

- printf() fonksiyonu 2 değerini yazdıracaktır çünkü ptr artırıldıktan sonra bir sonraki konuma işaret eder.
- Burada dikkat edilmesi gereken bir nokta, eğer x bir tam sayı değişkeni ise x++; x değerine 1 ekler.
- Ancak ptr bir işaretçi değişkenidir, dolayısıyla ptr+i yazdığımızda, i eklediğimizde orijinal işaretçiden daha ilerideki dizideki i elemanını işaret eden bir işaretçi elde ederiz.

İŞARETÇİLER VE DİZİLER

- ++ptr ve ptr++ her ikisi de ptr+1'e eşdeğer olduğundan, tekli ++ operatörünü kullanarak bir işaretçiyi artırmak, depoladığı adresi sizeof(type) tarafından verilen miktar kadar artırır; burada type, işaret ettiği değişkenin veri türüdür (yani, bir tam sayı için 2).
- Örneğin, Şekil 3.25'i ele alalım.
- Eğer ptr başlangıçta arr[2]'yi işaret ediyorsa, ptr++ bir sonraki öğeyi, yani arr[3]'ü işaret edecektir.
- Bu durum Şekil 3.25'te gösterilmiştir.

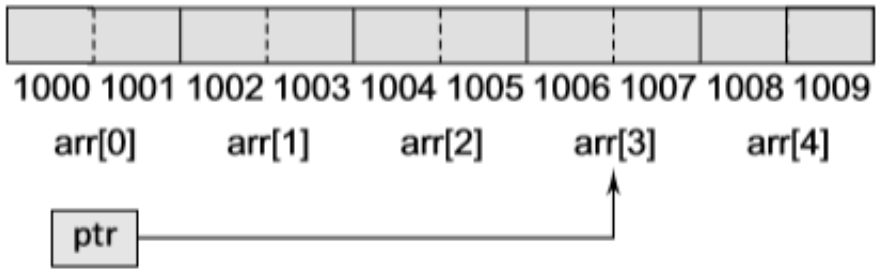


Figure 3.25 Pointer (ptr) pointing to the fourth element of the array

Programming Tip

When an array is passed to a function, we are actually passing a pointer to the function. Therefore, in the function declaration you must declare a pointer to receive the array name.

İŞARETÇİLER VE DİZİLER

- Bu bir karakter dizisi olsaydı, bellekteki her bayt ayrı bir karakteri depolamak için kullanılacaktı. ptr++ daha sonra ptr'nin adresine yalnızca 1 bayt eklerdi.
- İşaretçiler kullanıldığında, arr[i] gibi bir ifade, *(arr+i) yazmaya eşdeğerdir.
- Birçok yeni başlayan, dizi adını bir işaretçi olarak düşünerek kafası karışır.
- Örneğin, şunu yazabiliriz:

  ptr = dizi; // ptr = &dizi[0]

  arr = ptr yazamayız;
- Bunun sebebi ptr'nin değişken, arr'nin ise sabit olmasıdır.

İŞARETÇİLER VE DİZİLER

- arr'nin ilk öğesinin depolanacağı konum, arr[] bildirildikten sonra değiştirilemez.
- Bu nedenle, bir dizi adının genellikle sabit bir işaretçi olduğu bilinir.
- Özetle, bir dizinin adı, ilk elemanının adresine eşdeğerdir; tıpkı bir işaretçinin de işaret ettiği elemanın adresine eşdeğer olması gibi.
- Bu nedenle diziler ve işaretçiler aynı kavramı kullanırlar.

```
i[arr], *(arr+i), *(i+arr) gives the same value.
```

İŞARETÇİLER VE DİZİLER

- Bir dizinin içeriğini bir dizi işaretçisi kullanarak değiştiren aşağıdaki koda bakın.

```
int ana()
{
  int dizi[]={1,2,3,4,5};
  int *ptr, i;
  ptr=&arr[2];
  *ptr = –1;
  *(ptr+1) = 0;
  *(ptr–1) = 1;
  printf("\n Dizi şudur: ");
  i=0;i<5;i++) için
    printf(" %d", *(dizi+i));
  0 döndür;
}
```

