

Çok Boyutlu Diziler

Önceki derslerimizde dizileri görmüştük. Kısaca özetleyecek olursak, belirlediğimiz sayıda değişkeni bir sıra içinde tutmamız, diziler sayesinde gerçekleşiyordu. Bu dersimizde, çok boyutlu dizileri inceleyip, ardından dinamik bellek konularına gireceğiz.

Şimdiye kadar gördüğümüz diziler, tek boyutluydu. Bütün elemanları tek boyutlu bir yapıda saklıyorduk. Ancak dizilerin tek boyutlu olması gerekmez; istediğiniz boyutta tanımlayabilirsiniz. Örneğin 3x4 bir matris için 2 boyutlu bir dizi kullanırız. Ya da üç boyutlu Öklid uzayındaki x, y, z noktalarını saklamak için 3 boyutlu bir diziyi tercih ederiz.

Hemen bir başka örnek verelim. 5 kişilik bir öğrenci grubu için 8 adet test uygulansın. Bunların sonuçlarını saklamak için 2 boyutlu bir dizi kullanalım:

```
#include<stdio.h>
int main( )
{
    // 5 adet ogrenci icin 8 adet sinavi
    // temsil etmesi icin bir ogrenci tablosu
    // olusturuyoruz. Bunun icin 5x8 bir matris
    // yaratilmasi gerekiyor.
    int ogrenci_tablosu[ 5 ][ 8 ];
    int i, j;
    for( i = 0; i < 5; i++ ) {
        for( j = 0; j < 8; j++ ) {
            printf( "%d no.'lu ogrencinin ", ( i + 1 ) );
            printf( "%d no.'lu sinavi> ", ( j + 1 ) );
            // Tek boyutlu dizilerdeki gibi deger
            // atiyoruz
            scanf( "%d", &ogrenci_tablosu[ i ][ j ] );
        }
    }
}
```

Bu programı çalıştırıp, öğrencilere çeşitli değerler atadığımızı düşünelim. Bunu görsel bir şekle sokarsak, aşağıdaki gibi bir çizelge oluşur:

		8 Sınav							
5 Öğrenci	1	80	76	58	90	27	60	85	95
	2	60	59	75	80	82	79	64	87
	3	77	...						
	4								
	5				...	67	60	84	

Tabloya bakarsak, 1. öğrenci sınavlardan, 80, 76, 58, 90, 27, 60, 85 ve 95 puan almış gözüküyor.

Ya da 5.öğrencinin, 6.sınavından 67 aldığını anlıyoruz. Benzer şekilde diğer hücelere gerekli değerler atanıp, ilgili öğrencinin sınav notları hafızada tutuluyor.

Çok Boyutlu Dizilere İlk Değer Atama

Çok boyutlu bir diziyi tanımlarken, eleman değerlerini atamak mümkündür. Aşağıdaki örneği inceleyelim:

```
int tablo[3][4] = { 8, 16, 9, 52, 3, 15, 27, 6, 14, 25, 2, 10
};
```

Diziyi tanımlarken, yukardaki gibi bir ilk değer atama yaparsanız, elemanların değeri aşağıdaki gibi olur:

```
Satır 0 : 8 16 9 52
Satır 1 : 3 15 27 6
Satır 2 : 14 25 2 10
```

Çok boyutlu dizilerde ilk değer atama, tek boyutlu dizilerdekiyle aynıdır. Girdiğiniz değerler sırasıyla hücelere atanır. Bunun nedeni de basittir. Bilgisayar, çok boyutlu dizileri sizin gibi düşünmez; dizi elemanlarını hafızada arka arkaya gelen bellek hücreleri olarak değerlendirir.

Çok boyutlu dizilerde ilk değer atama yapacaksanız, değerleri kümelendirmek iyi bir yöntemdir; karmaşıklığı önler. Örneğin yukarıda yazmış olduğumuz ilk değer atama kodunu, aşağıdaki gibi de yazabiliriz:

```
int tablo[3][4] = { 8, 16, 9, 52, 3, 15, 27, 6, 14, 25, 2, 10};
```

Farkedeceğimiz gibi elemanları dörderli üç gruba ayırdık. Bilgisayar açısından bir şey değişmemiş olsa da, kodu okuyacak kişi açısından daha yararlı oldu. Peki ya dört adet olması gereken grubun elemanlarını, üç adet yazsaydık ya da bir-iki grubu hiç yazmasaydık n'olurdu? Deneyelim...

```
int tablo[3][4] = { {8, 16}, {3, 15, 27} };
```

Tek boyutlu dizilerde ilk değer ataması yaparken, eleman sayısından az değer girerseniz, kalan değerler 0 olarak kabul edilir. Aynı şey çok boyutlu diziler için de geçerlidir; olması gerektiği sayıda eleman ya da grup girilmezse, bu değerlerin hepsi 0 olarak kabul edilir. Yani üstte yazdığımız kodun yaratacağı sonuç, şöyle olacaktır:

```
Satır 0 : 8 16 0 0
Satır 1 : 3 15 27 0
Satır 2 : 0 0 0 0
```

Belirtmediğimiz bütün elemanlar 0 değerini almıştır. Satır 2'ninse bütün elemanları direkt 0 olmuştur; çünkü grup tanımı hiç yapılmamıştır.

