C dilinde dizi kavramı

C dilinde **dizi (array)**, aynı türdeki birden fazla veriyi bir arada tutmak için kullanılan bir veri yapısıdır. Diziler, bellekte ardışık olarak saklanan birden fazla öğeden oluşur ve bu öğelere indeksler aracılığıyla erişilir. Bir dizi, belirli bir türde (örneğin, int, float, char) sabit bir boyutta tanımlanır ve her bir öğe, dizideki konumuna göre bir indeks numarası ile tanımlanır. Diziler, programlama dilinde verilerin toplu olarak yönetilmesini sağlar ve işlemler üzerinde kolaylık sağlar.

Örneğin, 10 elemandan oluşan bir dizi tanımlandığında, bu dizinin her bir elemanı ardışık bellek hücrelerinde saklanır. Böyle dizi kullanmak yerine, 10 farklı isimle ayrı ayrı değişkenler tanımlamak da mümkündür. Ancak, bu durumda değişkenlerin bellekte ardışık bir şekilde yerleşeceği garanti edilemez. Bellekte rastgele yerleşen ayrı değişkenlerin yönetimi zorlaşabilir ve verilerin erişim süresi uzayabilir. Buna karşın, bir dizi tanımlandığında, dizinin her bir elemanı bellekte sıralı bir şekilde depolanır. Bu ardışık yapı, özellikle diziler üzerinde döngülerle işlem yaparken büyük avantaj sağlar ve belleğin verimli kullanılmasını garanti eder.

Dizilerin Özellikleri

1. **Sabit boyut:** Diziler tanımlandıklarında belirli bir boyuta sahip olurlar ve bu boyut programın çalışma süresi boyunca sabit kalır. Dizi boyutu, bellekte ayrılan yer miktarını belirler.
2. **Aynı türde elemanlar:** Bir dizi, yalnızca belirli bir veri türünden öğeler içerebilir. Örneğin, int türünde bir dizi sadece tamsayılar içerir.
3. **Sıralı bellek:** Dizideki öğeler, bellekte sıralı bir şekilde saklanır. Bu, her öğeye doğrudan indeksle erişimi hızlı ve verimli hale getirir.
4. **Sıfır tabanlı indeksleme:** C dilinde diziler, 0'dan başlayan indekslerle erişilir. İlk öğenin indeksi 0, ikinci öğenin indeksi 1 vb. şeklindedir.
5. **Doğrudan erişim:** Bir dizi elemanına doğrudan indeks numarası ile erişilebilir. Bu, belirli bir öğeye hızlı erişim sağlar.

Dizilerin tanımlanması

Dizi tanımlarken, köşeli parantezler "[ ]" kullanılır. Bu semboller arasındaki veriler dizinin boyutunu ve elemanlarına erişimini belirler. Tek boyutlu diziler, matematiksel olarak vektörlere benzer, yani bir dizi ardışık elemandan oluşur ve bu elemanlara sıralı olarak erişilir. Çift boyutlu diziler ise matematikte matrislere karşılık gelir ve verilerin satır ve sütunlar halinde düzenlenmesini sağlar. Bu sayede, diziler verilerin organize edilmesi ve yönetilmesi için güçlü bir araç haline gelir. Programlama açısından, dizilerle çalışmak, verileri bir bütün olarak ele almayı ve işlemleri daha sistematik bir şekilde gerçekleştirmeyi sağlar.

Bir diziyi oluşturan her bir öğeye **dizi elemanı** adı verilir ve bu elemanların her biri, bellekte belirli bir veri değerini tutar. Bu değere **eleman değeri** denir. Her elemanın dizideki konumu, bir **indis (index) numarası** ile belirlenir. İndisler, dizi elemanlarına erişimi kolaylaştırır ve elemanlar arasında gezinti yapmayı mümkün kılar. Dizi boyutu, içerdiği elemanların sayısıyla tanımlanır ve bu büyüklük dizinin hafızada kapladığı alanı da belirler. Bu nedenle, dizilerin boyutunun önceden belirlenmesi, belleğin verimli kullanımı açısından önemlidir.

Bir diziyi tanımlama formatı/düzeni şu şekildedir:

```
veri tipi  dizi ismi [ eleman sayısı ];
```

- **veri tipi:** Dizide saklanacak elemanların veri türünü belirtir. Örneğin, int, float, char gibi veri türleri olabilir.
- **dizi ismi:** Dizinin adını belirtir. Değişkenlerde olduğu gibi, diziler de tanımlandığında bir isim alır ve bu isim, dizinin bellekteki yerini temsil eder.
- **[eleman sayısı]:** Dizinin kaç eleman saklayacağını belirler. Köşeli parantezler içinde belirtilen bu değer, dizinin boyutunu tanımlar ve bellekte ne kadar yer ayrılacağını belirler. Bir indis her zaman tamsayı olmalıdır.

