3. Diziler ve Pointerler

Bellekte ardışık bir biçimde bulunan ve aynı türden verilerin oluşturduğu veri yapısına dizi (array) denir. Diziler hafızada bir yer kaplarlar. Programcı, her elemanın tipini ve dizide kaç eleman kullanacağını belirterek bilgisayarın en uygun hafızayı ayırmasını sağlar. Dizi boyutu başlangıçta belirtildikten sonra bellekte dizinin eleman sayısı kadar sıralı olarak yer ayrılır. Dizi boyutu daha sonra program içinde çalışma anında tekrar değiştirilmez. Bu yüzden gerekli olduğunda çalışma hatası ile karşılaşmamak için dizi boyutu aşılıp aşılmadığı kontrol edilmelidir. Dizi tanımlaması şu şekilde yapılır.

Veri tipi Dizi ismi [dizi boyutu];

Örnek 10 elemanlı tamsayı türünde A isimli bir dizi tanılmamak istediğimizde aşağıda verilen tanımlamayı yapmak gerekmektedir.

```
int  A[10];

float  x[100], y[50], z[10];  //x 100 elamanlı, y 50 elemanlı   z 10 elemanlı bir dizi

#define  BOYUT 100

int dizi[BOYUT];                // 100 elemanlı bir dizi
```

a[n] şeklinde tanımlanan bir dizinin ilk elemanı a[0] ve son elemanı a[n-1] dir.

Aşağıda verilen for döngüsü dizi içeriğini sıfırlamaktadır.

```
for (i = 0; i < BOYUT; ++i)
    dizi[i]=0; //Dizinin bütün elemanlarını sıfırlar

dizi[5]=78;   // dizinin 5. elemanına 78 değeri atanmaktadır.
```

Dizilere tanımlandıkları anda ilk değerleri de verilebilir.

Örnekler:

```
double sample[5] = {1.3, 2.5, 3.5, 5.8, 6.0};
char str[4] = {'d', 'i', 'z', 'i'};
unsigned[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

Dizilere yukarıdaki gibi ilk değer verildiğinde, verilen değerler dizinin ilk elemanından başlayarak dizi elemanlarına sırayla atanmış olur.

Dizilerin tüm elemanlarına ilk değer verme zorunluluğu yoktur. Dizinin eleman sayısından daha az sayıda elemana ilk değer verilmesi durumunda kalan elemanlara otomatik olarak 0 değeri atanmış olur.

```
int a[20] = {0};  //yalnızca dizinin ilk elemanına 0 değeri vermekle derleyici dizinin kalan elemanlarına otomatik olarak 0 değeri atayacaktır.
```

dizi elemanlarına ilk değer verme işleminde dizi uzunluğu belirtilmeyebilir, bu durumda derleyici dizi uzunluğunu verilen ilk değerleri sayarak kendi hesaplar ve dizinin o uzunlukta açıldığını kabul eder. Örneğin :

int sample[] = {1, 2, 3, 4, 5};

derleyici yukarıdaki ifadeyi gördüğünde sample dizisinin 5 elemanlı olduğunu kabul edecektir.
başka örnekler :

char isim[ ] = {'S', 'a', 'l', 'i', 'h', '\0'};
unsigned short count[ ] = {23, 4, 8, 7, 8, 9, 12, 75, 13, 45};

derleyici isim dizisinin uzunluğunu 6, count dizisinin uzunluğunu ise 10 olarak varsayacaktır.

```
//Dizi kullanımı
#include <iostream.h>
#include <conio.h>

int main( )
{
   int  n[10]={32, 27, 64, 18, 95, 14, 90, 70, 60, 37};
   cout<< "Eleman    Degeri\n";
   for (int i = 0; i <= 9 ;i++)
   cout<< i<<"         "<<n[i]<<endl;
   getch();
  return 0;
}
```

```
Eleman    Degeri
0          32
1          27
2          64
3          18
4          95
5          14
6          90
7          70
8          60
9          37
```

Dizileri Fonksiyonlara giriş olarak göndermek için dizinin ismini parantez kullanmadan belirtilir. Genellikle Diziler fonksiyonlara geçirilirken boyutları da geçirilir.

int myDizi[24];
myFunction(myDizi, 24);

Göstericiler (İşaretçiler-Pointers)

İşaretçi, bellek alanındaki herhangi bir gözün adresinin saklandığı değişkenlerdir. İşaretçiler verileri değil o verilerin bulunduğu yerlerin bellek adreslerini tutarlar. Tüm işaretçilerin bellekte kapladığı alan int boyutundadır. Her değişken bellekte yer kapladığına göre belirli bir adrese sahiptir. Değişkenlerin adresleri, derleyici ve işletim sistemi tarafından ortaklaşa olarak belirlenir. Nesnelerin adresleri program yüklenmeden önce kesin olarak bilinemez ve programcı tarafından da önceden tespit edilemez. Programcı, nesnelerin adreslerini ancak programın çalışması sırasında (run time) öğrenebilir. Örneğin:

