Göstergeler/işaretçiler - pointers

C dilinde göstergeler (pointers), bir değişkenin bellekteki adresini saklayan özel türde değişkenlerdir. Göstergelerin temel işlevi, bellekteki verilere doğrudan erişim sağlamaktır. Bu özellik, programların çalışma şeklinde esneklik sunar ve aynı zamanda performansı artırır. Göstergeler, bellek yönetimi gibi düşük seviyeli işlemlerden, diziler ve fonksiyonlarla çalışmaya kadar birçok önemli alanda kullanılır. Örneğin, bir değişkenin değerine dolaylı yoldan erişmek veya bir dizinin elemanları üzerinde işlem yapmak için göstergelerden faydalanılır. Bu yönüyle göstergeler, programcıya hem güçlü bir kontrol mekanizması hem de etkili bir çözüm sunar.

Göstergeler, özellikle dinamik bellek tahsisi, veri yapılarının uygulanması ve işlevlere veri geçişinde kritik bir rol oynar. Dinamik bellek tahsisinde, göstergelerle bellekte ihtiyaç duyulan alan rezerve edilir ve işlem tamamlandığında bu alan serbest bırakılır. Ayrıca bağlantılı listeler, ağaç yapıları ve grafikler gibi karmaşık veri yapılarının oluşturulması ve yönetilmesi de göstergeler sayesinde mümkün hale gelir.. Bununla birlikte, göstergelerle çalışırken dikkatli olunması gerekir; yanlış bellek erişimi veya bellek sızıntısı gibi sorunlar programın çalışmasını olumsuz etkileyebilir.

Göstergelerin Genel Özellikleri

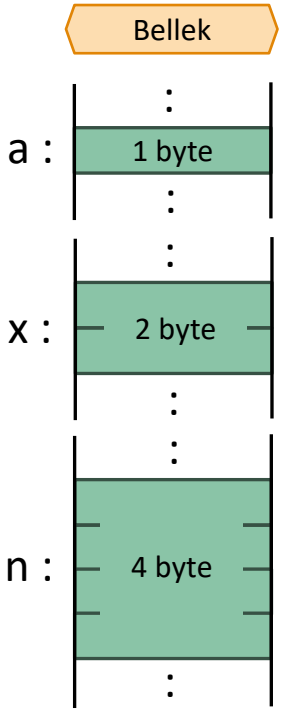
1. **Bellek Adreslerini Saklarlar:** Göstergeler bir değişkenin bellek adresini saklar, bu adres sayesinde değişkenin içeriğine erişilir.
2. **Doğrudan Bellek Erişimi:** Bellekte doğrudan erişim ve işlem yapma imkanı sunar.
3. **Dinamik Bellek Yönetimi:** malloc, calloc, ve free gibi fonksiyonlarla dinamik bellek tahsisinde kullanılır.
4. **Esneklik Sağlar:** Diziler, dizilerden oluşan diziler (multi-dimensional arrays), işlevler (functions) ve veri yapıları üzerinde işlem yaparken esneklik sağlar.
5. **Performansı Artırır:** Kopyalama yerine bellekteki veriyle doğrudan çalışıldığı için performans açısından avantaj sağlar.

Değişkenlerin bellek yerleşimi

Bir program içerisinde tanımlanan tüm değişkenler, program çalıştırıldığında bellekte yer tutarlar. Bellek, her biri farklı adreslerle tanımlanmış küçük hücrelerden meydana gelir. Bu adresler, hangi verinin nerede saklandığını belirlemek ve erişmek açısından önemlidir. Eğer bellek adresleri olmasaydı, programlar bellekteki verilere ulaşamaz ve işlemleri gerçekleştiremezdi. Bir değişken tanımlandığı zaman, bu değişkenin veri tipine bağlı olarak bellekte belirli bir uzunlukta yer ayrılır. Örneğin, **char** tipi için 1 bayt, **short int** tipi için 2 bayt, **float** tipi için 4 bayt büyüklüğünde alanlar rezerve edilir. Bu mekanizma, programların verileri düzenli bir şekilde yönetmesini ve işlem yapmasını sağlar.

Bir programın başlangıcında "a", "x" ve "n" isimli değişkenler aşağıdaki şekilde tanımların yapıldığı bir durumu dikkate alalım. Programın çalıştırılması ile birlikte bu değişkenler için bellekte, tanımlanan veri tiplerine göre gerekli büyüklükte yerler rezerve edilecektir. Bu işlem sembolik olarak şöyle gösterilebilir.

```
char a;
short x;
long n;
```



Burada "a" değişkeni karakter tipi olduğundan bellekte bu değişken için 1 byte'lık yer ayrılacaktır. Bu değişken için başlangıç adresinin 0010 olduğunu varsayarsak 0010 adresinden itibaren 1 byte'lık yani 8 bit'lik bir alan bloğu "a" değişkeni için kullanılacaktır.

