BÖLÜM 10

İŞARETÇILER

ANAHTAR KAVRAMLAR

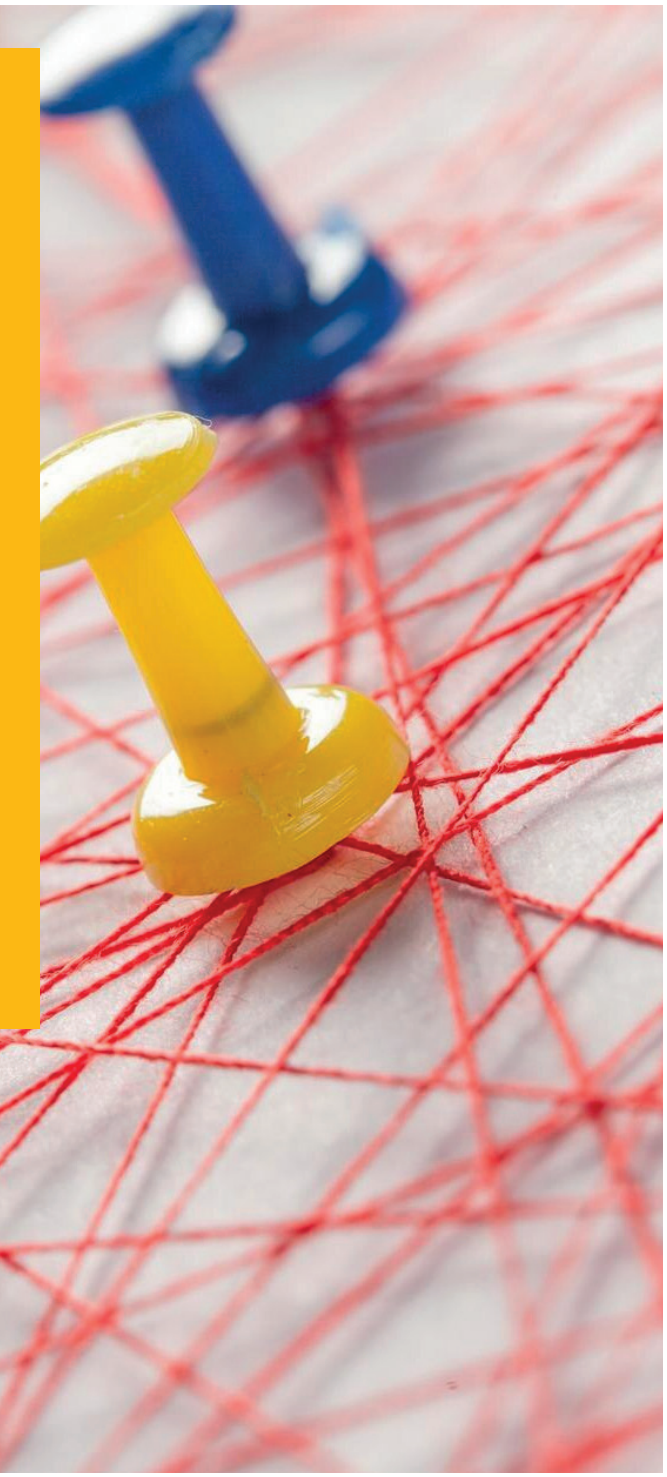
İşaretçiler, Adres İşleci, Referanstan Ayırma İşlemi, İşaretçi İfadesi, İşaretçi Aritmetiği

ÖĞRENME HEDEFLERİ

1. İşaretçi tanımlamayı ve kullanmayı bilir.
2. İşaretçi tanımlamayı ve kullanmayı bilir.
3. Fonksiyonlara argümanları işaretçi kullanarak referans ile gönderebilir.
4. İşaretçiler ile beraber const kullanmayı gerçekleştirebilir.
5. İşaretçi ifadeleri ve aritmetiği konularında bilgi sahibi olur.
6. Diziler ve işaretçiler arasındaki yakın ilişkiyi anlar ve kullanır.