Çıktı
Dizi şudur: 1 1 –1 0 5

İŞARETÇİLER VE DİZİLER

- C'de, orijinal konumdan farklı bir yere işaret eden yeni bir işaretçi elde etmek için bir işaretçiye bir tam sayı ekleyebilir veya çıkarabiliriz.
- C ayrıca iki işaretçi değişkeninin toplanmasına ve çıkarılmasına da izin verir.
- Örneğin aşağıda verilen koda bakalım.

```
int ana()
{
   int dizi[]={1,2,3,4,5,6,7,8,9};
   int *ptr1, *ptr2;
   ptr1 = dizi;
   ptr2 = dizi+2;
   printf("%d", ptr2–ptr1);
   0 döndür;
}
```

Çıktı

2

İŞARETÇİLER VE DİZİLER

- Kodda ptr1 ve ptr2 aynı dizinin elemanlarını işaret eden işaretçilerdir.
- Aynı diziye işaret ettikleri sürece iki işaretçiyi çıkarabiliriz.
- Burada çıktı 2'dir çünkü arr dizisinde ptr1 ve ptr2 arasında iki eleman vardır.
- Her iki işaretçi de aynı diziye veya dizinin sonundan sonrasına işaret etmelidir, aksi takdirde bu davranış tanımlanamaz.
- Ayrıca C, işaretçi değişkenlerin birbirleriyle karşılaştırılmasına da olanak tanır.
- Açıkçası, eğer iki işaretçi eşitse, dizide aynı konumu işaret ederler.
- Ancak, bir işaretçi diğerinden küçükse, işaretçi dizinin başlangıcına daha yakın bir öğeyi işaret ediyor demektir.
- Diğer değişkenlerde olduğu gibi, ilişkisel operatörler (>, <, >=, vb.) işaretçi değişkenlere de uygulanabilir.

İŞARETÇİ DİZİLERİ

- Bir işaretçi dizisi şu şekilde bildirilebilir:

```
int *ptr[10];
```

- Yukarıdaki ifade, her biri bir tamsayı değişkenine işaret eden 10 işaretçiden oluşan bir dizi bildirir.
- Örneğin aşağıda verilen koda bakalım.

```
int *ptr[10];
int p = 1, q = 2, r = 3, s = 4, t = 5;
ptr[0] = &p;
ptr[1] = &q;
ptr[2] = &r;
ptr[3] = &s;
ptr[4] = &t;
```

- Aşağıdaki ifadenin çıktısının ne olacağını söyleyebilir misiniz?

```
printf("\n %d", *ptr[3]);
```

İŞARETÇİ DİZİLERİ

- Çıktı 4 olacaktır çünkü ptr[3] tamsayı değişkeni s'nin adresini depolar ve *ptr[3] bu nedenle s'nin değeri olan 4'ü yazdıracaktır.
- Şimdi, işaretçiler dizisinde üç ayrı dizinin adresini sakladığımız başka bir koda bakalım:

```
int ana()
{
   int arr1[]={1,2,3,4,5};
   int arr2[]={0,2,4,6,8};
   int arr3[]={1,3,5,7,9};
   int *parr[3] = {dizi1, dizi2, dizi3};
   int i;
   i = 0;i<3;i++) için
      printf("%d", *parr[i]);
   0 döndür;
}
```

Çıktı 1 0 1

- Bu çıktı sizi şaşırttı mı?
- Kavramı anlamaya çalışın. For döngüsünde, parr[0] arr1'in temel adresini (veya &arr1[0]) depolar.
- Yani *parr[0] yazmak, &arr1[0]'da depolanan değeri yazdıracaktır. Aynı durum *parr[1] ve *parr[2] için de geçerlidir.