Gösterge kavramı ve tanımlanması

Değişkenlerinin değerlerinin saklandığı bellek hücreleri, başka değişkenlerin adreslerini saklamak için de kullanılabilir. İşte başka değişkenlerin adreslerini bellek tutan özel değişkenlere gösterge adı verilir. Bir adres göstermesi sebebiyle gösterge değişkenleri bellekte 2 byte'lık yer kaplarlar. Göstergelerin genel yazım düzenler şöyledir.

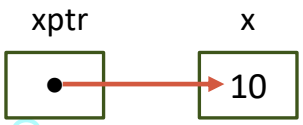
```
veri tipi  *  değişken adı ;
```

Burada veri tipi kullanılacak değişkenin türüne göre seçilir ve **char**, **int**, **float** gibi veri türlerinden her hangi biri olabilir. **"*"** sembolü kullanılacak değişken bir gösterge olduğu belirtir. **"*"** ile tanımlanan değişkenlere, önceden tanımlanmış değişkenlerin adresleri **"&"** adres operatörü kullanılarak atanabilir.

```
int *ptr;     // Bir tamsayıya işaret eden bir gösterge
float *fptr;  // Bir float değişkene işaret eden bir gösterge
char *cptr;   // Bir karaktere işaret eden bir gösterge
```

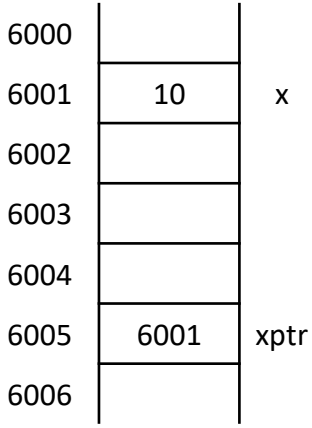
```
int x = 10;
int *xptr = &x;  // 'xptr' artık 'x' değişkeninin adresini saklar
```

Son örnekte **"x"** ve **"xptr"** program içinde tanımlanmış iki değişkendir. "x"in içeriği tamsayı iken **"xptr"** değişkeni ise bir göstergedir ve "x"in bellek adresini tutmaktadır.



Bu durumun bellek yerleşimi temsili olarak yandaki gibi gösterilebilir. "x" değişkeni 6001 no'lu adreste olduğunu varsayılsın. 6005 adresinde olan gösterge değişkeni "x" değişkeninin 6001 no'lu adres değerine sahiptir.

Bellek yerleşimi



Gösterge operatörleri

C dilinde göstergelerle (pointers) çalışırken kullanılan operatörler, gösterge değişkenlerinin tanımlanması, atanması ve değerlerine erişim için kritik bir rol oynar. Bu operatörler sayesinde, bellek adresleriyle kolayca işlem yapılabilir ve düşük seviyeli bellek yönetimi sağlanabilir.

Gösterge operatörleri genellikle iki ana kategoriye ayrılır:

- **"&" adres operatörü** ve
- **"*" içerik operatörü**

Adres Operatörü (&)

Bir değişkenin bellekteki adresini elde etmek için kullanılır. Değişkenin başına "&" operatörü konularak, o değişkenin saklandığı bellek adresine ulaşabilirsiniz. Adres, bir göstergeye atanabilir veya bellekle ilgili işlemlerde kullanılabilir. Bu adres değerleri program her çalıştırıldığında farklı bir adres değeri almaktadır.

```c
#include <stdio.h>
int main() {
    int x = 42;
    int *ptr = &x;  // x'in adresini ptr göstergesine atama

    printf("x'in adresi: %p\n", ptr);
    printf("x'in değeri: %d\n", *ptr);

    return 0;
}
```

```
x'in adresi: 000000937CBFFC14
x'in değeri: 42
```
1. çalışma

```
x'in adresi: 000000985BDFF6E4
x'in değeri: 42
```
2. çalışma

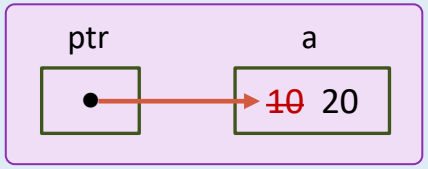
İçerik Operatörü (*)