Fonksiyonlara 2 Boyutlu Dizileri Aktarmak

İki boyutlu bir diziyi fonksiyona parametre göndermek, tek boyutlu diziyi göndermekten farklı sayılmaz. Tek farkı dizinin iki boyutlu olduğunu belirtmemiz ve ikinci boyutun elemanını mutlaka yazmamızdır. Basit bir örnek yapalım; kendisine gönderilen iki boyutlu bir diziyi matris şeklinde yazan bir fonksiyon oluşturalım:

```
#include<stdio.h>
main()
{ int i, j;
  // Ornek olması acisinden matrise keyfi
  // degerler atiyoruz. Matrisimiz 3 satir
  // ve 4 sutundan ( 3 x 4 ) oluyor.
  int matris[][ 4 ] = { {10, 15, 20, 25},
                       {30, 35, 40, 45},
                       {50, 55, 60, 65} };
  for( i = 0; i < 3; i++ ) {
    for( j = 0; j < 4; j++ ) {
      printf( "%d ", matris[ i ][ j ] );
    }
    printf( "\n" );
  }
}
```

Kod içersinde bulunan yorumlar, iki boyutlu dizilerin fonksiyonlara nasıl aktarıldığını göstermeye yetecektir. Yine de bir kez daha tekrar edelim... Fonksiyonu tanımlarken, çok boyutlu dizinin ilk boyutunu yazmak zorunda değilsiniz. Bizim örneğimizde *int dizi[][4]* şeklinde belirtmemiz bundan kaynaklanıyor. Şayet 7 x 6 x 4 boyutlarında dizilerin kullanılacağı bir fonksiyon yazsaydık tanımlamamızı *int dizi[][6][4]* olarak değiştirmemiz gerekirdi. Kısacası fonksiyonu tanımlarken dizi boyutlarına dair ilk değeri yazmamakta serbestsiniz; ancak diğer boyutların yazılması zorunlu! Bunun yararını merak ederseniz, sütun sayısı 4 olan her türlü matrisi bu fonksiyona gönderebileceğinizi hatırlatmak isterim. Yani fonksiyon her boyutta matrisi alabilir, tabii sütun sayısı 4 olduğu sürece...

Soru : Sol aşağıda bulunan 4x4 boyutundaki matrisi saat yönünde 90° döndürecek fonksiyonu yazınız. (Sol matris döndürüldüğü zaman sağ matrise eşit olmalıdır.)

12	34	22	98		38	90	88	12
88	54	67	11		39	91	54	34
90	91	92	93	>>>	40	92	67	22
38	39	40	41		41	93	11	98

Cevap:

```
#include<stdio.h>
void goster( int [][][4] );
void cevir( int [][][4] );
int main( )
{
    int matris[][4] = {
        { 12, 34, 22, 98},
        { 88, 54, 67, 11},
        { 90, 91, 92, 93},
        { 38, 39, 40, 41} };
    goster( matris );
    printf("\n");
    cevir( matris );
}
void goster( int dizi[][4] )
{
    int i, j;
    for( i = 0; i < 4; i++ ) {
        for( j = 0; j < 4; j++ )
```

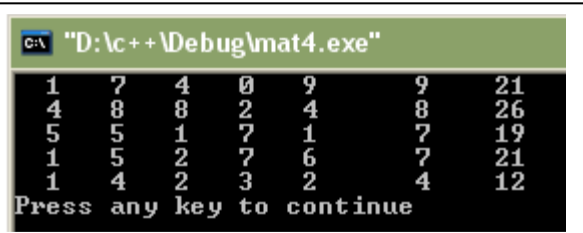
```
        printf( "%d ", dizi[i][j] );
        printf( "\n" );
    }
}
void cevir( int dizi[][4] )
{
    int i, j;
    for( i = 0; i < 4; i++ ) {
        for( j = 0; j < 4; j++ )
            printf( "%d ", dizi[3-j][i]
);
        printf( "\n" );
    }
}
```

örnek matrisin elemanları rasgele üretilsin sütunlarının enbuyugu ve toplamları bulunsun

```
#include <stdlib.h>
#include <stdio.h>
#define SATIR 5
#define SUTUN 5
double enbuyuk(double [][][SUTUN], int );
void matris_oku(double [][][SUTUN]);
main()
{
    double a[SATIR][SUTUN];
    double b[SATIR]; /* satirlardaki en buyuk elemanlar
    double c[SATIR]; /* satirlardaki sutunlarin toplami
    int i, j;
    matris_oku(a);
    /* Satirdaki elemanlarin toplamlarinin olusturdugu
    matris */
    for( i = 0; i < SATIR; i++)
    {
        b[i] = enbuyuk(a,i);
        c[i] = 0;
        for( j = 0; j < SUTUN; j++)
            c[i] = c[i] + a[i][j];
    }
    /* Biçimli yazdırma */
    for( i = 0; i < SATIR; i++) {
        for( j=0; j<SUTUN; j++)
            printf("%3.0f ", a[i][j]);
        printf(" %4.0f %4.0f\n",b[i],c[i]);
    }
```

```
    }
}
double enbuyuk(double a[][SUTUN], int sat)
{
    double r;
    int i;
    r = a[sat][0]; /* ilk eleman en buyuk */
    for( i = 1; i < SUTUN; i++)
        if( a[sat][i] > r)
            r = a[sat][i];
    return r;
} /* Function EnBuyuk */

void matris_oku(double a[][SUTUN])
{
    int i, j;
    for( i = 0; i < SATIR; i++)
        for( j = 0; j < SUTUN; j++)
            a[i][j] = rand()%10;
} /* matris_oku */
```



```
C:\> "D:\c++\Debug\mat4.exe"
1 7 4 0 9 9 21
4 8 8 2 4 8 26
5 5 1 7 1 7 19
1 5 2 7 6 7 21
1 4 2 3 2 4 12
Press any key to continue
```