İKİ BOYUTLU DİZİLER

- Şimdiye kadar sadece tek boyutlu dizilerden bahsettik.
- Tek boyutlu diziler yalnızca bir yönde doğrusal olarak düzenlenir.
- Ancak bazen verileri grid veya tablolar şeklinde depolamamız gerekir.
- Burada, tek boyutlu diziler kavramı iki boyutlu veri yapılarını da kapsayacak şekilde genişletilmiştir.
- İki boyutlu bir dizi, ilk indisin satırı, ikinci indisin sütunu belirttiği iki indis kullanılarak belirtilir.
- C derleyicisi iki boyutlu bir diziyi tek boyutlu dizilerden oluşan bir dizi olarak ele alır.
- Şekil 3.26, dizilerin dizisi olarak görülebilen iki boyutlu bir diziyi göstermektedir.

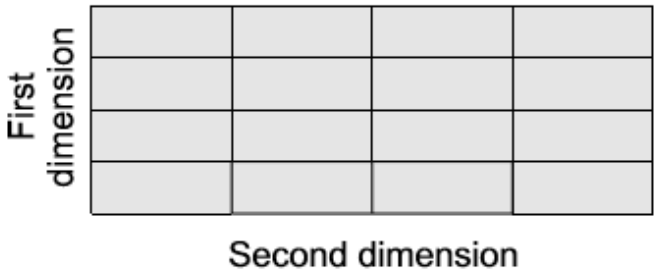


Figure 3.26 Two-dimensional array

İKİ BOYUTLU DİZİLER

- İki boyutlu dizileri bildirme
- Herhangi bir dizi kullanılmadan önce tanımlanmalıdır.
- Bildirim ifadesi derleyiciye dizinin adını, dizideki her bir elemanın veri türünü ve her bir boyutun boyutunu söyler.
- İki boyutlu bir dizi şu şekilde tanımlanır:
- veri_türü dizi_adı[satır_boyutu][sütun_boyutu];
- Bu nedenle, iki boyutlu m X n dizisi, m X n veri öğesi içeren bir dizidir ve her öğeye, i < m ve j < n olmak üzere iki simge, i ve j kullanılarak erişilir.
- Örneğin, üç öğrencinin beş farklı dersten aldığı notları saklamak istiyorsak, iki boyutlu bir diziyi şu şekilde tanımlayabiliriz:
- int işaretleri[3][5];

İKİ BOYUTLU DİZİLER

- İki boyutlu dizileri bildirme
- Yukarıdaki ifadede, m(3) satır ve n(5) sütundan oluşan marks adlı iki boyutlu bir dizi tanımlanmıştır.
- Dizinin ilk elemanı marks[0][0], ikinci elemanı marks[0][1] olarak gösterilir ve bu şekilde devam eder.
- Burada, marks[0][0], ilk öğrencinin ilk derste aldığı notları depolar, marks[1][0], ikinci öğrencinin ilk derste aldığı notları depolar.
- İki boyutlu dizinin resimsel biçimi Şekil 3.27'de gösterilmiştir.

Rows / Columns	Col 0	Col 1	Col 2	Col 3	Col 4
Row 0	marks[0][0]	marks[0][1]	marks[0][2]	marks[0][3]	marks[0][4]
Row 1	marks[1][0]	marks[1][1]	marks[1][2]	marks[1][3]	marks[1][4]
Row 2	marks[2][0]	marks[2][1]	marks[2][2]	marks[2][3]	marks[2][4]

Figure 3.27 Two-dimensional array

İKİ BOYUTLU DİZİLER

- İki boyutlu dizileri bildirme
- Dolayısıyla 2 boyutlu bir dizinin 1 boyutlu dizilerin bir koleksiyonu olarak ele alındığını görüyoruz.
- 2 boyutlu bir dizinin her satırı, n elemandan oluşan 1 boyutlu bir diziye karşılık gelir; burada n sütun sayısını ifade eder.
- Bunu anlamak için Şekil 3.28'de gösterilen iki boyutlu dizinin gösterimine de bakabiliriz.