Bir göstergenin işaret ettiği bellek adresindeki veriye erişmek veya bu veriyi değiştirmek için kullanılır. Gösterge değişkeninin başına "*" operatörü eklenerek, bellekteki ilgili alana erişim sağlanır. Bu işleme **"dereferencing"** (gösterge çözme) adı verilmektedir. İçerik operatörü kullanılarak bellek adresindeki veri, hem okunabilir hem de güncellenebilir. Yani, "*" operatörünün iki kullanım amacı vardır. Birincisi, eğer tanıtım aşamasında değişkenin önüne getirilirse, o değişkenin gösterge olduğu belirtilmiş olur. İkincisi, eğer kod içerisinde bir gösterge değişkenin önüne getirilirse o değişkende kayıtlı adres üzerinde işlem yapılacağı belirtilir.

```c
#include <stdio.h>
int main() {
    int a = 10;
    int *ptr = &a;

    printf("ptr'nin işaret ettiği değer: %d\n", *ptr);
    *ptr = 20; // ptr'nin işaret ettiği alanı günceller, yani a'nın değeri değişir
    printf("a'nın değeri: %d\n", a);

    return 0;
}
```




```
ptr'nin işaret ettiği değer: 10
a'nın değeri: 20
```

Adres ve içerik operatörlerin birlikte kullanımı

& ve * operatörleri birlikte kullanılarak bir değişkenin adresi alınabilir ve bu adreste saklanan veriye erişim sağlanabilir. Örneğin:

```c
int a = 30;
int *ptr = &a;
printf("a'nın adresindeki değer: %d\n", *(&a)); // * ve & operatörleri birleştirildi
```

```
a'nın adresindeki değer: 30
```

Göstergelerde veri tipinin seçimi

Gösterge atamalarında dikkat edilmesi gereken önemli bir nokta, değişkenlerin tipleridir. Bir gösterge değişkeni ile işaret edilen değişkenin tiplerinin aynı olması, doğru verilere erişmek için gereklidir. Örneğin, bir **long int** tipi değişken 4 baytlık bir bellek alanını kaplar ve bu alan tamsayı verileri saklamak için tasarlanmıştır. Eğer bu değişkeni işaret eden bir gösterge de **long int** tipi olursa, doğru şekilde bellekteki veriye erişilebilir. Ancak, işaretçi değişken **float** tipinde olursa, yine 4 baytlık bir alana erişilir fakat bu alan kayan noktalı sayı formatında yorumlanır. Bu durumda, örneğin 32556 sayısını bir **long int** olarak kaydedip **float** olarak okumaya çalışırsak, yanlış bir değer elde ederiz.

Aşağıdaki kod bu yanlış bir uygulamaya bir örnektir. Burada **"m"** değişkenin karakter tipi değişkendir. Ancak bu değişkenin göstergesi **"p"** tamsayı değişkeni olarak tanımlanmıştır. Bu tür farklı değişken tanımlamalarında derleyici "uyumsuz gösterge" hatası verecektir. Bu yüzden, bir göstergenin doğru çalışabilmesi için işaret ettiği bellek alanındaki verinin tipine uygun olarak tanımlanması gerekir. Bu, verilerin güvenilir şekilde saklanması ve işlenmesi açısından büyük önem taşır.

```
char m = 'x';
int *p;
p = &m;
```

```
error: initialization of 'int *' from incompatible pointer type 'char *'
```

Bir göstergenin bellekte herhangi bir bellek hücresini işaret etmesini istemediğimiz durumlarda, NULL isimli sabit kullanılır. NULL, bir göstergenin hiçbir bellek alanını göstermediğini ifade eden özel bir değerdir ve genellikle 0 ile tanımlanır. Bu yaklaşım, göstergelerin kullanılmadığı durumlarda onları tanımsız veya geçersiz bir adresle bırakmak yerine açıkça işaretlemenin bir yoludur. Aşağıdaki örnek gösterge NULL değeriyle boşaltılmaktadır.

```
int *hic;
hic = NULL;
```

Göstergelerin birbirlerine atanması

C dilinde bir göstergenin başka bir göstergeye atanması, bir göstergenin adres bilgisinin diğer bir göstergeye depolaması anlamına gelir. Bu işlem, özellikle göstergeler arasında veri paylaşımı, işlevler arasında veri iletimi ve karmaşık veri yapılarının manipülasyonu için oldukça önemlidir. Ancak, bu tür bir atama sırasında belirli özellikler ve gereklilikler dikkate alınmalıdır:

1. Göstergelerin Veri Tiplerinin Uyumu

Göstergelerin başarılı bir şekilde atanabilmesi için, her iki göstergenin veri tipi aynı olmalıdır. Örneğin:

- **int* ptr1** ve **int* ptr2** aynı veri tipini işaret ettiği için doğrudan atama yapılabilir.
- Ancak **int*** tipindeki bir gösterge, **float*** tipindeki bir göstergeye atanamaz. Çünkü bu durumda, işaret edilen bellek alanındaki veri tipi farklı yorumlanır ve bu, belirsiz davranışlara yol açabilir.

```
int x = 10;
int *ptr1 = &x;
int *ptr2 = ptr1; // Geçerli: Aynı veri tipinde göstergeler.
```

2. Bellek Adresinin Geçerliliği

Bir göstergenin başka bir göstergeye atanması işleminde, eğer kaynak gösterge geçersiz veya tanımsız bir adresi işaret ediyorsa, bu tür bir atama bellek hatalarına neden olabilir.

