10. İŞSARETÇİLER (POINTERS)

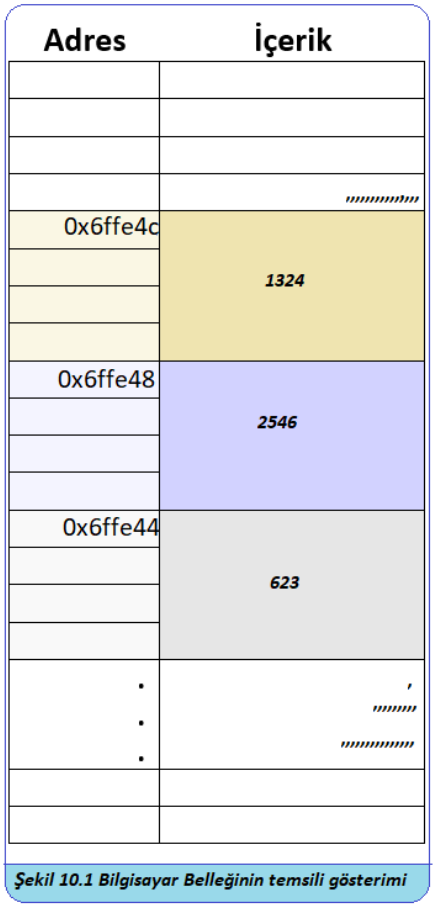
Giriş

Bazı kaynaklarda işaretçiler konusu ***Göstericiler*** başlığı altında incelenmektedir. Konuyla ilgili farklı kaynaklara müracaat edildiğinde bir karışıklığa yol açmamak için, daha başlangıçta bu hususu vurgulamakta fayda olduğu düşünüldüğünden konuya bu şekilde bir açıklama cümlesiyle giriş yapıldı.

İşaretçiler konusu C++ ve C programlama dillerinin en popüler konularından biri, belki de en popüler konusudur. Ayrıca işaretçiler, başka programlama dillerinde yer almayan, başka bir deyişle başka programlama dilleri tarafından desteklenmeyen bir özelliktir.

İşaretçiler konusunda, bilgisayar programlamaya yeni başlayanlar arasında, "*işaretçileri öğrenmek zordur*" şeklinde yaygın, yanlış bir kanaat vardır. Ancak burada şunu belirtmeliyiz; C++ 'in diğer konularına göre, işaretçiler konusunu öğrenmek için biraz daha fazla zaman ayırmanız gerekebilir.

Şimdi asıl konumuza gelip işaretçi nedir? Ne işe yarar? Sorularının cevaplarını arayalım. İşaretçi, içerisinde başka değişkenlerin adresini tutan bir değişkendir. Bilgisayar belleğinde her bir Byte 'ın bir adresi vardır ve bu adresler heksadesimal (16 Tabanlı Sayı Sisteminde) sayılarla ifade edilir. Şekil 10.1'de Bilgisayar belleği temsili olarak gösterilmiştir.



Şekil 10.1 Bilgisayar Belleğinin temsili gösterimi

```
#include <iostream>
using namespace std;
int main()
{
    int sayi1=1324,sayi2=2546,sayi3=623;
    cout<<&sayi1<<endl;
    cout<<&sayi2<<endl;
    cout<<&sayi3<<endl;
    return 0;
}
```

Bilgisayar Belleğinde çalışan Program

```
0x6ffe4c
0x6ffe48
0x6ffe44

--------------------------------
Process exited after 0.04614 seconds with return value 0
Press any key to continue . . .
```

Bellekte Çalışan Programın Çıktısı

Programların derlenmesi sırasında derleyici, program içerisinde tanımlanmış değişkenler için, değişkenin boyutuna göre, bilgisayar belleğinde yer ayırır. Örneğin program içerisinde ***char*** tipinde bir değişken tanımlanmış ise bu değişken için bellekte bir Byte yer ayrılır, ***int*** tipinde bir değişken tanımlanmış ile bilgisayar belleğinde dört Byte yer ayrılır, Eğer double tipinde bir değişken tanımlanmışsa bu defa bellekte sekiz Byte yer ayrılır. Bir Byte 'tan fazla yer ayrılan değişkenlerin adresi, o değişken için ayrılan ilk Byte 'ın adresidir. Yukarıda verilen örnekte üç tane ***int*** tipinde değişken tanımlanmış, Bu değişkenler için bilgisayar belleğinde dörder Byte 'lık yer ayrılmış ve değişkenlerin adresleri olarak, ayrılan dörder Byte 'tan ilk Byte 'ların adresi olduğu gösterilmiştir. Bu adreslerin heksadesimal olarak ifade edildiğine dikkat ediniz.

C++'ta işaretçileri kullanmadan da program yazabiliriz. Nitekim bundan önceki bölümlerde böyle programlar yazdık. Ancak daha hızlı çalışan programlar yazmak, sistemden bellek almak, veri yapılarını tanımlamak gibi işlemlerde işaretçilerin kullanılmasının önemli ölçüde yararı vardır. C ve C++'ın belki de en güçlü yanı işaretçileri kullanmasından kaynaklanmaktadır.

Bu bölümde adres operatörünün kullanılmasını, işaretçilerin tanımlanması ve kullanılmasını, dizilerde ve fonksiyonlarda işaretçilerin kullanılmasını öğreneceğiz.