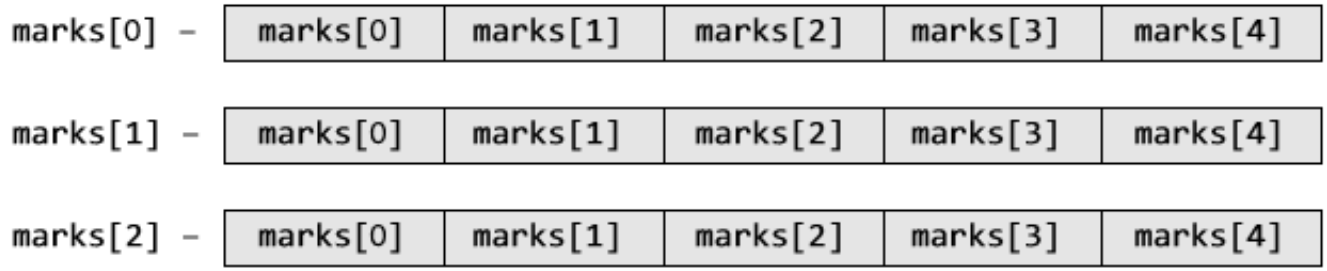


Figure 3.28 Representation of two-dimensional array `marks[3][5]`

İKİ BOYUTLU DİZİLER

- **İki boyutlu dizileri bildirme**
- İki boyutlu bir dizinin dikdörtgen resmini göstermiş olmamıza rağmen, aslında bu elemanlar hafızada sıralı olarak saklanacaktır.
- İki boyutlu bir diziyi hafızaya kaydetmenin iki yolu vardır.
- Birinci yol satır majör sıralaması, ikinci yol ise sütun majör sıralamasıdır.
- 2 boyutlu bir dizinin elemanlarının satır bazında nasıl saklandığını görelim.
- Burada ilk satırın elemanları, ikinci ve üçüncü satırın elemanlarından önce saklanır.
- Yani, dizinin elemanları satır satır saklanır ve ilk satırın n elemanı ilk n konumu işgal eder. Bu, Şekil 3.29'da gösterilmiştir.

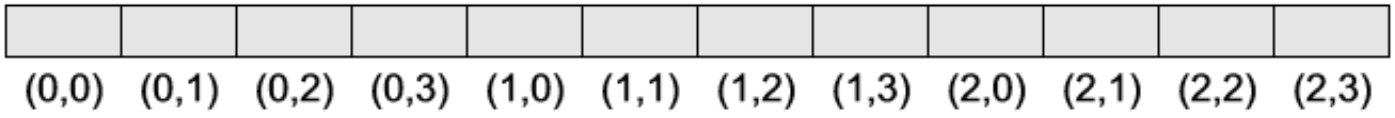


Figure 3.29 Elements of a 3 × 4 2D array in row major order

İKİ BOYUTLU DİZİLER

- İki boyutlu dizileri bildirme
- Ancak elemanları sütun büyük sırasına göre depoladığımızda, birinci sütundaki elemanlar ikinci ve üçüncü sütundaki elemanlardan önce depolanır.
- Yani dizinin elemanları sütun sütun saklanır ve ilk sütunun m elemanı ilk m konumu işgal eder.
- Bu, Şekil 3.30'da gösterilmiştir.

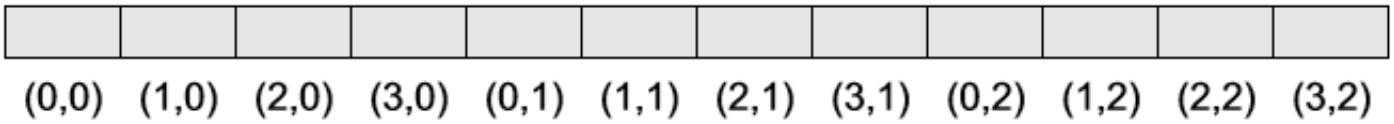


Figure 3.30 Elements of a 4 × 3 2D array in column major order

İKİ BOYUTLU DİZİLER

- **İki boyutlu dizileri bildirme**
- **Tek boyutlu dizilerde bilgisayarın dizideki her elemanın adresini takip etmediğini gördük.**
- **Sadece ilk elemanın adresini saklar ve diğer elemanların adresini taban adresinden (ilk elemanın adresi) hesaplar.**
- **İki boyutlu dizilerde de durum aynıdır.**
- **Burada da bilgisayar taban adresini saklar ve diğer elemanların adresi aşağıdaki formül kullanılarak hesaplanır.**
- **Dizi elemanları sütun ana sırasına göre depolanırsa,**
  Adres(A[i][j]) = Temel_Adres + w{M (j – 1) + (i – 1)}
- **Ve eğer dizi elemanları satır ana sırasına göre depolanırsa,**
  Adres(A[i][j]) = Temel_Adres + w{N (i – 1) + (j – 1)}
  Burada w, bir öğeyi depolamak için gereken bayt sayısıdır, N sütun sayısıdır, M satır sayısıdır ve i ve j dizi öğesinin alt simgeleridir.