3. NULL Değerinin Atanması

Bir göstergenin NULL durumunda olması, bellekte herhangi bir alanı işaret etmediğini gösterir. Eğer NULL bir gösterge başka bir göstergeye atanırsa, her iki gösterge de bellek alanını işaret etmez hale gelir.

```
int *ptr1 = NULL;
int *ptr2 = ptr1; //  Artık ptr2 de herhangi bir belleği işaret etmiyor.
```

4. İlişkili Bellek Alanları

Bir göstergenin başka bir göstergeye atanması, her iki göstergenin aynı bellek adresini işaret etmesine neden olur. Bu durum, iki gösterge üzerinden aynı veriye erişim sağlanmasına olanak tanır. Ancak, biri üzerinden yapılan değişiklik diğer göstergeyi de etkiler. Bu nedenle, bu tür bir atama yaparken veri bütünlüğüne dikkat edilmelidir.

```
int x = 20;
int *ptr1 = &x;
int *ptr2 = ptr1;
*ptr2 = 50; // Hem ptr1 hem de ptr2 üzerinden x'in değeri değişir.
```

5. Tür Dönüşümü ile Atama (Cast)

Eğer farklı veri tiplerine sahip göstergeler arasında bir atama yapılması gerekiyorsa, tür dönüşümü (type casting) kullanılır. Ancak bu, genellikle bellekle doğrudan çalışılması gereken düşük seviyeli programlama durumlarında kullanılır ve dikkatli yapılmalıdır. Yanlış bir tür dönüşümü, verilerin hatalı bir şekilde işlenmesine yol açabilir.

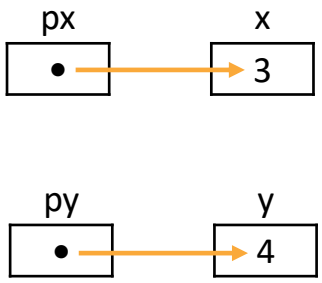
```
void *ptr1;
int x = 100;
ptr1 = &x;
int *ptr2 = (int*)ptr1; // Tür dönüşümü ile geçerli hale gelir.
```

Sonuç olarak bir göstergenin başka bir göstergeye atanması güçlü ve esnek bir yöntemdir, ancak doğru yapılmadığında bellek hatalarına, çakışmalara ve veri kaybına neden olabilir. Bu nedenle, veri tiplerinin uyumuna, geçerli bellek adreslerine ve işaretçilerin kullanım bağlamına dikkat etmek kritik öneme sahiptir.

Göstergelerin birbirlerine atanması – örnek kod

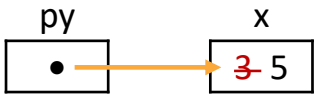
Aşağıdaki örnek kodu inceleyelim. Program çalıştırıldığında 4. satırın sonunda "px" göstergesi "x" değişkenini, "py" göstergesi ise "y" değişkenini göstermiş olacaktır. Bu durum alttaki şekilde temsili gösterilmiştir.

```
1   int *px,*py;
2   int x=3, y=4;
3   px = &x;
4   py = &y;
5   py = px;
6   *py = 5;
7   printf("x\'in değeri: %d", *px)
```



5. satırda "py = px" ataması ile "px"in gösterdiği adres "py"ye kopyalanır. Artık "py" göstergesi artık "x" değişkeni adresini gösterecektir. Bu işlem "py"nin daha önce gösterdiği "y" değişkeni adresinin silinmesine sebep olacaktır.

Dolayısıyla 6. satırda "*py = 5" ataması artık "y" değerini değil "x" değerini değiştirecektir. Bu durumun bellek görüntüsü ise şöyle olacaktır. Bu nedenle 7. satırdaki printf çıktısında "x" değeri 5 olarak yazılacaktır.




```
x'in değer: 5
```

Gösterge okutmak ve yazdırmak

Önceki örneklerde de gösterildiği üzere içerik operatörü **"*"** ile bir değişkenin gösterge olacağı belirtildiği gibi bir göstergenin işaret ettiği adresteki değeri okutmak ve aynı adrese yeni değer atamak için de kullanılmaktadır. Diğer taraftan Gösterge bir değişkenin adresini tuttuğu için, değişkenin adresini ya da adresin gösterdiği değerini ayrı ayrı yazdırmak mümkündür. Göstergenin tuttuğu adres **"%p"** yer tutucu formatıyla yazdırılır. Adresin gösterdiği değer ise veri türüne uygun formatla yazdırılır.

```c
#include <stdio.h>
int main() {
    char c = 'A';    // Karakter tipi bir değişken
    char* ptr = &c;  // c'nin adresini tutan gösterge

    printf("c'nin adresi: %p\n", ptr);     // Gösterge adresini yazdırır.
    printf("c'nin değeri: %c\n", *ptr);    // Göstergeyi kullanarak c'nin değerini yazdırır.
    return 0;
}
```

```
c'nin adresi: 00000070451FFDE7
c'nin değeri: A
```