10.1. Adres Operatörü (&)

Adres operatörü, sabitlerin veya değişkenlerin bellekte hangi adreslere yerleştiklerini öğrenmek için kullanılır. Adres operatörünün simgesi (&) işaretidir. Bir değişken veya sabitin adresini öğrenmek için değişkenin önüne **&** yazılır. Örneğin program içerisinde tamsayı tipinde ***sayi1*** isimli bir değişkenimiz olduğunu varsayalım. Bu değişkenin adresini öğrenmek için:

&sayi1

yazmamız yeterlidir. Program 10.1'de tamsayı (***int***) tipinde üç değişken tanımlanmıştır. Daha sonra adres operatörü kullanılarak bu değişkenlerin bilgisayar belleğinde kullandıkları adresler ekrana yazdırılmıştır. Program 10.1'in aşağıda verilen ekran çıktısı incelendiğinde, değişkenlerin ekrana yazdırılan adreslerinin heksadesimal olarak ifade edildiği (**0x6ffe4c, 0x6ffe48, 0x6ffe44**) görülecektir. Dikkat edilirse ekrana yazdırılan adresler dörder Byte olarak azalan sıradadır. Bunun nedeni yerel değişkenlerin belleğin aşağı yönde büyüyen kısmında saklanmasıdır. Program 10.1'deki değişkenler global değişken olarak tanımlansaydı adresleri artan sırada olacaktı. Çünkü global değişkenler belleğin yukarı yönde büyüyen kısmında saklanır.

```
1   #include <iostream>
2   using namespace std;
3   int main() {
4       int sayi1=3;
5       int sayi2=5;
6       int sayi3=7;
7       cout<<"  "<<&sayi1 <<endl;
8       cout<<"  "<<&sayi2 <<endl;
9       cout<<"  "<<&sayi3 <<endl;
10      return 0;
11  }
```

Adres Operatörü

Adresi Ekrana yazılacak Değişken

Program 10.1 Adres Operatörü

```
0x6ffe4c
0x6ffe48
0x6ffe44

--------------------------------
Process exited after 4.348 seconds with return value 0
Press any key to continue . . .
```