İKİ BOYUTLU DİZİLER

- **İki boyutlu dizileri bildirme**

.5 Consider a 20 × 5 two-dimensional array `marks` which has its base addı
:e of an element = 2. Now compute the address of the element, `marks[18][4`
ements are stored in row major order.

```
ss(A[I][J]) = Base_Address + w{N (I - 1) + (J - 1)}
ss(marks[18][4]) = 1000 + 2 {5(18 - 1) + (4 - 1)}
                 = 1000 + 2 {5(17) + 3}
                 = 1000 + 2 (88)
                 = 1000 + 176 = 1176
```

İKİ BOYUTLU DİZİLER

- İki boyutlu dizileri başlatma
- Diğer değişkenlerde olduğu gibi iki boyutlu bir dizi tanımlamak sadece hafızada dizi için yer ayırır.
- İçinde hiçbir değer saklanmaz. İki boyutlu bir dizi, tek boyutlu bir diziyle aynı şekilde başlatılır.
- Örneğin, int marks[2][3]={90, 87, 78, 68, 62, 71};
- İki boyutlu bir dizinin başlatılmasının satır satır yapıldığına dikkat edin.
- Yukarıdaki ifade şu şekilde de yazılabilir:
  int işaretleri[2][3]={{90,87,78},{68, 62, 71}};

İKİ BOYUTLU DİZİLER

- İki boyutlu dizileri başlatma
- Yukarıdaki iki boyutlu dizide iki satır ve üç sütun bulunmaktadır.
- İlk olarak birinci satırdaki elemanlar başlatılır ve ardından ikinci satırdaki elemanlar başlatılır.
- Bu nedenle, puan[0][0] = 90 puan[0][1] = 87 puan[0][2] = 78 puan[1][0] = 68 puan[1][1] = 62 puan[1][2] = 71
- Yukarıdaki örnekte her satır, parantez içinde yer alan üç elemandan oluşan tek boyutlu bir dizi olarak tanımlanmıştır.
- Virgüllerin hem satırdaki elemanları ayırmak hem de iki satırın elemanlarını ayırmak için kullanıldığını unutmayın.

İKİ BOYUTLU DİZİLER

- İki boyutlu dizileri başlatma
- Tek boyutlu dizilerde, eğer dizi tamamen başlatılmışsa, dizinin boyutunu atlayabileceğimizi tartışmıştık.
- Aynı kavram iki boyutlu bir diziye de uygulanabilir, ancak yalnızca ilk boyutun boyutu atlanabilir.
- Bu nedenle aşağıda verilen beyan ifadesi geçerlidir.

  int işaretler[][3]={{90,87,78},{68, 62, 71}};
- Tüm iki boyutlu diziyi sıfırlarla başlatmak için ilk değeri sıfır olarak belirtmeniz yeterlidir.
- Yani, int marks[2][3] = {0};

İKİ BOYUTLU DİZİLER

- İki boyutlu dizileri başlatma
- İki boyutlu bir dizinin bireysel elemanları, burada gösterildiği gibi atama operatörü kullanılarak başlatılabilir.
- işaretler[1][2] = 79;
- veya işaretler[1][2] = işaretler[1][1] + 10;

İKİ BOYUTLU DİZİLER

- İki boyutlu dizinin elemanlarına erişim
- 2 boyutlu dizinin elemanları bitişik bellek konumlarında saklanır.
- Tek boyutlu dizilerde, her geçişte indeksi değiştirmek için tek bir for döngüsü kullandık, böylece tüm elemanlar taranabildi.
- İki boyutlu dizi iki adet abone dizini içerdiğinden, elemanları taramak için iki adet for döngüsü kullanacağız.
- İlk for döngüsü 2 boyutlu dizideki her satırı tarayacak ve ikinci for döngüsü dizideki her satır için ayrı sütunları tarayacaktır.
- 2 boyutlu bir dizinin elemanlarına erişmek için iki for döngüsü kullanan programlara bir bakın.