Farklı türde dizi örnekleri

1. Tamsayı dizisi tanımlama

Aşağıda, 5 elemanlı bir tamsayı dizisi tanımlanmıştır:

```
int sayilar[5];
```

```
unsigned long bakiye[15];
```

Bu tanımlama ile, **sayilar** adında bir dizi oluşturulur ve bellekte 5 adet **int** türünden eleman saklamak için yer ayrılır. Bu elemanlara sayilar[0], sayilar[1], sayilar[2], sayilar[3], ve sayilar[4] şeklinde indeks numaraları ile erişilebilir.

2. Karakter dizisi tanımlama

Karakter dizisi, **char** veri türü ile tanımlanır. Aşağıdaki örnek, 20 karakter uzunluğunda bir dizi tanımlar:

```
char isim[20];
```

Bu örnekte, **isim** adlı dizi, 20 karakter uzunluğunda bir dizi olarak tanımlanmıştır. Genellikle karakter dizileri, bir metni saklamak için kullanılır ve sonuna \0 (null karakter) eklenir.

3. Ondalık sayı dizisi tanımlama

Aşağıda, 10 elemanlı bir **float** türünde ondalık sayı dizisi tanımlanmıştır:

```
float degerler[10];
```

```
double ortalama[15];
```

Bu tanımlama ile degerler adlı bir dizi oluşturulur ve bellekte 10 adet float türünden ondalık sayı saklamak için yer ayrılır.

Dizilerin tanımlanmasındaki önemli noktalar

1. Boyut sabittir

C dilinde bir dizi tanımlandığında, dizinin boyutu tanımlama sırasında belirlenir ve bu boyut program çalıştığı sürece sabit kalır. Yani, bir dizi oluşturulurken belirtilen eleman sayısı sonradan değiştirilemez. Örneğin, int sayilar[10]; şeklinde tanımlanan bir dizide, sadece 10 adet tamsayı saklanabilir. Bu, dizilerin avantajı olduğu kadar, dezavantajı da olabilir. Boyut sabitliği, bellekte ayrılan alanın net olarak belirlenmesini sağlar, bu da bellek yönetimini kolaylaştırır. Ancak, dinamik veri ihtiyaçları durumunda sınırlayıcı olabilir.

2. Sıfır tabanlı indeksleme

C dilinde diziler, sıfır tabanlı indeksleme mantığıyla çalışır, yani dizinin ilk elemanı 0 indeksine sahiptir. Örneğin, int sayilar[5]; şeklinde tanımlanmış bir dizide, ilk eleman sayilar[0], ikinci eleman sayilar[1] olarak adlandırılır. Bu sıfır tabanlı indeksleme, veri işlemlerinde diziler üzerinde döngülerin kullanılması gerektiğinde oldukça pratiktir. Örneğin, bir dizinin tüm elemanlarını yazdırmak için for döngüsü kullanıldığında, for (int i = 0; i < boyut; i++) şeklinde bir yapı tercih edilir. Bu, dizilerin başlangıcından sonuna kadar kolayca işlem yapılmasına olanak tanır.

3. Bellek Kullanımı

Diziler bellekte ardışık olarak saklanır, yani dizideki elemanlar bellekte birbirini izleyen adreslerde yer alır. Bu, belleğin düzenli bir şekilde kullanılmasını ve verilerin daha hızlı erişilmesini sağlar. Ardışık bellek yerleşimi sayesinde, belirli bir elemanın adresi kolayca hesaplanabilir ve bu da veri erişimini hızlandırır. Örneğin, bir int dizisi eleman başına 4 byte kaplar ve sayilar[0]'ın adresi biliniyorsa, sayilar[3]'ün adresi hesaplanabilir: sayilar[0]'ın adresi + (3 * 4 byte).

4. Bellek yönetimi ve dizilerin avantajları

Dizilerde her bir eleman, bellekte ardışık olarak saklandığı için veri işleme sırasında doğrudan bellek erişimi sağlanır. Bu, dizilere belirli bir indeks numarası ile erişimi hızlı hale getirir ve işlemler üzerinde performansı artırır. Ancak, sabit boyutlu oldukları için, dizinin bellekte kaplayacağı yer önceden belirlenmelidir.

Dizilere değer atama

C dilinde dizilere değer atama işlemi bir programın gereksinimlerine ve kullanım amacına göre değişiklik gösterir. Dizilere değer atama işlemi, dizi tanımlama sırasında yapılabildiği gibi, programın çalışması sırasında da dinamik olarak gerçekleştirilebilir. Tanımlama sırasında, başlangıç değerleri küme parantezleri "{ }" içinde belirtilerek dizi elemanlarına atanabilir. Bu yöntem, özellikle sabit bir dizi oluşturulurken ve elemanların başlangıç değerlerinin bilindiği durumlarda idealdir. Ayrıca bir dizinin elemanlarını sıfırlamak, güncellemek veya belirli bir mantıksal işlemle doldurmak için döngülerden faydalanılabilir.