Bir göstergenin işaret ettiği bellek hücresine kullanıcıdan alınan bir değeri atamak istersek, gösterge zaten bir hücrenin adresini taşıdığı için tekrardan "&" adres operatörünü kullanmaya gerek yoktur. Örneğin scanf fonksiyonunda değişken parametre yerine direk gösterge değişkeninin ismini yazmak yeterli olacaktır.

```c
int y;
int* ptr = &y;
printf("Bir sayı girin: ");
scanf("%d", ptr);     // Göstergeyi kullanarak y'ye değer atar.
printf("Girilen sayı: %d\n", *ptr);  // Gösterge üzerinden değeri okur.
```

Gösterge operatörleri & ve * kullanımına örnek

Gösterge operatörleri "&" ve "*" birbirlerinin eşleniğidir. Birlikte kullanıldıkları durumlarda öncelik sırasının bir önemi yoktur ve aynı sonuçları verirler. Aşağıdaki örnek kod bu durumları göstermektedir.

```c
#include <stdio.h>
int main() {
    int a;          /*  a tanıtılır.  */
    int *aPtr;      /*  aPtr göstergesi tanıtılır.  */

    a = 7;
    aPtr = &a;      /*  göstergeye a'nın adresi atanır.  */

    printf("a'nın adresi: %p\n", &a);          /*  a'nın adresi yazılır.  */
    printf("aPtr'nin degeri: %p\n", aPtr);     /*  a'nin adresi cikar.   */

    printf("a'nın degeri: %d\n", a);           /*  a'nin sayı değeri cikar.  */
    printf("*aPtr'nin degeri: %d\n", *aPtr);   /*  a'nin sayı değeri cikar.  */

    printf("a'nın adresi: %p\n", &*aPtr);        /*  a'nin adresi cikar.  */
    printf("a'nın adresi: %p\n", *&aPtr);        /*  a'nin adresi cikar.  */

    return 0;
}
```

```
a'nın adresi: 000000A4993FFE1C
aPtr'nin degeri: 000000A4993FFE1C
a'nın degeri: 7
*aPtr'nin degeri: 7
a'nın adresi: 000000A4993FFE1C
a'nın adresi: 000000A4993FFE1C
```

Gösterge aritmetiği

Gösterge aritmetiği, bir göstergeyi (pointer) artırma, azaltma veya başka bir göstergeyle karşılaştırma gibi işlemleri ifade eder. Bu, bellekteki belirli konumlara ulaşmayı sağlar ve özellikle dinamik veri yapılarını yönetmek, dizilerle işlem yapmak, bellek blokları arasında gezinmek ve belirli bir alan üzerinde optimize edilmiş işlemler gerçekleştirmek için çok güçlü bir araçtır. Ancak, dikkatli ve doğru bir şekilde kullanılmadığında ciddi hatalara neden olabileceğinden, gösterge aritmetiği kullanırken titizlik gereklidir.

Gösterge aritmetiği işlemleri şu başlıklar altında verilebilir:

1. Artırma ve Azaltma Operasyonları (+ ve -)

Bir göstergeye 1 eklemek (ptr++ veya ptr = ptr + 1), göstergeyi bir sonraki bellek hücresine taşır. Ancak bu işlem, gösterge tarafından işaret edilen veri türünün boyutuna bağlıdır. Örneğin:

- Eğer bir gösterge **int** tipindeyse (4 byte), artırma işlemi göstergeyi **4 byte ilerletir.**
- Eğer bir gösterge **char** tipindeyse (1 byte), artırma işlemi göstergeyi yalnızca **1 byte ilerletir**.