İKİ BOYUTLU DİZİLER ÜZERİNDEKİ İŞLEMLER

- İki boyutlu diziler, matrislerin matematiksel kavramını uygulamak için kullanılabilir.
- Matematikte matris, satırlar ve sütunlar halinde düzenlenmiş sayılardan oluşan bir ızgaradır.
- Bu nedenle, iki boyutlu dizileri kullanarak m X n matrisi üzerinde aşağıdaki işlemleri gerçekleştirebiliriz:
- *Transpoze m X n matris A'nın transpozesi n X m matris B olarak verilir, burada Bi,j = Aj,i.*
- *Toplam Birbiriyle uyumlu iki matris toplanarak sonuç üçüncü matriste saklanır.*
- İki matrisin satır ve sütun sayıları aynı olduğunda uyumlu oldukları söylenir.
- İki matrisin elemanları şu şekilde yazılarak toplanabilir: Ci,j = Ai,j + Bi,j

İKİ BOYUTLU DİZİLER ÜZERİNDEKİ İŞLEMLER

- ***Fark Birbiriyle uyumlu iki matris çıkarılıp sonuç üçüncü matriste saklanabilir.***
- **İki matrisin aynı sayıda satır ve sütuna sahip olması durumunda uyumlu oldukları söylenir. İki matrisin elemanları şu şekilde yazılarak çıkarılabilir: Ci,j = Ai,j – Bi,j**
- ***Ürün İki matris, birinci matristeki sütun sayısı ikinci matristeki satır sayısına eşitse birbiriyle çarpılabilir. Bu nedenle, n=p ise m X n matrisi A, p X q matrisi B ile çarpılabilir.***
- **Ürün matrisinin boyutu m ¥ q'dur. İki matrisin elemanları şu şekilde yazılarak çarpılabilir: Ci,j = Σ Ai,kBk,j k = 1 ila n için**

İKİ BOYUTLU DİZİLERİ FONKSİYONLARA AKTARMA

- İki boyutlu bir diziyi bir fonksiyona geçirmenin üç yolu vardır.
- Öncelikle dizinin elemanlarını tek tek geçirebiliriz.
- Bu, tek boyutlu bir dizinin bir elemanını geçirmekle tamamen aynı şeydir.
- İkinci olarak, iki boyutlu dizinin tek bir satırını geçebiliriz.
- Bu, tek boyutlu dizinin tamamını daha önceki bir bölümde ele alınan bir fonksiyona geçirmeye eşdeğerdir.
- Üçüncüsü, iki boyutlu dizinin tamamını fonksiyona geçirebiliriz.
- Şekil 3.31, iki boyutlu dizilerin fonksiyonlar arası iletişimde kullanılmasının üç yolunu göstermektedir.