sayi1, sayi2 ve sayi3 değişkenleri için ayrılan bellek adresleri

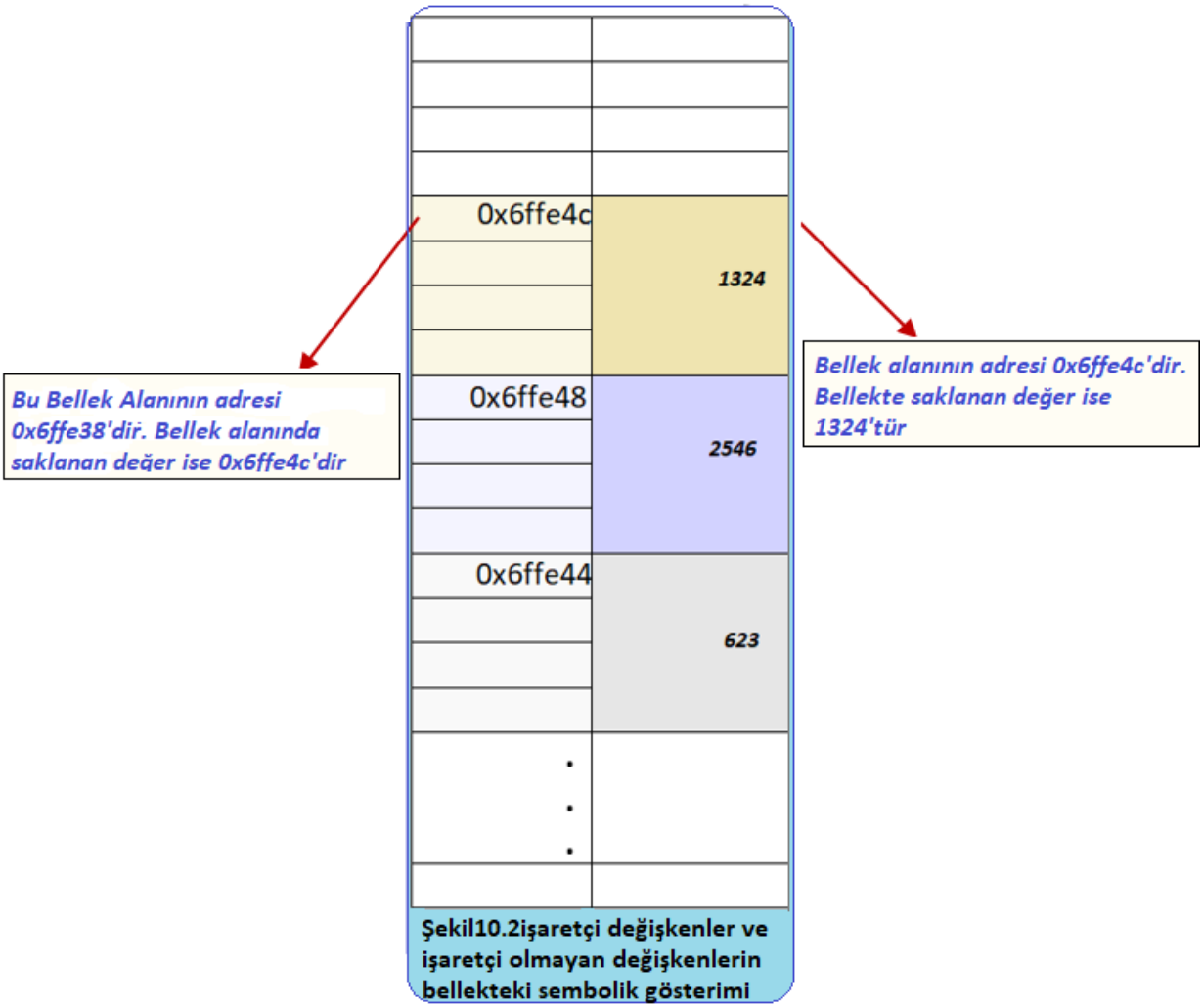
Program 10.1'in Ekran Çıktısı

Burada şu iki hususu vurgulamakta yarar var. Bu hususlardan birincisi derleyicinin sabitler veya değişkenler için belleğin hangi adreslerinin kullanacağını bizim müdahalemiz olmadan, bizim adımıza takip etmesidir. Vurgulanması gereken ikinci husus ise, muhtemelen program 10.1'i siz kendi bilgisayarınızda yazıp çalıştırdığınızda, ekranınızda değişkenlerin adresleri muhtemelen farklı olacaktır. Çünkü sabit ve değişkenler için hangi adreslerin kullanılacağı, bilgisayarınızın fiziksel özellikleri, kullandığınız işletim sistemi, sizin programı çalıştırdığınızda bilgisayarınızda o anda çalışan diğer programlar gibi parametrelere bağlıdır.

10.2. İşaretçilerin Tanımlanması ve Kullanılması

İşaretçi değişkenlerinin tanımlanması diğer değişkenlerin tanımlanmasına benzer. Ancak bir işaretçi değişkeni tanımlanırken değişkenin başına asteriks (*) işareti konulur (*Bkz. Program 10.2 satır 6*). Bu işaretin adı işaretçi operatörüdür. Bir program içerisinde tanımlanmış değişkenin, işaretçi değişken olup olmadığına değişkenin tanımlanması sırasında başına işaretçi operatörü konulup konulmadığına bakılarak karar verilir.

İşaretçi değişkenleri adres değerlerini tutan değişkenlerdir. Bir adres değerini tutan değişkene işaretçi değişkeni ya da kısaca işaretçi denir. İşaretçiler tanımlanırken değişkenlerin tanımlanması sırasında olduğu gibi tipleri (***char, int, float vb.***) belirlenir. Ancak belirlenen bu tipler işaretçinin tuttuğu değerin türünü belirtmez. Yukarıda da belirlendiği gibi işaretçiler her zaman adres bilgisi tutar. Yani işaretçi değişkenin char, int, float vb. olması işaretçinin içeriği ile ilgili değildir. Ancak ileride açıklanacak bir istisnayı bir kenara bırakacak olursak işaretçiler adreslerini tutacakları değişkenlerin tipleri ile aynı tipte tanımlamalıdır. Yani işaretçinin adresini tutacağı değişken ***int*** ise işaretçinin tipi de ***int*** olmalı, değişken ***float*** ise işaretçi de ***float*** olarak tanımlanmalıdır. İşaretçi tanımında kullanılan char, int, float vb. tip tanımlamaları, işaretçinin tuttuğu adreste bulunan verinin tipi ile doğrudan ilgilidir. ***Şekil 10.2***'de işaretçi değişkenler ve işaretçi olmayan değişkenler ve bunların değerleri gösterilmiştir.



```
1   #include <iostream>
2   using namespace std;
3   int sayi1,sayi2;
4   int main() {
5       int sayi3,sayi4;
6       int* i1, *i2,*i3,*i4;
7       i1=&sayi1;i2=&sayi2;
8       i3=&sayi3;i4=&sayi4;
9       cout<<i1<<endl;
10      cout<<i2<<endl<<endl;
11      cout<<i3<<endl;
12      cout<<i4<<endl;
13      return 0;
14  }
```

Global Değişkenler

Yerel değişkenler

İşaretçi değişkenlerin tanımlanması

Değişkenlerin Adreslerinin işaretçi değişkenlere atanması

İşaretçi değişkenlerin değerlerinin ekrana yazdırılması

Program 10.2a İşaretçi Değişkenleri

```
0x4a7030
0x4a7034

0x6ffe2c
0x6ffe28

--------------------------------
Process exited after 3.314 seconds with return value 0
Press any key to continue . . .
```

İşaretçi değişkenlerin değerleri

Program 10.2'a'nın Ekran Çıktısı

Program 10.2a'nın üçüncü satırında iki global değişken (sayi1, sayi2) ve beşinci satırında iki yerel değişken (sayi3, sayi4) tanımladık. Programın 6. satırında dört adet işaretçi tanımladık (i1, i2, i3, i4). Yedinci ve sekizinci satırlarda global ve yerel değişkenlerin adreslerini işaretçi değişkenlere atadık. Dokuz, on, on bir ve on ikinci satırlarda işaretçi değişkenlerin değerlerini ekrana yazdırdık. Bkz. Program 10.2a'nın ekran çıktısı. Ekrana yazdırılan bu adreslerin işaretçi değişkenlerin değerleri olduğunu ve bu heksadesimal değerlerin sırası ile global ve yerel değişkenlerin adresleri olduğunu unutmayınız. Program 10.2a'nan ekran çıktısı incelendiğinde global değişkenlerin adreslerinin artan sırada yerel değişkenlerin adreslerinin ise azalan sırada olduğu görülmektedir.

```
2   #include <locale.h>
3   using namespace std;
4   int main() {
5       setlocale(LC_ALL,"Turkish");
6       int s1=1324,s2=2546,s3=623;
7       int *ss1, *ss2, *ss3;
8       ss1=&s1;
9       ss2=&s2;
10      ss3=&s3;
11      cout<<"İşaretçinin Adresi     İşaretçinin İçinde Tuttuğu Adres"<<endl;
12      cout<<"--------------------+--------------------------------+"<<endl;
13      cout<<&ss1<<"            +  "<<ss1<<endl;
14      cout<<&ss2<<"            +  "<<ss2<<endl;
15      cout<<&ss3<<"            +  "<<ss3<<endl;
16      cout<<"--------------------+--------------------------------+"<<endl;
17      return 0;
18  }
```

```
İşaretçinin Adresi     İşaretçinin İçinde Tuttuğu Adres
--------------------+--------------------------------+
0x6ffe28            +  0x6ffe3c
0x6ffe20            +  0x6ffe38
0x6ffe18            +  0x6ffe34
--------------------+--------------------------------+
```