Kullanıcıdan veri alınarak diziye değer atama, etkileşimli programlarda sıklıkla tercih edilir ve programın kullanıcı girdilerine duyarlı olmasını sağlar. Doğru değer atama yöntemlerinin seçimi ve uygulanması, dizilerin daha etkili ve verimli bir şekilde kullanılmasını sağlar. Bu, özellikle karmaşık uygulamalarda bellek yönetiminin optimize edilmesine ve kodun daha okunabilir hale gelmesine katkıda bulunur. Dizilere değer atama işlemi birkaç farklı yöntemle yapılabilir ve her biri farklı ihtiyaçlara uygun kullanım şekilleri sunar.

1. Diziyi tanımlarken değer atama

Dizi tanımlanırken, başlangıç değerleri küme parantezleri "{ }" içinde belirtilir. Bu yöntem, tanımlama sırasında elemanlara otomatik olarak başlangıç değerleri atar.

```c
int sayilar[5] = {10, 20, 30, 40, 50};
```

```c
char mesaj[] = "Merhaba";
```

Bu tanımlamada, **sayilar** dizisinin her bir elemanına sırasıyla 10, 20, 30, 40, ve 50 değerleri atanır. Eğer belirtilen değerler dizinin boyutundan küçükse, geri kalan elemanlara otomatik olarak 0 atanır. Ayrıca dizi boyutu belirtilmeden de başlangıç değerleri atanabilir. Bu durumda, dizi boyutu verilen değerlerin sayısına göre otomatik olarak belirlenir.

```c
int sayilar[] = {5, 10, 15}; // Dizi boyutu otomatik olarak 3 olur.
```

2. Elemanlara tek tek değer atama

Bir dizi tanımlandıktan sonra, elemanlarına tek tek değer atanması, özellikle belirli hesaplamalar ya da işlemler sonucunda elemanların belirlenmesi gereken durumlarda sıklıkla kullanılan bir yöntemdir. Bir dizinin her bir elemanına istenilen değerleri atamak için döngülerin kullanılması büyük bir kolaylık sağlamaktadır. Bu, özellikle eleman değerlerinin belirli bir matematiksel işlemle belirlendiği veya kullanıcı girdisiyle doldurulduğu durumlarda oldukça pratiktir.

```
int sayilar[6];
sayilar[0] = 10;
sayilar[1] = 20;
sayilar[2] = 30;
sayilar[3] = 40;
sayilar[4] = 50;
sayilar[5] = 60;
```

```
char harfler[6];
harfler[0] = 'S';
harfler[1] = 'e';
harfler[2] = 'l';
harfler[3] = 'a';
harfler[4] = 'm';
harfler[5] = '\0';
```

Bu yöntemde her bir eleman

dizi[indis] = değer;

formatıyla atanır.

Eğer bir dizideki eleman sayısı çok fazla olduğu durumlarda yukarıdaki şekilde tek tek değer atama işlemi pek verimli olmaz ve bütün elemanları ayrı ayrı yazmak fazla zaman alacaktır. Bu bakımdan dizilere döngü kullanarak değer atamak daha verimli ve düzenli bir yöntemdir.

```
int sayilar[5];
for (int i = 0; i < 5; i++) {
    sayilar[i] = i * 10;
}
// Eleman değerleri: 0, 10, 20, 30, 40
```

3. Dizinin değerlerini güncelleme

Bir dizinin elemanlarının değerleri programın herhangi bir aşamasında güncellenebilir ve bu işlem program çalışırken dinamik olarak değişen veri ihtiyaçlarını karşılamak için oldukça kullanışlıdır. Örneğin, bir kullanıcıdan alınan girdilere dayanarak dizinin belirli elemanlarını değiştirmek veya algoritmanın gereksinimlerine göre hesaplanan yeni değerleri atamak mümkündür. Ayrıca, güncelleme işlemleri yalnızca tek bir eleman üzerinde değil, aynı zamanda bir döngü yardımıyla birden fazla eleman üzerinde de gerçekleştirilebilir.

```
int sayilar[3] = {10, 20, 30};
sayilar[1] = 25;  // İkinci elemanın değeri 20'den 25'e değişir.
```

```
int deger[20] = {0}; /* Dizinin ilk elemanı 0'dır.
                        Ancak diğer elemanlara da otomatik 0 atanır.*/
int k = 10;
int i = 5;

deger[k++] = 10; /* a dizisinin 10 indisli elemanina yani
                    11. elemanına 100 değeri ataniyor. */

deger[--i] = 200;  /* a dizisinin 4 indisli elemanina yani
                      5. elemanına 200 değeri ataniyor. */
```

```
char mesaj[] = "Merhaba";
mesaj[0] = 'S'; // İlk harfi 'M'den 'S'ye değiştirme. Çıktı Serhaba
```

Bu süreçte dizinin belirli bir elemanına erişmek için indeksleme yöntemi kullanılır ve dizi[indis] = yeni_değer; formatıyla istenilen değişiklik kolaylıkla yapılabilir.