İKİ BOYUTLU DİZİLERİ FONKSİYONLARA AKTARMA

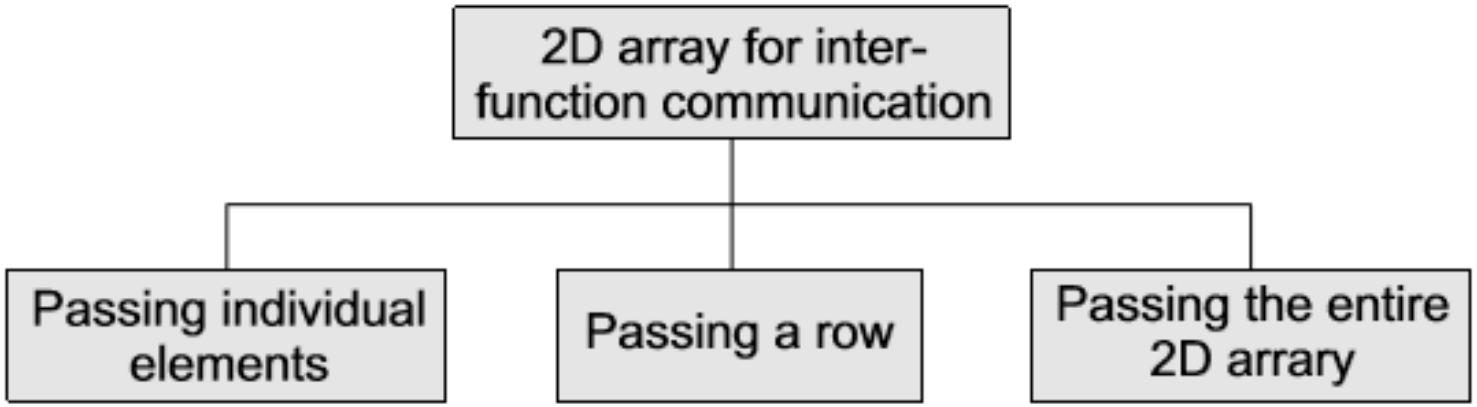


Figure 3.31 2D arrays for inter-function communication

İKİ BOYUTLU DİZİLERİ FONKSİYONLARA AKTARMA

- **Bir Sırayı Geçmek**
- **İki boyutlu bir dizinin bir satırı, dizi adının satır numarasıyla indekslenmesiyle geçirilebilir.**
- **İki boyutlu bir dizinin tek bir satırının çağrılan fonksiyona nasıl geçirilebileceğini gösteren Şekil 3.32'ye bakın.**

Calling function

```
main()
{
      int arr[2][3] = ({1, 2, 3}, {4, 5, 6});
      func(arr[1]);
}
```

Called function

```
void func(int arr[])
{
      int i;
      for(i=0;i<3;i++)
        printf("%d", arr[i] * 10);
}
```

Figure 3.32 Passing a row of a 2D array to a function

İKİ BOYUTLU DİZİLERİ FONKSİYONLARA AKTARMA

- Tüm 2D Dizisini Geçmek
- İki boyutlu bir diziyi bir fonksiyona geçirmek için, gerçek parametre olarak dizi adını kullanırız (1 boyutlu bir dizi durumunda yaptığımız gibi).
- Ancak çağrılan fonksiyondaki parametre dizinin iki boyutlu olduğunu belirtmelidir.

İŞARETÇİLER VE İKİ BOYUTLU DİZİLER

- int mat[5][5] olarak tanımlanan iki boyutlu bir diziyi ele alalım;
- İki boyutlu bir diziye işaretçi bildirmek için int **ptr yazabilirsiniz
- Burada int **ptr, işaretçilerin (tek boyutlu dizilere) bir dizisidir, int mat[5][5] ise 2 boyutlu bir dizidir.
- Bunlar aynı tipte değildir ve birbirlerinin yerine kullanılamazlar.
- mat dizisinin bireysel elemanlarına şunlardan biri kullanılarak erişilebilir: mat[i][j] veya *(*(mat + i) + j) veya *(mat[i]+j);

İŞARETÇİLER VE İKİ BOYUTLU DİZİLER

- İşaretçi kavramını daha iyi anlamak için, *(multi + row) ifadesini X ile değiştirelim; böylece *(*(mat + i) + j) ifadesi *(X + col) olur.
- İşaretçi aritmetiğini kullanarak, X + col + 1'in işaret ettiği adresin (yani değerinin), X + col adresinden sizeof(int)'e eşit bir miktarda büyük olması gerektiğini biliyoruz.
- mat iki boyutlu bir dizi olduğundan, yukarıda kullanılan multi + row ifadesinde multi + row + 1 değerinin bir sonraki satırı işaret etmek için gereken miktarda artması gerektiğini biliyoruz; bu durumda bu miktar COLS * sizeof(int) değerine eşit olacaktır.