program 10.2b'nin ekran çıktısı

***Program 10.2b** İşaretçilerin adresleri ve işaretçilerin içinde tuttukları adresler*

Program 10.2b'den görülebileceği gibi, işaretçi değişken işaretçi olmayan bir değişkenin adresini tutarken aynı zamanda işaretçi değişkenin de kendine ait bir adresi vardır. Başka bir şekilde ifade etmek gerekirse, her işaretçi değişkeni, birincisi işaretçinin içerisinde tuttuğu adres, ikincisi işaretçinin bellekte tutulduğu adres olmak üzere iki adrese sahiptir. Bu adresleri birbirine karıştırmamak gerekir.

Bu bölümün buraya kadar ele aldığımız kısmında, işaretçi değişkenlerin tanımlanmasını, işaretçi değişkenlerin adres bilgilerini nasıl tuttuklarını ve adres bilgilerine nasıl ulaşabildiğimizi öğrendik. Şimdi işaretçi değişkenlerin hangi nedenlerle kullanıldığını öğreneceğiz.

```
1  #include <iostream>
2  #include <locale.h>
3  using namespace std;
4  int main() {
5      setlocale(LC_ALL,"Turkish");
6      int sayi=120;
7      int *isr=&sayi;
8      cout<<"sayi degşkeninin adresi adres operatörü ile yazdırıldı     : "<<&sayi<<endl;
9      cout<<"sayi degişkeninin adresi işaretçi değişkeni ile yazdırıldı : "<<isr<<endl;
10     return 0;
11 }
```

Program 10.3 İşaretçi Değişken

```
sayi degşkeninin adresi adres operatörü ile yazdırıldı     : 0x6ffe44
sayi degişkeninin adresi işaretçi değişkeni ile yazdırıldı : 0x6ffe44

--------------------------------
Process exited after 0.07254 seconds with return value 0
Press any key to continue . . .
```

Program 10.3' ün Ekran Çıktısı

İşaretçiler, adreslerini tuttukları değişkenler üzerinde işlem yapmak için kullanılırlar. Program 10.3'te Altıncı satırda ***sayi*** değişkeni tanımlandı ve değişkene 120 değeri atandı. Yedinci satırda ***isr*** isimli bir işaretçi tanımlandı ve bu işaretçi değişkenine sayi değişkeninin adresi atandı.

```
6      int sayi=120;
7      int *isr=&sayi;
```

Bu işlemlerin sonunda şekil 10.3'ten de görülebileceği gibi ***isr*** işaretçi değişkeni bellekteki ***sayi*** değişkenine işaret eder.

Şekil 10.3 *isr* değişkeni *sayi* değişkenini işaret ediyor

isr işaretçi değişkeninin ***sayi*** değişkenine işaret etmesinden yararlanılarak sayi değişkeni kullanılmadan işaretçi değişkeni kullanılarak sayi değişkeninin değeri değiştirilebilir. Program 10.3b'yi inceleyiniz.

```
1   #include <iostream>
2   #include <locale.h>
3   using namespace std;
4   int main() {
5       setlocale(LC_ALL,"Turkish");
6       int sayi=120;
7       int *isr=&sayi;
8       cout<<"Sayı Değişkeninin ilk  değeri = "<<sayi<<endl;
9       *isr=987;
10      cout<<"Sayı Değişkeninin yeni değeri = "<<sayi<<endl;
11      return 0;
12  }
```

Program 10.3b İşaretçi değişkeni kullanarak sayi değişkeninin değerini değiştirmek

```
Sayı Değişkeninin ilk  değeri = 120
Sayı Değişkeninin yeni değeri = 987

--------------------------------
Process exited after 4.205 seconds with return value 0
Press any key to continue . . .
```

Program 10.3b'nin Ekran Çıktısı

Programın dokuzuncu satırındaki ifade ile sayi değişkeninin değeri 987 yapılmıştır.

```
9       *isr=987;
```

Yukarıdaki ifadede ***isr*** işaretçinin önündeki * operatörü(içerik operatörü), ***sayi*** değişkeninin işaret ettiği adrese karşılık geldiği için ***sayi*** değişkeninin kullanılması gerektiği her noktada ****isr*** ifadesini kullanabiliriz. Örneğin, sayi değişkeninin değerini ***s1*** isimli bir değişkene kopyalamak istediğimizde aşağıdaki ifade yazabilir. Bunun için Program 10.3c'yi inceleyiniz.

```
int s1=*isr;
```

```
1   #include <iostream>
2   #include <locale.h>
3   using namespace std;
4   int main() {
5       setlocale(LC_ALL,"Turkish");
6       int sayi=120,s1;
7       int *isr=&sayi;
8       cout<<"Sayı Değişkeninin ilk  değeri = "<<sayi<<endl;
9       *isr=987;
10      cout<<"Sayı Değişkeninin yeni değeri = "<<sayi<<endl;
11      s1=*isr;
12      cout<<"s1 Değişkeninin değeri        = "<<s1<<endl;
13      return 0;
14  }
```

Program 10.3c Gösterici değişkeni kullanarak diğer değişkenlere değer atamak

```
Sayı Değişkeninin ilk  değeri = 120
Sayı Değişkeninin yeni değeri = 987
s1 Değişkeninin değeri        = 987

--------------------------------
Process exited after 3.71 seconds with return value 0
Press any key to continue .
```

Program 10.3c'nin Ekran Çıktısı

Program 10.3c'nin Program 10.3b'den farkı, Program 10.3c'ye On bir ve on ikinci satırların eklenmiş olmasıdır. On birinci satırda içerik operatörü kullanılarak ***sayi*** değişkeninin değeri ***s1*** değişkenine atanmış ve on ikinci satırda ***s1*** değişkeninin değeri ekranda gösterilmiştir.