4. Dizilere kullanıcı verisi atama

Kullanıcıdan veri alarak dizilere değer atamak, etkileşimli uygulamaların temel özelliklerinden biri olup, kullanıcı girdilerini programın işleyebilmesi için önemli bir mekanizmadır. Bu yöntem, programın kullanıcıyla dinamik bir etkileşim kurmasını sağlar ve dizi elemanlarının kullanıcı ihtiyaçlarına veya tercihlerine göre doldurulmasına olanak tanır. Örneğin, bir anket uygulamasında kullanıcıdan alınan yanıtların bir dizide saklanması veya bir alışveriş sepetindeki ürün miktarlarının kullanıcı girdilerine göre belirlenmesi gibi durumlarda bu yöntem sıklıkla kullanılır.

```c
#include<stdio.h>

int main() {
    // Ürünler ve sepet miktarlarını saklamak için diziler
    char* urun[] = {"Elma", "Muz", "Portakal", "Üzüm", "Çilek"};
    int miktar[5];
    int adet = 5;

    // Kullanıcıdan her ürün için miktarları alma
    printf("Lütfen alışveriş sepetiniz için ürün miktarlarını giriniz:\n");
    for (int i = 0; i < adet; i++) {
        printf("%s miktarı: ", urun[i]);
        scanf("%d", &miktar[i]);    // Direkt diziye değer atanıyor
    }

    // Sepeti ekrana yazdırma
    printf("\nAlışveriş Sepetiniz:\n");
    for (int i = 0; i < adet; i++) {
        printf("%s: %d adet\n", urun[i], miktar[i]);
    }

    return 0;
}
```

Elemanlara tek tek değer atama - örnek

Kullanıcıdan negatif bir sayı girilene kadar tamsayılar alan, bunları bir dizide saklayan (en fazla 30 sayı) ve girilen bu değerlerin ortalamasını hesaplayan bir programı düşünelim.

```
#include<stdio.h>
int main(void) {
    int sayi[30];         // maksimum eleman sayısı ile dizi tanımlama
    int s=0, toplam=0;
    double ortalama;

    printf("Tamsayi değerleri girin:");
    scanf("%d", & sayi[s]);

    while(sayi[s]>0) {
        toplam = toplam + sayi[s];
        ++s;                          // girilen değer sayacı
        printf("%d. sayı değeri:", s);
        scanf("%d", &sayi[s]);          // Kullanıcıdan değer alma
    }

    ortalama = (double)toplam / s;
    printf("Ortalama=%4.2f", ortalama);
    return 0;
}
```

Bu örnekte bir dizinin elemanlarına **scanf()** fonksiyonuyla kullanıcıdan alınan değerlerin atanması sağlanmıştır. **while()** döngüsü içinde değerler tek tek alınıp diziye aktarılmış ve girilen bir değerin negatif olması halinde girdi işlemi bitirilerek ortalama hesabına geçilmiştir. Toplam ise döngü içinde değerler girilirken yapılmıştır.

Dizilere değer atamada yapılan hatalar

C dilinde dizilerde değer atamaları sırasında sıklıkla karşılaşılan hatalar, genellikle programın çalışmasını olumsuz etkileyebilecek ve beklenmeyen sonuçlara yol açabilecek kritik sorunlardır. Bu hatalar şu şekilde sıralanabilir:

1. Dizi sınırlarını aşma (Out-of-Bounds)
Dizilere atama yaparken, dizi sınırlarının dışında bir indekse erişmeye çalışmak yaygın bir hatadır. C dilinde, dizilerde sınır kontrolü yapılmadığı için bu hata, bellek taşmalarına (buffer overflow) veya diğer bellek alanlarının üzerine yazmaya neden olabilir. Örneğin, 10 elemanlı bir diziye 11. elemanı atamak programın davranışını belirsiz hale getirir. Örnek int dizi[5]; dizi[5] = 10; // Dizi sınırlarının dışında

2. Doğru tipte değer atamama
Dizi elemanlarının türü ile uyumsuz bir veri türü atanması, beklenmeyen sonuçlara neden olabilir. Örneğin, **int** türündeki bir diziye bir **float** değeri atamak veri kaybına veya dönüşüm hatalarına yol açabilir. Örnek int dizi[3]; dizi[0] = 3.14; // Veri kaybı: float değeri int'e dönüşür