İŞARETÇİLER VE İKİ BOYUTLU DİZİLER

- Dolayısıyla, iki boyutlu bir dizi söz konusu olduğunda, ifadeyi değerlendirmek için (satır ana 2 boyutlu bir dizi için), toplam 4 değeri bilmemiz gerekir:
- 1. Dizinin ilk elemanının adresi, dizinin adıyla (örneğin bizim durumumuzda mat) verilir.
- 2. Dizi elemanlarının türünün boyutu, yani bizim durumumuzda tam sayıların boyutu.
- 3. Satırın belirli endeks değeri.
- 4. Sütun için belirli endeks değeri.

İŞARETÇİLER VE İKİ BOYUTLU DİZİLER

- int (*ptr)[10]'a dikkat edin;
- ptr'yi 10 tam sayıdan oluşan bir diziye işaretçi olarak tanımlar.
- Bu, int *ptr[10]'dan farklıdır;
- bu da ptr'yi int türünde 10 işaretçiden oluşan bir dizinin adı yapacaktır.
- Eğer bir dizi işaretçiniz varsa işaretçi aritmetiğinin nasıl çalıştığını merak ediyor olmalısınız.

İŞARETÇİLER VE İKİ BOYUTLU DİZİLER

- Örneğin:
- int * dizi[10] ;
- int ** ptr = dizi ;
- Bu durumda arr'nin tipi int **'dır.
- Tüm işaretçiler aynı boyutta olduğundan, ptr + i'nin adresi şu şekilde hesaplanabilir:

  addr(ptr + i) = addr(ptr) + [int *) * i'nin boyutu]

  = addr(ptr) + [2 * i]
- arr int ** tipinde olduğundan,

  dizi[0] = &dizi[0][0],

  arr[1] = &arr[1][0] ve genel olarak,

  dizi[i] = &dizi[i][0].

İŞARETÇİLER VE İKİ BOYUTLU DİZİLER

- İşaretçi aritmetiğine göre, arr + i = & arr[i], ancak bu 5 öğenin tüm bir satırını atlar, yani, tam 10 baytı atlar (her biri 2 bayt boyutunda 5 öğe). Bu nedenle, arr adres 1000 ise, arr + 1 adres 1010'dur. Özetlemek gerekirse, &arr[0][0], arr[0], arr ve &arr[0] temel adresi işaret eder.
  &arr[0][0] + 1 puan arr[0][1]
  arr[0] + 1 puan arr[0][1]
  arr + 1 puan arr[1][0]
  &arr[0] + 1 puan arr[1][0]
- Sonuç olarak, iki boyutlu bir dizi, 1 boyutlu dizilere ait işaretçilerden oluşan bir dizi ile aynı şey değildir.
- Aslında iki boyutlu bir dizi şu şekilde tanımlanır:
  tamsayı (*ptr)[10] ;
- Burada ptr 10 elemanlı bir dizinin işaretçisidir.
- Parantezler isteğe bağlı değildir.
- Bu parantezlerin yokluğunda, ptr 10 tam sayı dizisine işaretçi değil, 10 işaretçiden oluşan bir dizi haline gelir.

İŞARETÇİLER VE İKİ BOYUTLU DİZİLER

Look at the code given below which illustrates the use of a pointer to a two-dimensional array

```
#include <stdio.h>
int main()
{
        int arr[2][2]={{1,2}, {3,4}};
        int i, (*parr)[2];
        parr = arr;
        for(i = 0; i < 2; i++)
        {
                for(j = 0; j < 2 ;j++)
                        printf(" %d", (*(parr+i))[j]);
        }
        return 0;
}
```

Output

```
1  2  3  4
```

The golden rule to access an element of a two-dimensional array can be given as

```
arr[i][j] = (*(arr+i))[j] = *((*arr+i))+j) = *(arr[i]+j)
```

Therefore,

```
arr[0][0] = *(arr)[0] = *((*arr)+0) = *(arr[0]+0)
arr[1][2] = (*(arr+1))[2] = *((*(arr+1))+2) = *(arr[1]+2)
```

İŞARETÇİLER VE İKİ BOYUTLU DİZİLER

lare an array of pointers using, `ta_type *array_name[SIZE];` represents the number of rows and for columns that can be dynamically	If we declare a pointer to an array `data_type (*array_name)[SIZ` Here `SIZE` represents the nu columns and the space for rows tha dynamically allocated (refer Appe see how memory is dynamically al