ONUNCU BÖLÜM

GİRİŞ



Bu bölümde C++ programlama dilinde yer alan işaretçiler anlatılacaktır. İşaretçiler, diğer tüm değişkenler gibi belirli bir bellek alanı kullanırlar. Diğer değişkenlerden farklı olarak gerçek veri değeri yerine adres değeri tutarlar. Bu adres bir veri tipinde tanımlanmış bir değişkenin bellekte yer aldığı alanın adresidir. Bu nedenle işaretçiler de uygun tiplerde tanımlanırlar. Tam sayı olarak tanımlanmış bir işaretçi, tam sayı olarak tanımlanmış bir değişkenin bellekteki adresini gösterir. Farklı veri tipleri bellekte farklı miktarda yer kullandıkları için işaretçinin hangi tipte tanımlandığı önemlidir. İşaretçiler kullanılarak parametreler fonksiyonlara referans ile gönderilebilirler. Bu durumu gerçekleştirmek için fonksiyon çağırma işleminde adres işleci kullanılarak (&) bir değişkenin adresi alınır ve fonksiyona gönderilir. Fonksiyonun parametre listesinde ise gönderilen bu adres değerinin atanacağı bir işaretçi tanımlanır. İşaretçilerin tanımlanmasında const niteleyicisi kullanılarak işaretçinin veya gösterdiği değerin sabit olarak tanımlanması sağlanabilir. İşaretçiler aritmetik ifadelerde, atama ifadelerinde ve karşılaştırma işleçlerinde işlenen olarak yer alabilirler. Artırma ve azaltma işleçleri işaretçiler üzerinde kullanılabilir. Dizinin ilk elemanının adresi bir işaretçi değişkenine atanabilir. Bu şekilde dizinin ismi ve adresin atandığı işaretçi kullanılarak iki farklı gösterim yöntemiyle dizinin farklı elemanlarına ulaşmak mümkün olabilir. Bu yöntemler işaretçi sapma ve işaretçi alt simge gösterimleridir.

1. İŞARETÇİ DEĞİŞKENİ TANIMLAMA VE İLK DEĞER ATAMA

İşaretçiler C++ programlama dilinde adres değerine sahip değişkenler olarak tanımlanırlar. Bir işaretçi değişkeninin aldığı değer bellekte yer alan herhangi bir baytlık alanın adres değeridir. Bir işaretçi C++ da yer alan diğer değişkenler gibi tanımlanır ancak tanımlamada '*' işleci kullanılır. İşaretçiler herhangi bir veri tipinde tanımlanabilirler. int olarak tanımlanan bir işaretçi tam sayı bir değişkenin yer aldığı bellek alanının ilk baytının adresini gösterir. Daha önceki bölümlerde ifade edildiği gibi farklı tipteki değişkenler bellekte farklı miktarlarda alan kullanırlar. Örneğin, char 1 bayt, int 4 bayt (veya 8 bayt) ve double 8 baytlık alanlar kullanır. Bir işaretçi değişkeni (bu örnekte ptr) aşağıdaki şekilde tanımlanabilir:

```
int a=5;
int* ptr;
ptr=&a;
```

ptr bir tam sayı değişkenin adresini tutmak amacıyla işaretçi değişkeni olarak tanımlanmıştır. Bu örnekte a tam sayı değişkeninin adresi (&) işleci kullanılarak elde edilmiş ve ptr işaretçi değişkenine atanmıştır. Bu noktadan itibaren ilgili tam sayı değişkenine erişmek için değişken ismi a kullanılabileceği gibi, aynı değişkenin bellek adresini tutan ptr işaretçi değişkeni de kullanılabilecektir. Yalnız değişken ismi diğer bölümlerde de anlatıldığı üzere doğrudan kullanılırken, ilgili değişkenin değerine ulaşmak için işaretçi değişkeni referanstan ayırma işleci (*) ile birlikte kullanılacaktır.

```
//Program10.1.cpp
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
    int a=5;
    int* ptr;
    ptr=&a;
    cout<<"Dogrudan degisken ismi (a) kullanimi:"<<a<<endl;
cout<<"Dolayli olarak isaretci (*ptr) kullanimi:"<<*ptr<<endl;
      return 0;
}
Program10.1 Çıktısı
Dogrudan degisken ismi (a) kullanimi:5
Dolayli olarak isaretci (*ptr) kullanimi:5
```

Program10.1.cpp'de tanımlanan a tam sayı değişkeninin değeri ekrana önce değişken ismi kullanılarak sonraki satırda ise işaretçi kullanılarak yazdırılmıştır. İşaretçi kullanarak a değişkenine erişebilmek amacıyla öncelikle bu tam sayı değişkenin adresinin, işaretçi değişkene atanması gerekir. Bu atama işlemi main fonksiyonda 3. satırda gerçekleştirilmiştir.