Benzer şekilde, bir göstergeyi azaltmak (ptr-- veya ptr = ptr - 1), göstergeyi bir önceki bellek hücresine taşır.

```
int *xptr, x;  // int veri tip 4 byte'lık alan kaplar
xptr = &x;     // xptr'nin 1000 adresini gösterdiği varsayılsın.
xptr++;        // bu durumda xptr'nin değeri 1004 olacaktır.
```

2. Gösterge ile Sabit Değer Aritmetiği (ptr + n, ptr - n)

Bir göstergeye sabit bir sayı eklemek veya çıkarmak, göstergenin işaret ettiği bellek adresini, o göstergeyle ilişkili veri türünün boyutunun büyüklüğü kadar ileriye veya geriye taşır. Örneğin:

```
int arr[5] = {10, 20, 30, 40, 50};
int *ptr = arr;      // ptr ilk elemanı işaret eder
ptr = ptr + 2;       // ptr şimdi üçüncü elemanı işaret eder (arr[2])
```

3. Gösterge Farkı (ptr1 - ptr2)

İki gösterge arasındaki fark, bu iki göstergenin işaret ettiği adresler arasındaki bellek hücrelerinin sayısını verir. Örneğin, bir **int** türü 4 baytlık bir alan kaplıyorsa, iki **int** göstergesi arasındaki fark, adresler arasındaki bayt farkının 4'e bölünmesiyle bulunur. Fark işlemleri yalnızca aynı dizinin elemanlarını işaret eden göstergeler arasında geçerlidir.

```
int arr[5] = {10, 20, 30, 40, 50};
int *ptr1 = arr[1];      // dizinin 2. elemanını işaret eder
int *ptr2 = arr[4];      // dizinin 5. elemanını işaret eder
int diff = ptr2 - ptr1;  // diff = 3 (4-1)
```

4. Gösterge Karşılaştırmaları (<, >, <=, >=, ==, !=)

Göstergeler arasındaki karşılaştırmalar, bellek adreslerine göre yapılır ve söz konusu iki göstergenin bellek üzerindeki konumlarını karşılaştırmak için kullanılır. Bu tür karşılaştırmalar özellikle dizilerde, bir göstergenin belirli bir değeri içerip içermediği veya dizinin sınırları içinde kalıp kalmadığını kontrol etmek için kullanılır. Genellikle dizilerde sıralama kontrolü gibi işlemler için oldukça yararlıdır.

```
int dizi[] = {10, 20, 30, 40, 50};
int *ilkptr = dizi;      // dizinin ilk elemanını işaret eder
int *sonptr = dizi + 4;  // dizinin son elemanını işaret eder

// Göstergeleri karşılaştırarak döngüyle diziyi yazdırma
printf("Dizinin tersten elemanları:\n");

while (sonptr >= ilkptr) {
    printf("%d ", * sonptr);      // Gösterilen elemanın değerini yazdır
    sonptr--;      // Göstergeyi bir önceki elemanı işaret edecek şekilde geriye al
}
```

```
Dizinin tersten elemanları:
50 40 30 20 10
```

Göstergeler ve Diziler

Göstergeler (pointers) ve diziler (arrays) arasında oldukça güçlü ve doğrudan bir ilişki bulunmaktadır. Bilindiği üzere diziler bellekte ardışık olarak saklanmaktadır ve bu yapısal özellik, dizinin adı üzerinden dizinin ilk elemanının adresine ulaşmayı mümkün kılmaktadır. Başka bir deyişle, bir dizi adı, dizinin ilk elemanının adresini temsil eden bir gösterge gibi davranır. Bu sayede, dizilerin elemanlarına doğrudan göstergeler aracılığıyla erişmek veya bu elemanlarla işlem yapmak oldukça pratik hale gelir.

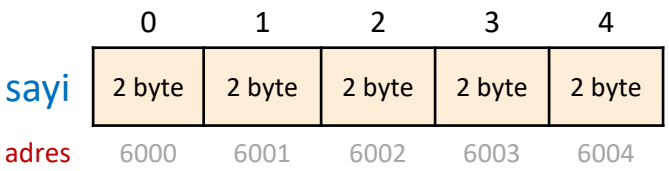
Bu ilişki, dizilerle göstergelerin birlikte kullanılmasını kolaylaştırır ve birçok durumda hem bellek yönetimini hem de işlem hızını optimize eder. Özellikle büyük veri kümeleriyle çalışırken ya da dinamik veri yapılarını işlerken, bu özellikler büyük avantaj sağlar. Göstergelerle yapılan diziler üzerindeki işlemler, bellekten veri okuma ve yazma işlemlerini hızlandırırken, aynı zamanda kodun esnekliğini artırır. Dolayısıyla, bu kombinasyon, verimli ve performans odaklı programlama için C dilinin en önemli özelliklerinden biri olarak öne çıkar.

Göstergelerle diziler arasındaki güçlü ilişkiler belirli başlıklar altında özetlenebilir.