3. Başlangıç değeri vermeden değerlere erişim
Dizilere değer atanmadan önce bu değerlere erişmeye çalışmak, rastgele bellek verilerine erişilmesine ve programın hatalı davranmasına neden olur. Örnek: int dizi[5]; printf("%d\n", dizi[0]); // Dizi elemanlarına başlangıç değeri verilmemiş

4. Diziyi yanlış boyutlandırma
Diziye atanacak eleman sayısı belirtilen boyuttan büyükse, belleğin üzerine yazılmasına neden olabilir. Bu, özellikle sabit bir başlangıç değeri kümesiyle dizi tanımlarken sık görülür. Örnek: int dizi[3] = {1, 2, 3, 4}; // Eleman sayısı dizi boyutunu aşıyor

5. NULL terminatörünü unutma (Metin Dizilerinde)
Metin dizilerinde sonlandırıcı karakter (\0) eklenmediğinde, dizi yanlış okunabilir veya belleğin diğer kısımlarına taşabilir. Örnek: char dizi[6] = {'H', 'e', 'l', 'l', 'o'}; printf("%s\n", dizi); // Hatalı sonuç

2-Boyutlu (2B) diziler - Matrisler

C dilinde 2 boyutlu diziler, sıklıkla matris olarak adlandırılır ve satır-sütun düzenine sahip verilere karşılık gelir. Bu diziler, verileri iki boyutlu bir tablo şeklinde organize ederek saklamayı ve işlemeyi mümkün kılar. Her eleman, bir satır ve sütun koordinatı ile tanımlanır, bu da verilerin daha düzenli ve erişilebilir bir şekilde temsil edilmesini sağlar. Örneğin, bir öğrencinin notlarını ders bazında tutmak veya bir ürün stoğunu kategori ve alt kategorilere göre düzenlemek gibi durumlarda, 2 boyutlu diziler oldukça kullanışlıdır.

2 boyutlu diziler matematikteki matris kavramını bilgisayar ortamında modellemek için idealdir ve matris çarpımı, transpoz alma veya determinant hesaplama gibi işlemler için temel oluşturur. Bellekte sıralı bir şekilde iki boyutta depolandıkları için hem okunması hem de yazılması kolaydır. Bu ise matrisleri karmaşık verileri yönetmek ve düzenli bir yapı içerisinde işlem yapmak için güçlü bir araç haline getirir. Bir 2B dizisini tanımlama formatı şu şekildedir:

```
veri tipi  dizi ismi [ satır sayısı ] [ sütun sayısı ];
```

2B Dizilerin özellikleri

- **Dizilim şekli:** 2 boyutlu diziler, satırlar ve sütunlar halinde düzenlenir. Örneğin, int matris[3][4]; ifadesi 3 satır ve 4 sütundan oluşan bir matris tanımlar.
- **Sabit boyut:** C dilinde bir dizinin boyutları derleme zamanında sabitlenir, yani tanımlandıktan sonra değiştirilemez.
- **Bellekte depolama:** Matris elemanları bellekte satır sıralı (row-major order) olarak saklanır. Yani önce birinci satırın tüm sütunları, ardından ikinci satırın sütunları bellekte yer alır.
- **İndis kullanımı:** Elemanlara erişim için sırasıyla satır ve sütun indisleri kullanılır: matris[i][j]. İndisler 0'dan başlar. Örneğin, matris[0][0] ilk elemanı ifade eder.
- **Başlangıç değeri atama:** Matris tanımlanırken değerleri doğrudan atanabilir: Örnek: int matris[2][3] = {{1, 2, 3}, {4, 5, 6}};

Matrislerin kullanım alanları

2 boyutlu diziler, verilerin düzenli bir şekilde saklanması ve işlenmesi gereken durumlarda güçlü bir araç olarak öne çıkar. Satır ve sütun yapısıyla, özellikle tablo biçiminde verilerin düzenlenmesi, analiz edilmesi ve yönetilmesi için etkili bir çözüm sunar. Bu yapılar, sistematik hesaplamalar, matematiksel işlemler ve düzenli veri manipülasyonu gerektiren uygulamalarda hem okunabilirlik hem de işlem kolaylığı sağlayarak programcıların karmaşık problemlere pratik çözümler üretmesine olanak tanır.

Matrislerin kullanım şekilleri

1. **Tabloların Saklanması:** İki boyutlu diziler, öğrenci notları, alışveriş faturaları gibi tablo biçimindeki verileri depolamak için idealdir.
2. **Matematiksel Hesaplamalar:** Matrisler, özellikle lineer cebirde matematiksel işlemler (matris çarpımı, transpoz, determinant hesaplama) için kullanılır.
3. **Grafik Uygulamaları:** Görsel uygulamalarda piksel değerlerini temsil etmek için 2 boyutlu diziler kullanılır.
4. **Oyun Geliştirme:** Oyun haritası veya satranç tahtası gibi düzenli ızgara yapıları 2 boyutlu dizilerle temsil edilebilir.