İşaretçiler başlığı altında incelediğimiz bu bölümün buraya kadar olan kısmında üç tane operatör kullandık. Bu operatörlerden birincisi Adres Operatörüdür (**&**). Adres operatörünü, değişkenlerin bellekteki adreslerini göstermek için kullandık. Operatörlerin ikincisi, İşaretçi tanımlamam operatörüdür (*). İşaretçi tanımlama operatörünü, değişkenlerin tanımlanması sırasında, tanımlanan değişkenin işaretçi değişkeni olacağını belirtmek için kullandık. Kullandığımız üçüncü operatör ise içerik operatörüdür (*). Asteriks(*) işareti tanımlama satırında kullanılmıyorsa içerik operatörüdür. İçerik operatörünü, işaretçinin işaret ettiği değişkenin değerlerini maniple etmek için kullandık. Burada dikkat edilmesi gereken en önemli husus işaretçi tanımlama operatörü ile içerik operatörünü karıştırmamaktır.

10.3. void İşaretçiler

Bu bölümde şu ana kadar bir işaretçinin içine yerleştirdiğimiz adres, işaretçi ile aynı tipteydi. Yani ***int***' a işaret eden bir işaretçiye ***int*** bir değişkenin adresini, ***float***' a işaret eden bir işaretçiye ***float*** bir değişkenin adresini, vb. atadık. Bu konunun bir istisnası vardır. ***void*** tipindeki işaretçiler herhangi bir veri tipine işaret edebilir. Bu nedenle ***void*** tipi işaretçilere genel amaçlı işaretçiler denir.

Burada dikkat edilmesi gereken en önemli husus, ***void*** işaretçilerin türü olmadığı için içerik operatörü olarak kullanılamayacakları hususudur. Yani ***void*** işaretçiler bir değerin işaretçisi olamazlar ve ayrıca aritmetik işlemlerde kullanılamazlar.

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4      int ins;
5      float fos;
6
7      int* isri;
8      float* isrf;
9      void* isrv;
10
11     isri = &ins;
12     isrf=&fos;
13
14     isrv=&ins;
15     cout<<isrv<<endl;
16     isrv=&fos;
17     cout<<isrv<<endl;
18     return 0;
19 }
```

Program 10.4 void işaretçiler

```
0x6ffe34
0x6ffe30

--------------------------------
Process exited after 0.05195 seconds with return value 0
Press any key to continue . .
```

Program 10.4'ün Ekran çıktısı

Program 10.4'te dördüncü satırda ***int*** türünde, beşinci satırda ***float*** düründe iki değişken tanımlanmıştır. Yedi, sekiz ve dokuzuncu satırlarda sırasıyla ***int, float, void*** işaretçiler tanımlanmıştır. On bir ve on ikinci satırlarda sırasıyla ***int*** ve ***float*** değişkenlerin adresleri yine sırasıyla ***int*** ve ***float*** işaretçi değişkenlere atanmıştır. Aynı şekilde on dört ve on altıncı satırlarda ***int*** ve ***float*** değişkenlerin adresleri ***void*** işaretçiye aktarılmış ve bu adresler ekrana yazdırılmıştır. ***Program 10.4*** ve program ***10.4'ün ekran çıktısını*** inceleyiniz.

Program 10.4'te ki uygulamadan ***void*** tipindeki işaretçilerin herhangi bir veri tipine işaret edebildiği görülmektedir.

10.4. İşaretçi Aritmetiği

Değişkenler üzerinde aritmetiksel işlemler yapılabildiği gibi, işaretçiler üzerinde de aritmetiksel işlemler yapılabilir. Örneğin işaretçi değişkenleri tamsayılarla arttırılıp azaltılabilir. Değeri arttırılan bir işaretçi bellekte bir sonraki alanı işaret eder. Değeri azaltılan bir işaretçi ise bellekte bir önceki alanı işaret eder.

Program 10.5'te dördüncü satırda beş elemanı olan ***int*** türünde bir dizi tanımlanmış ve dizi elemanlarına ilk değer ataması yapılmıştır. Programın beşinci satırında bir işaretçi değişken tanımlanmış ve altıncı satırda dizinin ilk elemanının adresi işaretçi değişkene atanmıştır. Yedinci satırda işaretçi değişkenin değeri ve işaret ettiği değişkenin değeri (dizinin 0. Elemanın değeri) yazdırılmış, sekizinci satırda arttırma operatörü kullanılarak işaretçi değişkenin değeri arttırılmıştır. Dokuzuncu satırda işaretçi değişkenin yeni değeri ve işaret ettiği değişkenin değeri (dizinin 1. Elemanın değeri) yazdırılmıştır.

Program 10.5'in ekran çıktısı incelendiğinde görülebileceği gibi, ***isr*** değişkeni tamsayı türünden bir işaretçi olduğu için işaretçi bir arttırılmış buna karşılık adres dört artmıştır.