1. **Dizilerin adresi ve göstergeler:** Bir dizinin adı, dizinin ilk elemanının adresine eşdeğerdir. Örneğin, `int arr[5];` ifadesinde **arr** ifadesi `&arr[0]` adresini temsil eder. Bu nedenle, bir diziye gösterge üzerinden erişmek için `int *ptr = arr;` yazılarak, **ptr** dizinin ilk elemanını gösteren bir göstergeye dönüştürülebilir.
2. **Dizi elemanlarına göstergelerle erişim:** Dizi elemanlarına hem köşeli parantez `"[ ]"` hem de gösterge aritmetiği ile erişilebilir. Örneğin, `arr[2]` ile `*(arr + 2)` aynı işlemi yapar ve dizinin üçüncü elemanına erişir. Göstergelerle bu erişim yöntemleri, özellikle dinamik olarak boyutlandırılan dizilerde ve büyük veri işlemlerinde kullanışlıdır.
3. **Göstergelerin dizilerle esnek kullanımı:** Göstergeler, diziler arasında geçiş yapmayı veya bir dizinin bir alt kümesine erişmeyi kolaylaştırır. Örneğin, `ptr = &arr[2];` ifadesi, **ptr** göstergesini dizinin üçüncü elemanına yönlendirebilir. Artık **ptr** üzerinden işlemler, dizinin bu kısmında gerçekleştirilebilir.

1. Dizinin tüm elemanlarına göstergelerle erişim

C dilinde bir dizi tanımlandığında aslında dizinin ismi, dizinin bellekteki ilk elemanının adresini temsil eden sabit bir gösterge olarak oluşturulur. Bu, dizi elemanlarına erişimi hızlı ve etkili hale getirir. Dizinin herhangi bir elemanına erişim işlemi, derleyici tarafından otomatik olarak gösterge aritmetiğine dönüştürülür. Örneğin int sayi[5]; şeklindeki bir tanımlama bellekte aşağıdaki şekilde bir alan oluşturur.



Burada **"sayi"** ismi oluşturulan dizinin ilk elemanını gösteren göstergedir. Yani int *sayi = &sayi[0]; şeklinde bir tanım yapılmış gibidir. Böyle bir dizinin elemanlarına erişmek için gösterge kullanılmak istediğimizde aşağıdaki tanımlamanın yapılması yeterlidir.

```
int *sayiptr;
sayiptr = sayi; // İlk elemanı işaret eder
```

Bu durumda **"sayiptr"** göstergesi **"sayi"** dizisinin ilk elemanın adresini sayiptr→ 6000 şeklinde bellekte tutacaktır. Artık bu göstergeyi kullanarak dizinin diğer elemanlarına kolaylıkla erişebiliriz. Aşağıdaki kod ile dizinin ilk elemanından son elemanına kadar bütün dizi elemanları yazdırılmaktadır.

```
for (int i = 0; i < 5; i++) {
   printf("Eleman %d: %d\n", i, *(sayiptr + i));
}
```

2. Gösterge kullanarak dizi elemanlarını güncelleme

Bir dizinin elemanlarını indeks numaraları kullanarak güncelleyebilmemize rağmen gösterge kullanarak dizi elemanların güncellenmesi hem performans hem de kod yazımının daha kısa sürmesi bakından daha avantajlıdır. Özellikle çok elemanlı diziler üzerinde yapılan işlemler için gösterge kullanımı daha fazla verimlilik sağlar. Diğer avantajlı durumlar şu şekilde sıralanabilir.

Bellek üzerinde daha doğrudan kontrol: Gösterge, bellekte doğrudan erişim sağladığı için daha esnek bir güncelleme mekanizması sunar. Göstergeyi dizinin herhangi bir elemanını gösterecek şekilde ayarlamak ve doğrudan bu eleman üzerinde işlem yapmak mümkündür. Örneğin:

```
int dizi[5] = {10, 20, 30, 40, 50};
int *ptr = &dizi[2]; // Dizinin üçüncü elemanını gösteriyor
*ptr = 100;          // Üçüncü elemanı güncelle
```

İşlemlerin basitleştirilmesi (Döngülerde Kolaylık): Gösterge, bir dizi elemanını işlemden sonra bir sonrakine doğrudan geçmeyi sağlar. Döngülerde indeks artırmak yerine göstergenin artırılması, hem okunabilirliği artırır hem de yazımı kolaylaştırır. Örneğin:

```
int dizi[5] = {10, 20, 30, 40, 50};
int *ptr = dizi; // Dizinin ilk elemanını gösteriyor
for (int i = 0; i < 5; i++) {
    *ptr += 5;   // Her elemanı 5 artır
    ptr++;       // Bir sonraki elemana geç
}
```

Dinamik verilerle uyumlu olması: Gösterge kullanımı, dinamik olarak tahsis edilmiş dizilerle çalışırken zorunludur. Dinamik bellek tahsisi sırasında dizinin başlangıç adresi bir göstergeye atanır ve tüm işlemler bu gösterge üzerinden yapılır. İndeks numarası yalnızca sabit boyutlu dizilerde etkili bir yöntem olup dinamik dizilerde verimli değildir.