```
char ch='A';  //ASCII kodu belleğe yerleşir 65
int b=45;
long f=0x12345678;   //Hexadesimal değer f değişkenine atanıyor
```

biçiminde bir tanımlamayla karşılaşan derleyici bellekte ch değişkeni için 1 byte, b değişkeni için 2 byte ve f değişkeni için 4 byte yer ayıracaktır. Program çalıştırılarak belleğe yüklenmeden derleyicinin ch değişkeni için bellekte hangi adresi ayıracağını önceden bilmek mümkün değildir. Bu durum b ve f değişkenleri içinde geçerlidir.. Bu ancak programın çalışması sırasında öğrenilebilir. Aşağıda verilen çizimde değişkenler için ayrılan adresler ve içerikleri görülmektedir.

	Adres	**İçeriği**
	1A00	?
ch	1A01	65
	1A02	?
b	1A03	45
	1A04	00
	1A05	?
f	1A06	78
	1A07	56
	1A08	34
	1A09	12
	1A0A	?
	1A0B	?
	1A0C	?
	1A0D	?

Göstericiler program içinde tanımlanmış diğer değişkenlerin adreslerini tutan değişkenlerdir. Değişkenlerin adreslerini tutmak ve adreslerle ilgili işlemler yapmak için kullanılır. Göstericilerin içlerinde adres bilgileri bulunur. Bu nedenle gösterici ile adres hemen hemen eş anlamlı olarak düşünülebilir. Tüm göstericilerin bellekte kapladığı alan int boyutundadır

Gösterici bildirimlerinin genel biçimi şöyledir:

*Veri tipi * gösterici ismi;*

Örnek :

```
float *f;   //f değişkeni float türünden bir değişkenin adresini tutacak
```

char *s; //s değişkeni char türünden bir değişkenin adresini tutacak
int *dizi; //dizi değişkeni int türünden bir değişkenin adresini tutacak

göstericilerle ilgili işlem yaparken kullanılan operatörler şu şekildedir.
& ya da adres operatörü, operandının adresini döndüren bir tekli operatördür.
* operatörü hem gösterici tanımlanırken kullanılır. hemde göstericinin tuttuğu(sakladığı, buna gösterdiği de denilebilir) adresin içindeki değere ulaşmak için kullanılır. Bu yüzden içerik operatörü olarak ta söylenmektedir.

Örneğin,
Aşağıda tanımlanan değişkenlerin yukarda verilen şekle göre işlemleri şu şekildedir.

```
char ch='A';  //ASCII kodu belleğe yerleşir 65
int b=45;
long f=0x12345678;   //Hexadesimal değer f değişkenine atanıyor
```

aşağıdaki şekilde tanımlanan pointer değişkenlerle ilgili olarak,

```
char *p1;
int *p2;
long *p3;

p1=&ch;     //p1=1A01  olur
p2=&b;      //p2=1A03  olur
p3=&f;      //p3=1A06 olur
```

işlemleri yapıldıktan sonra

```
char c1;
int  c2;
long c3;

c1=*p1;  //c1=65
c2=*p2;  //c2=45
c3=*p3;  //c3=0x12345678
 olur
```

Gösterici Aritmetiği
Bazen göstericinin işaret etttiği adresten ileri ya da geri gitmek gerekebilir.

```
int *a;
float *b;;
char *c;
```

a++ denildiğinde a nın değeri int tipli bir değişkeni işaret ettiğinden int değişkenin belllekte kapladığı alan (2) kadar artar. Benzer olarak b++ denildiğinde b'nin değeri 4 artacaktır.

```
// Göstericiler * ve & operatörünü kullanmak
#include <iostream.h>
#include <conio.h>

int main()
{
int a;
int *aPtr;   //aPTr bir tamsayı göstericisi
a = 7;
aPtr = &a;   // a değiskeninin adresi aPtr ye atanıyor.

cout << "a nin adresi : " << &a
     << "\n aPtr degeri : " << aPtr;
cout << "\n\n a nin degeri : " << a
     << "\n *aPtr nin degeri " << *aPtr;
cout << "\n\n &*aPtr = " << &*aPtr        // *ve & ikisi birlikte kullanımı
     << "\n *&aPtr = " << *&aPtr << endl;  // yerleri fark etmez
getch();
return 0;
}
```

```
a nin adresi : 0x23ff74
 aPtr degeri : 0x23ff74

 a nin degeri : 7
 *aPtr nin degeri 7

 &*aPtr = 0x23ff74
 *&aPtr = 0x23ff74
```