```
1   #include <iostream>
2   using namespace std;
3   int main() {
4       int dizi[]={1,3,5,7,9};
5       int *isr;
6       isr=dizi;
7       cout<<isr<<" Adresindeki Değer = "<<*isr<<endl;
8       ++isr;
9       cout<<isr<<" Adresindeki Değer = "<<*isr<<endl;
10      return 0;
11  }
```

Program 10.5 İşaretçilerin arttırılıp azaltılması

Adreslerin dörder Byte olarak ilerlediğine dikkat ediniz

önce dizinin 0. elemanı sonra 1. elemanı yazdırıldı

```
0x6ffe20 Adresindeki Deger = 1
0x6ffe24 Adresindeki Deger = 3

--------------------------------
Process exited after 2.573 seconds with return value 0
Press any key to continue .
```

Program 10.5'in Ekran Çıktısı

10.5. İşaretçiler ve Diziler

Bölüm 6'da Diziler konusunu ele almıştık ve bir dizi tanımlanırken önce dizide yer alacak elemanların tipini, sonra ismini ve daha sonra da dizinin eleman sayısını(boyutunu) belirliyorduk. Örneğin 5 elemanlı, int

tipinde sayıları tutacak bir diziyi aşağıdaki gibi tanımlıyorduk.

```
int dizi[5];
```

Bu tanımlama bellekte program için araka arkaya 4 Byte'lık 5 adet alan ayrılmasını sağlıyordu. Bu alanların adreslerine adres operatörü ile ulaşabiliriz.

Dizinin Elemanlarının Sırası	Dizinin Elemanları	Dizi Elemanlarının Adresi
İlk eleman	dizi[0]	&dizi[0]
İkinci eleman	dizi[1]	&dizi[1]
Üçüncü eleman	dizi[2]	&dizi[2]
Dördüncü eleman	dizi[3]	&dizi[3]
Beşinci Eleman	dizi[4]	&dizi[4]

Tablo 10.1 Dizi Elemenlarının Adreslerine Adres operatörü ile erişim

İşaretçilerle diziler arasındaki ilişkiyi, ***bir dizinin ismi, o dizinin ilk elemanının adresini tutan bir göstericidir*** şeklinde ifade edebiliriz. Dolayısı ile bu tanımdan faydalanılarak dizi elemanlarının adreslerinin iki farklı şekilde yazdırılabileceğini söyleyebiliriz. Tablo 10.2'de dizi elemanlarının adreslerinin iki farklı şekilde nasıl yazdırılacağı ve dizi elemanlarının değerlerine, 6.bölümden bildiğimiz dizi operatörü ile ulaşılmasından farklı olarak, nasıl erişilebileceği gösterilmiştir.

Dizi Elemanlarının Sırası	Dizi Elemanlarının Adresi (1)	Dizi Elemanlarının Adresi (2)	Dizi Elemanlarının Değeri
İlk Eleman	&dizi[0]	dizi	*dizi
İkinci Eleman	&dizi[1]	dizi + 1	*(dizi + 1)
Üçüncü Eleman	&dizi[2]	dizi + 2	*(dizi + 2)
Dördüncü Eleman	&dizi[3]	dizi + 3	*(dizi + 3)
Beşinci Eleman	&dizi[4]	dizi + 4	*(dizi + 4)

Tablo 10.2 Dizi Elemanlarının Adreslerine ve Değerlerine Erişim

Program 10.6 dizi elemanlarının adreslerini ve dizi elemanlarını değerlerini ekrana yazdırmak için yazılmıştır.

Siz Program 10.6'yı *for* döngüsü veya *for* döngüleri kullanarak yeniden yazınız.

```cpp
#include <iostream>
#include <locale.h>
using namespace std;
int main() {
    setlocale(LC_ALL,"Turkish");
    int dizi[5]={12,4,7,19,11};
    cout<<"\n--Adres Operatörü ile Dizi Elamanlarının Adresinin Yazdırılması--\n";
    cout<<&dizi[0]<<endl;
    cout<<&dizi[1]<<endl;
    cout<<&dizi[2]<<endl;
    cout<<&dizi[3]<<endl;
    cout<<&dizi[4]<<endl;
    cout<<"\n--Dizi Adını(işaretçi) kullanarak Dizi Elemanlarının Adresinin Yazdırılması--\n";
    cout<<dizi<<endl;
    cout<<dizi+1<<endl;
    cout<<dizi+2<<endl;
    cout<<dizi+3<<endl;
    cout<<dizi+4<<endl;
    cout<<"\n--İşaretçi kullanarak Dizi Elemanlarının Değerlerinin Yazdırılması--\n";
    cout<<*(dizi)<<endl;
    cout<<*(dizi+1)<<endl;
    cout<<*(dizi+2)<<endl;
    cout<<*(dizi+3)<<endl;
    cout<<*(dizi+4)<<endl;
    return 0;
}
```

Program 10.6 Farklı İki Yöntemle Dizi Elemenlarının Adresini ve İşaretçi Kullanarak Dizi Elemanlarının Değerlerini Yazdırmak

```
--Adres Operatörü ile Dizi Elamanlarının Adresinin Yazdırılması--
0x6ffe30
0x6ffe34
0x6ffe38
0x6ffe3c
0x6ffe40

--Dizi Adını(işaretçi) kullanarak Dizi Elemanlarının Adresinin Yazdırılması--
0x6ffe30
0x6ffe34
0x6ffe38
0x6ffe3c
0x6ffe40

--İşaretçi kullanarak Dizi Elemanlarının Değerlerinin Yazdırılması--
12
4
7
19
11
```

Program 10.6'nın Ekran Çıktısı

10.6. İşaretçiler ve Fonksiyonlar

Program 10.7'den de görülebileceği gibi bir fonksiyonu referans ile çağırmak, o fonksiyona parametre olarak bir gösterici göndermekle aynı anlama gelmektedir. Fonksiyonu referansla çağırırken fonksiyona gönderilen adres orijinal verinin adresidir (*Program 10.7'de konuyu açıklamak için basit bir değişken kullanılmıştır*). Dolayısıyla fonksiyona gönderilen adresteki veri üzerinde yapılan her değişiklik aslında orijinal veriyi değiştirir. Program 10.7'yi ve Program 10.7'nin ekran çıktısını inceleyiniz.