DİKKAT EDELİM

Bir işaretçi değişkeni kullanabilmek için, bu işaretçi değişkene herhangi bir adres atamasının gerçekleştirilmiş olması gerekir.

Program10.2.cpp'de ptr işaretçi değişkeninin değeri ekrana yazdırılmıştır. Programda a tam sayı değişkeninin adresini tutan işaretçi ile a değişkeninin adresinin aynı olması beklenmektedir. Doğrudan a değişkeninin adresi (&) işleci ile elde edilebilir.

```
//Program10.2.cpp
#include <iostream>
using namespace std;
int main()
{
    int a=5;
    int* ptr;
    ptr=&a;
    cout<<"a degiskeni bellek adresi:"<<&a<<endl;
    cout<<"ptr degiskeni degeri:"<<ptr<<endl;
    return 0;
}
```

Program10.2 Çıktısı

```
a degiskeni bellek adresi:0xffffcbf4
ptr degiskeni degeri:0xffffcbf4
```

DİKKAT EDELİM

İşaretçi değişkenleri, program farklı ortamlarda veya farklı zamanlarda çalıştığında farklı değerler alabilir. Programın çalıştığı ortama göre programda kullanılan değişkenler farklı bellek alanlarını kullanabilirler.

Adres işleci (&) ve referanstan ayırma işleci (*) birbirlerinin tersi işleçlerdir.

```
int a=5;
int* ptr;
ptr=&a;
```

Yukarıdaki kod parçası verilmiş olsun. *(&ptr) ile &(*ptr) ifadeleri aynı değeri üretir. Bu değer a değişkeninin adresidir. Program10.3.cpp bu durumu gösteren bir program içermektedir.

```
//Program10.3.cpp
#include <iostream>
using namespace std;
int main()
{
    int a=5;
    int* ptr;
    ptr=&a;
    cout<<"*&ptr: "<<*&ptr<<endl;
    cout<<"&*ptr: "<<&*ptr<<endl;
    return 0;
}
```

Program10.3.cpp Çıktısı

```
*&ptr: 0xffffcbfc
&*ptr: 0xffffcbfc
```

2. İŞARETÇİ KULLANARAK REFERANS İLE FONKSİYON ÇAĞIRMA

Bölüm 8'de fonksiyonların iki farklı şekilde çağrılabileceği anlatılmıştı. Bu yaklaşımlardan birisi parametreleri referans ile gönderme işlemiydi. Bu şekilde fonksiyon içerisinde, doğrudan çağıran fonksiyondaki orijinal değişkene erişim söz konusu olabiliyordu. Ayrıca boyutu büyük nesneler, tekrar oluşturulmaya gerek olmadan fonksiyonlara aktarılabiliyordu. Bu yaklaşımı gerçekleştirmek için referans parametreleri kullanılmıştı. Bu kısımda işaretçileri kullanarak referans ile fonksiyon çağırma işleminin nasıl gerçekleştirileceği anlatılacaktır.

```
//Program10.4.cpp
#include <iostream>
using namespace std;
void toplaIsaretci(int *a, int *b, int *t)
{
    *t=*a+*b;
}
int main()
{
    int i=5, j=15;
    int toplam;
    toplaIsaretci(&i,&j,&toplam);
    cout<<"Iki Sayinin Toplami:"<<toplam<<endl;
    return 0;
}
```