Fonksiyon parametresi olarak göstergeler

C dilinde bir fonksiyona parametreler genel olarak iki şekilde aktarılmaktadır: değerle (call by value) ve referansla (call by reference). Bu seçim fonksiyona aktarılacak değerlerin büyüklüğüne göre yapılmaktadır. Çünkü büyük ölçekli parametreler üzerinde yapılan işlemler küçük ölçekli parametre göre daha uzun zaman alabilmektedir.

Değerle çağırım

Bu yöntemde, fonksiyona gönderilen parametrenin bir kopyası oluşturulur ve fonksiyon bu kopya parametre üzerinde işlem yapar. Orijinal değişkenle fonksiyon içinde kullanılan değişken birbirinden bağımsızdır, bu nedenle fonksiyon içinde yapılan değişiklikler yalnızca kopya üzerinde etkili olur. Bu özellik, orijinal değişkenin değiştirilmesini istemediğiniz durumlarda büyük bir avantaj sağlar. Böylece verilerin güvenli bir şekilde taşınmasını sağlanır ve programın istenmeyen hatalara karşı daha korunaklı olmasını temin edilir. Ancak, parametre olarak çok büyük bir veri yapısı gönderildiğinde, her seferinde bu verinin kopyalanması bellekte fazla yer kaplayabilir ve performans sorunlarına yol açabilir. Aşağıdaki örnek kod değer gönderimi yöntemini göstermektedir.

```c
#include <stdio.h>
void degerleDegistir(int x) {
    x = 10;        // Bu işlem sadece yerel x değişkenini değiştirir
    printf("Fonksiyon içinde x: %d\n", x);
}

int main() {
    int a = 5;
    printf("Main fonksiyonunda a: %d\n", a);
    degerleDegistir(a);      // a'nın kopyası fonksiyona gönderilir
    printf("Main fonksiyonunda a (değişmeden): %d\n", a);
    return 0;
}
```

```
Main fonksiyonunda a: 5
Fonksiyon içinde x: 10
Main fonksiyonunda a (değişmeden): 5
```

Referansla çağırım

Bu yöntem parametre olarak gösterge kullanımıdır. Bu kullanımda parametrenin kendisi değil, bellekteki adresini işaret eden bir gösterge fonksiyona gönderilir. Böylece, fonksiyon bu gösterge üzerinden doğrudan orijinal veriye erişebilir ve onun üzerinde değişiklik yapabilir. Bu işlem, özellikle orijinal verinin güncellenmesi gerektiğinde veya verinin bir kopyasını oluşturmanın gereksiz maliyetli olduğu durumlarda büyük avantaj sağlar. Eğer büyük boyutlu bir veri kopyalanarak fonksiyona gönderilseydi, bu işlem hem zaman hem de bellek açısından maliyetli olurdu. Ancak referansla gönderimde yalnızca verinin adresi taşındığı için bellekten tasarruf edilir ve işlem hızı artar.

```c
#include <stdio.h>
void referanslaDegistir(int *x) {
    *x = 10;        // Bu işlem orijinal x değişkenini değiştirir
    printf("Fonksiyon içinde x: %d\n", x);
}

int main() {
    int a = 5;
    printf("Main fonksiyonunda a: %d\n", a);
    referanslaDegistir(&a);      // a'nın adresi fonksiyona gönderilir
    printf("Main fonksiyonunda a (değişmiş): %d\n", a);
    return 0;
}
```

```
Main fonksiyonunda a: 5
Fonksiyon içinde x: 10
Main fonksiyonunda a (değişmiş): 10
```

Referansla çağırım performans ve verimliliği arttırmakla beraber bu yöntemi kullanırken dikkatli olunması gerekir; çünkü fonksiyon, orijinal veriyi değiştirebileceği için beklenmeyen hatalar meydana gelebilir. Bu nedenle, programlama sırasında referansla gönderim yapılan parametrelerin doğru bir şekilde kontrol edilmesi, hataların önlenmesi açısından önemlidir.

Çalışma soruları

1. Bir tamsayı değişken tanımlayıp, bu değişkenin değerini bir gösterge aracılığıyla iki katına çıkaran C kodu yazın.

2. Başlangıç değerleriyle tanımlanan bir tamsayı dizisinin elemanlarını gösterge kullanarak, "Dizinin elemanları:" şeklindeki başlığın altına "1. eleman: ", "2. eleman: " şeklinde sırayla terminal yazdıran C kodunu yazınız. Dizinin eleman sayısı 5-6 civarında ve birbirinden farklı olarak kabul edilsin.

3. Başlangıç değerleriyle tanımlanan bir tamsayı dizisinin elemanlarını terminale tersten yazdıran C kodunu yazınız. Tersten yazdırma işlemi fonksiyon ile ve gösterge kullanılarak yapılacak. Dizinin eleman sayısı 8-10 civarında ve birbirinden farklı olarak kabul edilsin.