```
1   #include <iostream>
2   using namespace std;
3   int main() {
4       void cevir_m_cm(double*);
5       double metre=10.3;
6       cout<<" "<<metre <<" metre "<<endl;
7       cevir_m_cm(&metre);
8       cout<<" "<<metre <<" cm "<<endl;
9       return 0;
10  }
11  void cevir_m_cm(double* s_cm)
12  {
13      *s_cm*=100;
14  }
```

Program 10.7 Fonksiyona parametre olarak işaretçi göndermek

```
10.3 metre
1030 cm

--------------------------------
Process exited after 33.68 seconds with return value 0
Press any key to continue .
```

Program 107'ninEkran Çıktısı

Program 10.7'de ***main()*** fonksiyonu içerisinde ilk değer ataması yapılan ***double*** değişkenin adresi parametre olacak şekilde ***cevir_m_cm()*** çağırıldı. ***cevir_m_cm()*** fonksiyon içerisinde veri değiştirildi(). Daha sonra ***main()*** fonksiyonu içerisinde değişen veri gösterildi.

Basit değişkenlerin fonksiyonlara aktarılması sırasında olduğu gibi, fonksiyonlara dizi aktarılırken de işaretçileri kullanmak yaygın bir yaklaşımdır. Program 10.8, fonksiyonlara dizi aktarılırken işaretçilerin nasıl kullanılacağını açıklamak için yazılmıştır.

```
#include <iostream>
#include <locale.h>
using namespace std;
const int Eleman=6;
int main() {
    setlocale(LC_ALL,"Turkish");
    void cevir(double*);
    double dizi[Eleman]={11.3,9.8,5.5,12.5,7.8,14.6};
cout<<"Orijinal Dizi(Elemanlar metre)"<<endl;
    for(int i=0;i<Eleman;i++)
    cout<<dizi[i]<<"  ";
    cout<<endl<<endl;
    cevir(dizi);
cout<<"Fonksiyonda Elemanları santimetreye çevrilmiş Dizi"<<endl;
    for(int i=0;i<Eleman;i++)
    cout<<dizi[i]<<"  ";
    cout<<endl;
    return 0;
}
void cevir(double* d_el)
{
    for(int i=0;i<Eleman;i++)
    *(d_el+i)*=100;
}
```

Program 10.8 Dizileri Fonksiyonlara Aktarmak

```
Orijinal Dizi(Elemanlar metre)
11.3  9.8  5.5  12.5  7.8  14.6

Fonksiyonda Elemanları santimetreye çevrilmiş Dizi
1130  980  550  1250  780  1460
```

Program 10.8'in Ekran Çıktısı

Program 10.8'in sekizinci satırında ***dizi*** isminde altı elemana sahip ve elemanları double olan bir dizi tanımladık ve dizi elemanlarına metre cinsinden ilk değerler atadık. Onuncu satırda diziyi fonksiyona göndermeden for döngüsü ile dizinin elemanlarını ekrana yazdırdık. On üçüncü satırda diziyi, dizi ismini parametre olarak kullanarak cevir() fonksiyonuna gönderdik. Bu fonksiyon içerisinde içerik operatörünü kullanarak dizi elemanlarını santimetreye çevirdik. On beşinci satırda dizi elemanlarını santimetre olarak tekrar yazdırdık.

Bu örnekte iki hususa dikkat etmelisiniz. Bunlardan birincisi fonksiyona sadece dizinin ismi aktarıldı(Dizinin isminin, diziye ait ilk elemanın adresi olduğundan yararlandık). İkincisi ise dizinin son sınırı için hem main fonksiyonundan, hem de cevir fonksiyonundan ulaşılabilmesi için global değişken (***Eleman***) tanımladık.

Aşağıda verilen ***program 10.9***, elemanları küçük harf olarak atanmış bir karakter dizisini büyük harfe ve elemanları büyük harfe dönüştürülmüş bu dizinin elemanlarını tekrar küçük harfe dönüştürmektedir. Bunun için programda ***main()*** fonksiyonu dışında iki ayrı fonksiyon yazılmıştır. ***bharf()*** isimli fonksiyon kendisine gelen ve elemanları küçük harf olan diziyi büyük harfe çevirmektedir. ***kharf()*** isimli fonksiyon ise bu işlemin tam tersini yapmaktadır.

```
#include <iostream>
using namespace std;
void bharf(char*,int);
void kharf(char*,int);
int main() {
    char dizi[6]={'a','n','K','a','r','a'};
    int i=6;
    bharf(dizi,i); //dizinin elmanlarını büyük harfe çevirmek için
    for(i=0;i<6;i++)
    cout<<dizi[i]<<"  ";
    cout<<endl;
    kharf(dizi,i);//Büyük harfe dönüşen dizi elemanlarını küçük harfe çevirmek için
    for(i=0;i<6;i++)
    cout<<dizi[i]<<"  ";
    cout<<endl;
    return 0;
}
void bharf(char *kdizi, int ii){          //Küçük harfi
    for(int i=0;i<ii;i++){                //büyük harfe
        if(kdizi[i]>='a' && kdizi[i]<='z')//çeviren fonksiyon
            kdizi[i]=kdizi[i]-32;
        }
}
void kharf(char *bdizi, int ii){           //Büyük harfi
    for(int i=0;i<ii;i++){                 //küçük harfe
        if(bdizi[i]>='A' && bdizi[i]<='Z') //çeviren fonksiyon
        bdizi[i]=bdizi[i]+32;
        }
}
```

Program 10.9 küçük harf verlen diziyi büyük harfe, büyük harf verilen diziyi küçük harfe çevirmek