Program10.4.cpp Çıktısı

```
Iki Sayinin Toplami:20
```

Program10.4.cpp'de main fonksiyonda 2 tam sayı değişken (i ve j) tanımlanmış ve bu değişkenlere değer atanmıştır. Bu değişkenlerin toplamı için farklı bir tam sayı değişken (toplam) oluşturulmuştur

toplaIsaretci fonksiyonuna tanımlanan 3 tam sayı değişkenin adresleri gönderilmiştir. Fonksiyonun başlığında da görüleceği gibi parametre listesinde 3 işaretçi tanımlanmıştır. Gönderilen adresler sırası ile bu listede yer alan işaretçilere atanmıştır. Fonksiyonun içinde yer alan matematiksel ifadede referanstan ayırma işleci (*) ile ilgili adreslerin gösterdiği bellek alanındaki değerlere ulaşılmış ve toplama işlemi gerçekleştirilmiştir. Dolayısı ile fonksiyonun herhangi bir değer çevirmesine gerek yoktur ve bu nedenle dönüş tipi void olarak tanımlanmıştır.

3. İŞARETÇİLERDE CONST KULLANIMI

Eğer bir fonksiyon tanımında bir parametrenin değeri değişmiyorsa veya değişmemesi gerekiyorsa bu parametre const olarak tanımlanabilir. const niteleyicisinin fonksiyonların parametre listesinde yer alan işaretçiler ile 4 farklı kullanımı vardır. Ayrıca bu farklı durumlar bir fonksiyon içindeki işaretçi tanımlamalarında da geçerlidir.

- Sabit olmayan işaretçi ile sabit olmayan veri (const niteleyicisi kullanılmıyor.)
- Sabit olmayan işaretçi ile sabit veri
- Sabit işaretçi ile sabit olmayan veri
- Sabit işaretçi ile sabit veri

Sabit olmayan işaretçi, sabit olmayan veriyi gösterdiğinde const niteleyicisi kullanılmaz (örneğin, int *ptr;). Bu durum tanımlanabilecek en üst seviye erişim hakkıdır. Adresten ayırma işleci ve işaretçi ile gösterdiğimiz veriyi değiştirebiliriz. Aynı zamanda işaretçimizin bir başka bellek alanının göstermesini de sağlayabiliriz. Bu bölümde şu ana kadar kullanılan işaretçiler bu kısımda değerlendirilebilir.

int x=10, y =20;
*int * ptr=&x;*

Yukarıdaki tanımlamaların gerçekleştiğini varsayalım. Bu noktadan sonra programımızda, *ptr=30; ifadesi ile x tam sayı değişkeninin değerini 30 olarak değiştirebiliriz. Ayrıca ptr=&y; ifadesi ile ptr işaretçisinin ayrı bir değişkeni göstermesini de sağlayabiliriz. Bu haklara sahip olmamızın nedeni const niteleyicisinin kullanılmamış olmasıdır.

Sabit olmayan işaretçi ile sabit veriyi gösterdiğimizde const niteleyicisi veri tipinden önce yazılır (örneğin, const int *ptr;). Bu ifade "ptr işaretçisi sabit bir tam sayı değerini" gösterir şeklinde okunabilir. Adresten ayırma işleci ve işaretçi ile gösterdiğimiz veriyi değiştirebilmemiz söz konusu değildir. Ancak işaretçimizin bir başka bellek alanını göstermesini sağlayabiliriz.

```
int x=10, y =20;

const int * ptr=&x;
```

Yukarıdaki tanımlamaların gerçekleştiğini varsayalım. Bu noktadan sonra programımızda, *ptr=30; ifadesi ile x tam sayı değişkeninin değerini 30 olarak değiştirmeye çalıştığımızda derleyici hata üretir. Ancak ptr=&y; ifadesi ile ptr işaretçisinin ayrı bir değişkeni göstermesini sağlayabiliriz.