Avantajları

- Verilerin düzenli ve yapılandırılmış bir şekilde saklanmasını sağlar.
- Matematiksel ve mantıksal işlemleri kolaylaştırır.
- Tablolar ve çok boyutlu ilişkilerle çalışmayı basitleştirir.

Dezavantajları

- Sabit boyutlu yapısı, esneklik gerektiren durumlarda sınırlamalar oluşturabilir.
- Büyük boyutlu matrisler bellekte fazla yer kaplayabilir. Bu durum, bellek yönetimi ve performans açısından dikkat gerektirir.

2B Dizilerini oluşturma

2B dizileri normal tek boyutlu dizilerde olduğu gibi aynı yaklaşımla oluşturulur. Dizi tanımlanırken değerler atandığı gibi, dizi tanımlandıktan sonra da değer ataması yapılabilir. Ancak sıklıkla tercih edilen durum döngüler kullanarak değerlerin atanmasıdır.

Daha önceden belirtildiği üzere 2B dizileri bir matris olup bir tablo şeklinde ifade edilebilmektedir. 2B dizilerini oluştururken bu tablo yapısındaki özelliklerin dikkate alınması gereklidir. Örneğin aşağıdaki gibi 4x3 ebatlarında yani 4 satır ve 3 sütundan oluşan bir tabloyu göz önüne alalım.

Satır indisi \ Sütun indisi	0	1	2
0	35	72	11
1	50	20	68
2	18	44	77
3	30	58	16

C dilinde hatırlanacağı üzere 2B dizileri,

veri tipi dizi adı [satır sayısı][sütun sayısı];

yazım düzenine göre oluşturulmaktaydı. Burada öncelik satır sayısındadır, sütun sayısı ikinci parametredir. Bu bakımdan dizi tanımlanırken ilk adımda satır elemanları bir grup olarak virgüllerle ayrılmış şekilde belirtilmesi gerekmektedir. Buradaki tablonun C'de tanımlanması şu şekilde olacaktır.

```
int matris[4][3] = { {35, 72, 11}, {50, 20, 68}, {18, 44, 77}, {30, 58, 16} }
                     1. satır      2. satır      3. satır      4. satır
```

C derleyicisinin bir özelliği olarak dizi atama işlemlerinin bir kısmı otomatik olarak yapılabilmektedir. Örneğin aşağıdaki tanımlama şekilleri yukarıdaki tanımlama ile aynı sonucu üretmektedir.

```
int matris[][3] = { {35, 72, 11}, {50, 20, 68}, {18, 44, 77}, {30, 58, 16} }
int matris[4][3] = {35, 72, 11, 50, 20, 68, 18, 44, 77, 30, 58, 16 }
```

C'deki 2B dizilerini otomatik düzenleme örnekleri

C derleyicisi, satır yönünde dizi elemanlarına, ilk satırın ilk elemanından başlanıp, daha sonra sırayla diğer satırlara geçilerek ilk değerler atama yapmaktadır. Buna **satır yönünde atama** adı verilir. Şimdi bu durumu bazı örnekler üzerinde açıklayalım.

1. örnek: `int matris[][3] = { {35, 72, 11}, {50, 20, 68}, {18, 44, 77}, {30, 58, 16} }`

Bu örnekte satır sayısı verilmemiştir. Ancak burada genel dizi tanımı gösteren dış parantezlerin içinde her bir satır grubunu bir birinden ayıran virgül sayısı derleyiciye genel satır sayısı bilgisini vermektedir.

2. örnek `int matris[4][3] = {35, 72, 11, 50, 20, 68, 18, 44, 77, 30, 58, 16 }`

Bu örnekte ise iç parantezler belirtilmemiştir. Ancak satır sayısı ve sütun sayısı değerleri verilmiştir. Bu durumda derleyici her bir satırın 3 elemandan oluştuğu bilgisine ulaşmaktadır ve bütün verilen değerleri 3'li gruplar halinde satır bazlı olarak oluşturabilmektedir.

3. örnek `int matris[4][3] = {35, 72, 11, 50 }`

Bu örnek görüldüğü üzere verilen eleman sayısı eksik verilmiştir. Ancak satır ve sütun sayısı bilgileri mevcut olduğundan derleyici verilen matrisi oluşturabilmektedir. Verilen değerler satır bazlı olarak diziye aktarılır ve eksik olan değerlere ise sıfır değeri atanır.

35	72	11
50	0	0
0	0	0
0	0	0

4. örnek `int matris[4][3] = {{35}, {50}, {18}, {30} }`

Bu örnekte de satır ve sütun sayısı bilgileri verilmiştir. Ancak her bir satır için sadece bir eleman bilgisi mevcuttur. Bu durumda derleyici satır ve sütun sayısı bilgilerinden istenilen matrisi oluşturacaktır. Her bir satırın ilk eleman bilgisi de mevcut olduğundan geri kalan elemanlara otomatik olarak sıfır değeri atanacaktır.