```
A  N  K  A  R  A
a  n  k  a  r  a
```

Program 10.9'un Ekran Çıktısı

```
void bharf(char *kdizi, int ii){
    for(int i=0;i<ii;i++){
        if(kdizi[i]>='a' && kdizi[i]<='z')
            kdizi[i]=kdizi[i]-32;
        }
}
void kharf(char *bdizi, int ii){
    for(int i=0;i<ii;i++){
        if(bdizi[i]>='A' && bdizi[i]<='Z')
        bdizi[i]=bdizi[i]+32;
        }
}
```

ARAŞTIRINIZ!

32'nin çıkarılmasının sebebi nedir?

ARAŞTIRINIZ!

32'nin eklenmesinin sebebi nedir?

Program 10.9'un küçük bir problemi var. Bu problem, program Türkçe karakterleri küçük harften büyük harfe veya büyük harften küçük harfe dönüştürmüyor. Bunun için bir araştırma yaparak programa Türkçe karakterleri de dönüştürecek bir özellik katmayı deneyiniz.

10.7. Karakter Katarlarının Fonksiyon Argümanı Olarak Kullanılması

Program ***10.10'***un listesinde yer alan ***yazdır()*** fonksiyonu karakter katarı içerisindeki her karaktere sırayla erişerek karakter katarını ekrana yazdırır. Programda ***kk1*** dizi adresi, yazdır fonksiyonu çağrılırken argüman olarak kullanılmıştır. Bu adres bir sabittir, fakat fonksiyona değer olarak aktarılır ve fonksiyon içerisinde bir kopyası çıkarılır. Bu programda kopya işaretçi olan ***kk***'dır. Bir işaretçi sabitten farklı olarak değiştirilebilir. İşaretçinin bu özelliğinden faydalanılarak fonksiyonda ***kk*** arttırılır ve ****kk++*** ifadesi de karakter katarının arka arkaya gelen karakterlerini döndürür. Döngü karakter katarının ***null*** değerine kadar devam eder ve karakter katarının tamamı ekrana yazdırılır.

```
1   #include <iostream>
2   using namespace std;
3   int main() {
4       void yazdir(char*);
5       char kk1[]="Karakter Katari Sabiti";
6       yazdir(kk1);
7       return 0;
8   }
9   void yazdir(char* kk)
10  {
11      while(*kk)
12      {
13          cout<<*kk++;
14          }
15      cout<<endl;
16  }
```

Program 10.11Karakter Katarları ve fonk. Arg.

```
Karakter Katari Sabiti

--------------------------------
Process exited after 0.06507 seconds with return value 0
Press any key to continue .
```

Program 10.11'in Ekran Çoktısı

Bölüm Özeti

Diğer dillerde işaretçi olmamasına karşılık C++ ve C'nin en önemli özelliği işaretçileri destekliyor olmalarıdır. Bu dillerde işaretçi olmadan da birçok işlem yapılabileceğini bundan önceki bölümlerde gördük. Fakat veri yapılarını tanımlamak, sistemden bellek almak, bazı operatörleri kullanabilmek için işaretçileri kullanmamız gerekmektedir. Bu bölümün işaretçilere ayrılmasının nedeni programlama ile uğraşan herkesin işaretçiler konusunda bir fikri olması gerekliliğindendir.

İşaretçilerin bir değişken olduğunu ve içerisinde başka bir değişkenin adresini tuttuğunu, adres operatörünün(&) değişkenlerin bellek adreslerini öğrenmek için kullanıldığını, işaretçi tanımlarken değişkenin başına asteriks(*) işareti konulması gerektiğini, bu şekilde tanımlanan değişkenin adres bilgisi tutması gerektiğini, işaretçiye değer atarken kullanılan asteriks(*) işareti ile gösterici tanımlarken kullanılan asteriks(*) işaretinin farklı anlamlara geldiğini, gösterici kullanılırken, göstericiye kendi tipinde bir değer atanması gerektiğini, aksi halde veri kaybı yaşanabileceğini, işaretçilerle aritmetiksel işlemler yapılabildiğini öğrendik.

Bunlardan başka işaretçilerin fonksiyonlara aktarılabileceğini ve bunun işaretçilerin en faydalı kullanım şekillerinden biri olduğunu, Bir dizinin isminin, o dizinin ilk elemanının bellek adresi olduğunu, dizi ismi de bir işaretçi olduğu için işaretçilerle yapılan matematiksel işlemlerin dizi ismi ile de yapılabileceğini gördük.

Bunlara ek olarak C++ Programlarında dizi adresinin(dizinin ismi), fonksiyon çağırılırken argüman olarak Kullanılabileceğini bir örnekle gösterdik.

Dr. Rifat Çölkesen, Programlama Sanatı Algoritmalar, Papatya,2004

Dr. Rifat Çölkesen, Veri Yapıları ve Algoritmalar, Papatya,2007

H. Burak Tungut, Algoritma ve Programlama Mantığı, KODLAB Yayın Dağıtım Yazılım ve Eğitim Hizmetleri San. ve Tic. Ltd. Şti, 2019.

Duygu Arbatlı Yağcı, Nesne Yönelimli C++ Programlama Kılavuzu, Alfa Basım Yayım Dağıtım Ltd. Şti, 2016.

G. Murat Taşbaşı, C programlama, Altaş yayıncılık ve Elektronik Tic. Ltd. Şti. 2005.

Muhammed Mastar, Süha Eriş, C++, KODLAB Yayın Dağıtım Yazılım ve Eğitim Hizmetleri San. ve Tic. Ltd. Şti, 2012.

Sabahat Karaman, Pratik C++ Programlama, Pusula Yayınevi,2004