Sabit işaretçi ile sabit olmayan veriyi gösterdiğimizde const niteleyicisi işaretçiden önce yazılır (örneğin, int * const ptr;). Bu ifade "ptr sabit işaretçisi bir tam sayı değerini" gösterir şeklinde okunabilir. Adresten ayırma işleci ve işaretçi ile gösterdiğimiz veriyi değiştirebiliriz. Ancak işaretçimizin bir başka bellek alanını (bir başka değişken) göstermesini sağlamak mümkün olmaz.

int x=10, y =20;
*int * const ptr=&x;*

Yukarıdaki tanımlamaların gerçekleştiğini varsayalım. Bu noktadan sonra programımızda, *ptr=30; ifadesi ile x tam sayı değişkeninin değerini 30 olarak değiştirebiliriz. Ancak ptr=&y; ifadesi ile ptr işaretçisinin ayrı bir değişkeni göstermesini istediğimizde derleyici hata üretir. ptr const olarak tanımlanmıştır.

Sabit işaretçi ile sabit veriyi gösterdiğimizde const niteleyicisi işaretçiden önce ve veri tipinden önce olmak üzere 2 kez yazılır (örneğin, const int * const ptr;). Bu ifade "ptr sabit işaretçisi sabit bir tam sayı değerini" gösterir şeklinde okunabilir. Adresten ayırma işleci ve işaretçi ile gösterdiğimiz veriyi değiştirmemiz mümkün değildir. Aynı zamanda işaretçimizin bir başka bellek alanını (bir başka değişken) göstermesini de sağlayamayız.

int x=10, y =20;
*const int * const ptr=&x;*

Yukarıdaki tanımlamaların gerçekleştiğini varsayalım. Bu noktadan sonra programımızda, *ptr=30; ifadesi ile x tam sayı değişkeninin değerini 30 olarak değiştirmek istediğimizde derleyici hata üretir. Aynı zamanda ptr=&y; ifadesi ile ptr işaretçisinin ayrı bir değişkeni göstermesini istediğimizde derleyici hata üretir.

4. İŞARETÇİ İFADELERİ VE İŞARETÇİ ARİTMETİĞİ

İşaretçiler aritmetik ifadelerde, atama ifadelerinde ve karşılaştırma işleçlerinde işlenen olarak kullanılabilirler. Aşağıda verilen az sayıda aritmetik işlem işaretçiler üzerinde kullanılmak üzere tanımlanmışlardır.

- Artırma işleci (++)
- Azaltma işleci (--)
- Bir tam sayı, işaretçi değişkene eklenebilir (+ veya +=)
- Bir tam sayı, işaretçi değişkenden çıkartılabilir (- veya -=)
- Bir işaretçi diğer bir işaretçiden (aynı tipte tanımlanmaları şartı ile) çıkartılabilir (ancak bu çıkartma işlemi, işaretçiler bir dizinin elemanlarını gösterdiği takdirde anlamlı olabilir).

DİKKAT EDELİM

Günümüzde bilgisayar sistemleri 4 veya 8 baytlık tam sayılar kullanırlar. Bu nedenle işaretçi aritmetik işlemlerinin sonucu kullanılan sisteme göre değişebilir.

Bir işaretçiye bir dizi içinde herhangi bir dizi elemanının adresi atanmış olsun. Bu durumda eğer işaretçinin değerini 1 artırırsak işaretçi dizideki bir sonraki elemanı gösterir. Burada önemli olan durum işaretçinin gerçek değeri olan adres değerine 1 eklenmemiş, işaretçinin tipine göre işaretçinin değeri bir sonraki dizi elemanını gösterecek şekilde artırılmıştır. Eğer dizi double tipinde tanımlanmış ise ve double tipindeki işaretçi bu dizideki herhangi bir elemanını gösterecek şekilde tanımlanmışsa bu işaretçi değişkenini 1 artırmak adres değerini 8 bayt artırmak demektir. Bu durumun nedeni her bir

double değişkenin bellekte 8 baytlık alan kullanmasıdır.

```
//Program10.5.cpp
#include <iostream>
using namespace std;
int main()
{
    int a[10]={7,3,-8,12,5,-1,0,16,10,8};
    int *ptr;
    ptr=&a[2];
    cout<<*ptr<<endl;
    ptr++;
    cout<<*ptr<<endl;
    ptr=ptr+3;
    cout<<*ptr<<endl;
    ptr--;
    cout<<*ptr<<endl;
    return 0;
}
```