35	0	0
50	0	0
18	0	0
30	0	0

2B dizi elemanlara tek tek değer atama

2B dizilerine tek tek değer atama, satır ve sütun indisleri kullanılarak gerçekleştirilir. Bu yöntemde her bir elemana

dizi[satır_indisi][sütun_indisi] = değer;

formatıyla değer ataması yapılır. Eğer atanacak değer bir metin ise sütun indisi metin değişkendeki maksimum karakter sayısını gösterecektir.

```
int sayilar[2][3];
sayilar[0][0] = 10;
sayilar[0][1] = 20;
sayilar[0][2] = 30;
sayilar[1][0] = 40;
sayilar[1][1] = 50;
sayilar[1][2] = 60;
```

10	20	30
40	50	60

```
// 2B metin dizisi tanımlama
char metinler[3][20] = {"Merhaba", "Dünya", "C Dili" };

// 2B metin dizisi tanımlandıktan sonra atama
char metinler[4][30];

strcpy(metinler[0], "Merhaba");
strcpy(metinler[1], "Dünya");
strcpy(metinler[2], "C Dili");
strcpy(metinler[3], "Eğlencelidir");
```

2B metin dizileri gerçek anlamda bir 2B dizisi değildir. Görüldüğü üzere bir metin dizi şeklinde tek boyutlu bir dizi yapısı söz konusudur. Metin dizisinde ikinci boyut ile her bir metin dizisinin tek tek karakterlerine erişme söz konusudur. Metin dizisi tanımlandıktan sonra atama işlemi strcpy() fonksiyonu ile yapılmalıdır. (string.h ile)

```
// Metinleri ekrana yazdırma
for (int i = 0; i < 3; i++) {
   printf("%d. eleman: %s\n", i, metinler[i]);
}
printf("1. dizinin 2. elemanı: %c\n", metinler[0][1]);
```

```
1. eleman: Merhaba
2. eleman: Dünya
3. eleman: C Dili
1. dizinin 2. elemanı: e
```

2B diziye değer atama – döngü ile

Bir dizi tanımlandıktan sonra elemanları döngü kullanılarak atanabilir. Örnek olarak 4x3 şeklinde tanımlanan bir matrise 10 ile 80 arasında rastgele değerler atama yapılan bir durumu dikkate alalım.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
    int matris[4][3];

    // Rastgele sayı üretmek için srand fonksiyonu
    srand(time(0)); // Her çalıştırmada farklı sonuçlar için

    // Matris elemanlarına rastgele değer atama
    for (int i = 0; i < 4; i++) {     // satır döngüsü
        for (int j = 0; j < 3; j++) {   // sütun döngüsü
            matris[i][j] = 10 + rand() % 71; // 10 ile 80 arasında değer üretir
        }
    }

    // Matrisi ekrana yazdırma
    printf("4x3 Rastgele Matris:\n");

    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 3; j++) {
            printf("%d ", matris[i][j]);
        }
        printf("\n");  // her bir satırın sonu belirtir
    }

    return 0;
}
```

2B dizide satır/sütun işlemleri

Bir 2B dizisinin satır ya da sütunları üzerinde işlem yapmak için dış ve iç döngülerin sırasını değiştirmek yeterlidir. Dış döngü satır döngüsü olduğu durumda her bir sütundaki elemanlar üzerinde işlem yapılabilir. Tersi durumda yani dış döngü sütun döngüsü olduğu durumda ise her satırın elemanları üzerinde işlem yapılabilmektedir. Örneğin daha önce vermiş olduğumuz 4x3 matrisi dikkate alalım. Şimdi bu matriste her bir satırın ve sütunun toplamlarını ayrı ayrı hesaplayan kodu hazırlayalım.

```c
#include <stdio.h>
int main() {
    int matris[4][3] = {{35, 72, 11},{50, 20, 68},{18, 44, 77},{30, 58, 16}};
    int stn_top, str_top;

    // Her bir satırın toplamı
    for (int satir = 0; satir < 4; satir++) {

        str_top = 0;

        for (int sutun = 0; sutun < 3; sutun++) {
            str_top += matris[satir][sutun];
        }
         printf("%d. satırın toplamı: %d\n", satir+1, str_top);
    }

    printf("\n");

    // Her bir sütunun toplamı
    for (int sutun = 0; sutun < 3; sutun++) {

        stn_top = 0;

        for (int satir = 0; satir < 4; satir++) {
            stn_top += matris[satir][sutun];
        }
         printf("%d. sütunun toplamı: %d\n", sutun+1, stn_top);
    }

    return 0;

}
```

35	72	11	118
50	20	68	138
18	44	77	139
30	58	16	104
133	194	172	

```
1. satırın toplamı: 118
2. satırın toplamı: 138
3. satırın toplamı: 139
4. satırın toplamı: 104

1. sütunun toplamı: 133
2. sütunun toplamı: 194
3. sütunun toplamı: 172
```

Dizileri ve matrisleri fonksiyonlarda kullanma

C dilinde diziler ve matrisler, çok sayıda veriyi sistematik bir şekilde yönetmek için kullanılır. Bu yapıların fonksiyonlarda kullanılması, kodun daha düzenli, okunabilir ve yeniden kullanılabilir olmasını sağlar. Fonksiyonlar, diziler ve matrisler üzerinde belirli işlemleri (örneğin sıralama, toplama, yazdırma) gerçekleştirmek için özelleştirilebilir.

Dizileri Fonksiyona Geçirme

- Diziler fonksiyonlara adres referansı yoluyla geçirilir. Bu, dizinin kopyalanmadığı, sadece başlangıç adresinin (ilk elemanın adresi) gönderildiği anlamına gelir.
- Bu yöntem, özellikle büyük boyutlu dizilerle çalışırken bellek tasarrufu ve performans sağlar.
- Fonksiyona gönderilen dizi üzerinde yapılan değişiklikler, çağıran kod tarafından da görülür (çünkü aynı bellek bölgesi üzerinde çalışılır).

Fonksiyonlarda Dizi Parametresi

- Dizi parametreleri, bir dizi türü (int arr[]) veya işaretçi türü (int *arr) olarak tanımlanır. Her iki yöntem de işlevsellik açısından aynıdır.
- Dizinin boyutu genellikle ayrı bir parametre olarak gönderilir, çünkü C dilinde dizi boyutları otomatik olarak aktarılmaz.

Matrisleri Fonksiyona Geçirme

- Matrisler, tek boyutlu dizilerden farklı olarak, her boyutun boyut bilgisini fonksiyona geçirirken belirtmek gerekir.
- 2 boyutlu bir diziyi fonksiyona geçirirken, ikinci boyutun (sütun sayısı) belirtilmesi zorunludur, çünkü bellekte bir 2 boyutlu dizi ardışıl bir veri bloğu olarak tutulur ve bellekte bu sıralama sütün boyutuna göre düzenlenir.

Fonksiyonlarda Matris Parametresi

- 2 boyutlu bir dizi, bir int arr[][SUTUN] veya bir işaretçi olarak tanımlanabilir.
- Matrisin satır ve sütun bilgileri ayrı parametrelerle verilmelidir.

Tek boyutlu bir diziyi fonksiyona geçirme

Tek boyutlu bir dizinin fonksiyona aktarılması örneği olarak verilen bir dizinin elemanlarının toplamını bir fonksiyon içinde hesaplayan bir durumu göz önüne alalım. Burada ilk adımda verilen dizi parametre olarak fonksiyona aktarılacaktır. Daha sonra fonksiyon için bir döngü ile aktarılan dizinin elemanları toplanacaktır. Son adımda elde edilen toplan geri dönüş değeri olarak ana programa gönderilecektir.

```c
#include <stdio.h>
// Fonksiyon diziyi alır ve elemanlarını toplar
int diziToplam(int arr[], int boyut) {
    int toplam = 0;
    for (int i = 0; i < boyut; i++) {
        toplam += arr[i];
    }
    return toplam;
}

int main() {
    int dizi[] = {1, 2, 3, 4, 5};
    int boyut = sizeof(dizi) / sizeof(dizi[0]);

    printf("Dizinin toplamı: %d\n", diziToplam(dizi, boyut));

    return 0;
}
```

```
Dizinin toplamı: 15
```

Matrisleri bir diziyi fonksiyona geçirme

2 boyutlu bir dizinin fonksiyona aktarılması örneği olarak verilen bir dizinin elemanlarını bir fonksiyon içinde terminale yazdırmak istediğimizi düşünelim. Burada ilk adımda verilen 2B dizisi parametre olarak fonksiyona aktarılacaktır. Daha sonra fonksiyon için bir döngü ile aktarılan dizinin elemanları satır ve sütun bazında tek tek terminale yazdırılacaktır.

```c
#include <stdio.h>
// Fonksiyon matrisin elemanlarını yazdırır
int matrisYazdir(int matris[][3], int satir) {
    for (int i = 0; i < satir; i++) {
        for (int j = 0; j < 3; j++) {
            printf("%d ", matris[i][j]);
        }
        printf("\n");
    }
}

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

    printf("Matrisin elemanları\n");

    matrisYazdir(matris, 2);

    return 0;

}
```

```
Matrisin elemanları
1 2 3
4 5 6
```