Program10.5.cpp Çıktısı

```
-8
12
0
-1
```

Program10.5.cpp'de 10 elemanlı bir tam sayı dizi tanımlanmış ve dizinin elemanlarına başlangıç değerleri atanmıştır. Daha sonra bir tam sayı işaretçi tanımlanmış ve dizinin 3. elemanının adresi bu işaretçiye atanmıştır. Programın devam eden kısmında bazı aritmetik işlemler ilgili işaretçi üzerinde uygulanmıştır. Her işlemden sonra işaretçinin gösterdiği tam sayı değer ekrana yazdırılmıştır.

```
//Program10.6.cpp
#include <iostream>
using namespace std;
int main()
{
    double a[10]={7.0,3.2,-8.1,12.5,5.7,-1.3,0.0,16.2,10.8,8.5};
    double *ptr1;
    double *ptr2;
    ptr1=&a[4];
    ptr2=&a[1];
    cout<<ptr1-ptr2<<endl;
    if(ptr1>ptr2)
            cout <<"Ilk isaretci ikinci isaretciden daha buyuktur."<<endl;
    return 0;
}
```

Program10.6.cpp Çıktısı

```
3
```

İlk isaretci ikinci isaretciden daha buyuktur.

Program10.6.cpp'deki programda bir double dizi ve 2 ayrı double işaretçi değişken tanımlanmıştır. 1. double işaretçi 4. dizi elemanını, 2. double işaretçi ise 1. dizi elemanını gösterecek şekilde başlangıç değerleri almıştır. ptr2 değişkeni ptr1 değişkeninden çıkartıldığında 3 değerini elde ederiz. ptr1 değişkeni 5. elemanı, ptr2 değişkeni 2. elemanı gösterdiği için arada 3 elemanlık bir fark vardır. Dolayısı ile çıkarma işlemi bayt cinsinden bellek adresleri üzerinden gerçekleşmemiştir. Ayrıca aynı programda 2 işaretçi karşılaştırılmış ve daha yüksek indise sahip dizi elemanını gösteren işaretçinin büyük olduğu görülmüştür. Bayt adresi cinsinden bellekte yüksek indise sahip elemanın daha yüksek bir değere sahip olması beklenen bir durumdur.

5. İŞARETÇİLER VE DİZİLER ARASINDAKİ İLİŞKİ

İşaretçiler dizi alt simge işleci [ ] ile gerçekleştirilen tüm işlemlerde kullanılabilirler. Aşağıdaki şekilde bir tam sayı dizi ve bir tam sayı işaretçisi tanımlandığını düşünelim.

```
int a[5];
int *ptr;
```

a dizi ismi burada sabit bir işaretçi olarak değerlendirilebilir. Aynı zamanda dizinin ilk elemanının adresini tutar. Bir dizi tanımlandıktan sonra a'nın değeri (dizinin ilk elemanının adresi) program sonlanıncaya kadar değişmez. Ancak a değeri ile beraber işaretçi sapma ve işaretçi alt simge gösterimleri ile dizinin her bir elemanına erişmek mümkündür. ptr bir işaretçi değişkenidir ve const olarak tanımlanmadığı sürece program süresince farklı bellek alanlarında yer alan değişkenleri göstermek için kullanılabilir.

ptr=a; veya ptr=&a[0]; ifadeleri ile dizinin ilk elemanının adresi ptr işaretçi değişkenine atanır.

a dizisinin 3. elemanı dizi ismi ve alt simge işleci ile a[2] şeklinde erişilebilir. İşaretçi kullanarak ise *(ptr + 2) şeklinde işaretçi ifadesi ile aynı değere ulaşmak mümkündür. Bu gösterim işaretçi sapma gösterimi olarak isimlendirilir. Bu örnekte olduğu gibi &a[2] değerine (ptr +2) ifadesi ile erişilebilir. Aynı zamanda dizinin ismi bir işaretçi gibi kullanılarak *(a+2) şeklinde işaretçi sapma gösterimi ile de dizinin 3. elemanına ulaşmak mümkündür. Bunun yanında alt simge gösterimi de işaretçi olarak tanımlanan değişkenler ile birlikte kullanılabilir. Bu örnek için ptr[2] ifadesi ile aynı dizi elemanına erişilebilir. Dolayısı ile hem dizi ismi hem işaretçi ile işaretçi sapma ve işaretçi alt simge gösterimleri kullanılabilir. Bu da aşağıdaki 4 farklı durumu oluşturur:

- Dizi alt simge gösterimi
- İşaretçi sapma gösterimi (dizi ismi işaretçi)
- İşaretçi alt simge gösterimi
- İşaretçi sapma gösterimi (işaretçi değişkeni ile)

Program10.7.cpp'de yer alan programda bu 4 farklı durum gösterilmiştir. Farklı gösterimler ile dizinin 4. elemanına erişim sağlanmış ve elemanın değeri ekrana yazdırılmıştır.

```
//Program10.7.cpp
#include <iostream>
using namespace std;
```

```
int main()
{
    int a[5]={5, 10, 15, 20, 25};
    int *ptr;
    ptr=&a[0];
    cout<<"Dizi alt simge gosterimi: "<<a[3]<<endl;
    cout<<"Isaretci sapma gosterimi (dizi ismi isaretci): "<<*(a+3)<<endl;
    cout<<"Isaretci alt simge gosterimi: "<<ptr[3]<<endl;
    cout<<"Isaretci sapma gosterimi (isaretci degiskeni ile): "<<*(ptr+3)<<endl;
    return 0;
}
```

Program10.7.cpp Çıktısı

```
Dizi alt simge gosterimi: 20
Isaretci sapma gosterimi (dizi ismi isaretci): 20
Isaretci alt simge gosterimi: 20
Isaretci sapma gosterimi (isaretci degiskeni ile): 20
```

DİKKAT EDELİM

*(dizIsmi + i) ifadesinde dizIsmi değerini değiştirmiyoruz. Sadece bu değeri kullanarak yeni bir adres değeri oluşturuyoruz. Dizi isimleri sabittir ve program süresince değiştirilemezler.

BÖLÜM ÖZETİ

Bu bölümde C++ programlama dilinde kullanılan işaretçiler anlatılmıştır. İşaretçi herhangi bir veri tipinde tanımlanabilen ve adres tutan değişkenlerdir. İşaretçinin değeri, bellekte bir adrestir. İşaretçi tanımlandıktan sonra hangi veri tipinden tanımlandıysa o veri tipinden bir değişkenin adresini değer olarak alır. Bir başka ifade ile tanımlandığı veri tipi, sahip olduğu adres değerinin gösterdiği bellek alanında hangi veri tipinde değer olacağını gösterir. İşaretçiler kullanılarak değişkenler fonksiyonlara referans ile gönderilebilir. Fonksiyonlarda yer alan parametre listesinde, işaretçi tanımlamasında const kullanarak işaretçinin veya gösterdiği değerin değiştirilmesini engellemek mümkündür. Bazı aritmetik işlemler işaretçiler üzerinde kullanılabilir. Bu işlemler arasında; 1 artırma (++), 1 azaltma (--) ve atama işlemleri sayılabilir. Ayrıca aynı dizinin elemanlarını gösteren işaretçiler arasında çıkarma işlemi gerçekleşebilir. Diziler ve işaretçiler arasındaki ilişki de bu bölümde anlatılan bir diğer konudur. İşaretçi sapma ve işaretçi alt simge gösterimleri kullanarak hem dizi isimi hem de işaretçi değişkeni ismi ile ilgili dizinin elemanlarına erişmek mümkündür.

GÖZDEN GEÇİRELİM

1. **int a=10;**
   int *aptr=&a;

   Yukarıdaki kod parçası dikkate alındığında aşağıdakilerden hangisi bir adres değeri göstermez?
   a. a
   b. aptr
   c. &a
   d. &(*aptr)
   e. *(&aptr)

2. **int a=10;**
   int *aptr=&a;
   int *bptr;
   bptr=aptr;
   Yukarıdaki kod parçası dikkate alındığında aşağıdakilerden hangisi a değişkeninin değerini ekrana yazdırır?
   a. cout<<&a<<endl;
   b. cout<<*&aptr <<endl;
   c. cout<<*bptr<<endl;
   d. cout<<&aptr<<endl;
   e. cout<<aptr<<endl;

3. **Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?**

```
#include<iostream>
using namespace std;
int main()
{
    int a=7;
    int b=5;
    int *ptr=&a;
    ptr=&b;
    cout<<*ptr+a+2*b<<endl;
    return 0;
}
```

   a. 12
   b. 19
   c. 20
   d. 22
   e. 26

4. **int x[7]={-2, 4, 11, 7, 6, -5, 8};**
   int *ptr=&x[2];
   Aşağıdaki ifadelerden hangisi x[6] değerini
   ekrana yazdırmak için kullanılamaz?
   a. ptr+=4;
      cout<<*ptr;
   b. cout<<*(x+6);
   c. cout<<ptr[4];
   d. cout<<x[6];
   e. x+=6;
      cout<<*x;

5. **Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?**

```
#include <iostream>
using namespace std;
void fonksiyon(int *b)
{
    int i;
    for(i=0; i<=4; i++)
        b[i]=b[i]+5;
}
int main()
{
    int a[5]={4,8,3,5,6};
    fonksiyon(a);
    fonksiyon(a);
    cout<<a[4]<<endl;
    return 0;
}
```

   a. 11
   b. 16
   c. 17
   d. 20
   e. 21

6. Aşağıdakilerden hangisinde C++ programlama dilinde int hesap(int *ptr) prototipine sahip bir fonksiyon doğru bir şekilde çağrılmamıştır?

```
int a[5]={10,20,40,60,10};
int i=5;
int *j=&a[3];
```

a. hesap(a[3])
b. hesap(a)
c. hesap(j)
d. hesap(&a[3])
e. hesap(&i)

7.
```
int a =5;
int b=10;
int * const ptr=&a;
```
Aşağıda verilen ifadelerden hangisi hata üretir?

a. ptr=&b;
b. a=a+1;
c. b=b*2;
d. *ptr=*ptr+5;
e. b=a+b;

8. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
#include<iostream>
using namespace std;
int main()
{
    int a[5]={5, 10, 15, 11, 8};
    int *ptr;
    ptr=&a[1];
    cout<<*(++ptr)<<endl;
    return 0;
}
```

a. 5
b. 9
c. 10
d. 15
e. 16

9. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
#include <iostream>
using namespace std;

int main()
{
    int a[5]={18, 20, 15, 14, 16};
    int *ptr1=&a[4];
    int *ptr2=&a[2];
    cout<<ptr1-ptr2<<endl;
    return 0;
}
```

a. 1
b. 2
c. 3
d. 4
e. 5

10. Aşağıda verilen kod parçasının ekran çıktısı aşağıdakilerden hangisidir?

```
#include <iostream>
using namespace std;
void fonksiyon(int *b)
{
    *b=*b+15;
}
int main()
{
    int a=10;
  int b=20;
    fonksiyon(&a);
    cout<<a+b<<endl;
    return 0;
}
```

a. 10
b. 15
c. 20
d. 25
e. 45

Yanıt Anahtarı: 1-A, 2-C, 3-D, 4-E, 5-B, 6-A, 7-A, 8-D, 9-B, 10-E

Kaynakça

Deitel, P., Deitel, H. (2017). C++ How to Program. 10. Baskı. Pearson. Essex, İngiltere.

Dikici, M. (2017). C++ Programlama Dili: Yapısal Programlama, Nesnelerle Programlama ve Jenerik Programlama. Seçkin Yayıncılık. İstanbul, Türkiye.

Çobanoğlu, B. (2021). C/C++ Programlama Eğitim Kitabı. Kodlab. İstanbul, Türkiye.

Demir, S., Eryiğit, R., Ar, Y., Tuğrul, B., Gök, O., Kul, S., . . . Yağanoğlu, M. (2019). Programlama Temelleri. Erzurum: Atatürk Üniversitesi Açıköğretim Fakültesi.

Kaleli, C., Bilge, A., Ergin, S., & Şora Günal, E. (2016). Bilgisayar ve Programlamaya Giriş. Eskişehir: Anadolu Üniversitesi.