

C++

# Pointers (İşaretçiler)

# Pointers (İşaretçiler)

- Verilerin bilgisayar hafızasında tutulduğu fiziki alan adres olarak tanımlanabilir.
- Adres, hem donanımla hem de yazılımla ile ilişkilidir. Donanımsal açıdan adres bellekte yer gösteren bir sayıdan ibarettir.
- Mikroişlemci bellekte bir bölgeye ancak o bölgenin adres bilgisiyle ancak erişebilir.
- Fiziki olarak bilgilerin yerleşimi işlemciden işlemciye göre değişiklik göstermektedir. Verilerin 0000 adresinden artan sırayla yerleştirilmesine doğrusal adresleme denir.
- Bazı işlemciler ise bu yerleşim hafızanın en üst kısmından (FFFF adresinden) aşağı doğru olmaktadır.

# Pointers (İşaretçiler)

- Nesnelerin adresleri, sistemlerin çoğunda, derleyici ve programı yükleyen işletim sistemi tarafından ortaklaşa olarak belirlenir.
- Nesnelerin adresleri program yüklenmeden önce kesin olarak bilinemez ve programcı tarafından da önceden tespit edilemez.
- Programı yazan yada kullanan, nesnelerin adreslerini ancak programın çalışması esnasında (run time) görebilir.

# Pointers (İşaretçiler)

- C++ 'da oluşturduğumuz her tip hafızada belirli byte boyutunda yer kaplar.

- **32** bit işlemcili bilgisayarlarda;

`char` = 1 byte

`int` = 4 byte

`float` = 4 byte

`double` = 8 byte

- 64 bit sistemlerde ise, değerler iki katı olarak değişir.
- Pointerlar ise tipe bakmaksızın her zaman;
  - **32** bit sistemlerde 4 byte,
  - **64** bit sistemlerde 8 byte yer kaplar.
- Kısaca; bir integer değişkenin hafızada integer değeri, ya da bir double değişkenin ondalıklı sayı tutması gibi pointer da adres değeri tutan bir değişken olarak tanımlayabiliriz.

# Pointers

## Bellekte kapladığı alan

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

```
int main()
{
```

```
    int i;
    double d;
    float f;
    char c;
    int *pi;
    double *pd;
    float *pf;
    char *pc;
```

```
int i      = 4 byte
double d   = 8 byte
float f     = 4 byte
char c     = 1 byte
```

```
int pi     = 4 byte
double pd  = 4 byte
float pf   = 4 byte
char pc    = 4 byte
```

```
cout << "int i      = " << sizeof(i) << " byte" << endl;
cout << "double d   = " << sizeof(d) << " byte" << endl;
cout << "float f     = " << sizeof(f) << " byte" << endl;
cout << "char c     = " << sizeof(c) << " byte" << endl;
cout << endl;
cout << "int pi     = " << sizeof(pi) << " byte" << endl;
cout << "double pd  = " << sizeof(pd) << " byte" << endl;
cout << "float pf   = " << sizeof(pf) << " byte" << endl;
cout << "char pc    = " << sizeof(pc) << " byte" << endl;
```

```
}
```

# Pointers (İşaretçiler)

- Her değişkenin tipi, adı, değeri ve bellekte bulunduğu bir **adres**i olduğunu biliyoruz. Pointer ise bu yerin yani bellek alanındaki yerin adresinin saklandığı değişken türüdür.

## Örnek Verirsek;

okul\_no = 453 değişkeni için;

**Tipi** = int

**Adı** = okul\_no

**Değeri** = 453

**Adresi** = 1005

1-A dersliğine ait 1005 adresindeki sıra yeri sabittir. Derse değişik öğrencilerin gelmesi durumunda değişen sadece öğrenci numaralarıdır.

### 1-A DERSLİĞİ

1001, 123	1011, 789	1021, 823
1002, 752	1012, 111	1022, 901
1003, 696	1013, 222	1023, 903
1004, 678	1014, 333	1024, 907
1005, <b>453</b>	1015, bos	1025, boş
1006, 287	1016, 899	1026, boş
1007, 900	1017, 890	1027, 278
1008, 876	1018, bos	1028, boş
1009, boş	1019, boş	1029, boş
1010, boş	1020, boş	1030, boş

# Referans Operatörü

- Pointerlara, veriler değil de, o verilerin bellekte saklı olduğu bellek gözlerinin başlangıç adresleri atanır. Bir pointer, diğer değişkenler gibi, sayısal bir değişkendir. Bu sebeple kullanılmadan önce program içinde bildirilmelidir
- Pointerlar konusunda bilmemiz gereken 2 adet önemli operatör vardır. Bunlar;
  - **Reference** **&** ve (Referans Operatörü)
  - **Dereference** **\*** operatörleridir. (Referanstan Ayırmak)
- Pointerlar tek başlarına çalışamazlar ve değer alamazlar bunun için başka bir değişkeni referans almak zorundadırlar.

# Referans Operatörü

Referans operatörü (&) önüne geldiği değişkenin hafızada saklandığı yeri yani adresini gösterir.

```
int *b;
```

```
int okul_no=453;
```

```
b=&okul_no;           //1005 !!!dikkat b atanırken * yok.
```

- Burada b isimli işaretçi okul\_no isimli değişkeni referans olarak almaktadır ve “b eşittir okul\_no’nun adresi” diyebilmekteyiz.
  - Diğer bir deyişle, b değişkeni integer tipindeki herhangi bir değişkenin adresini tutabilecek olan bir işaretçidir.
- Ayrıca b değişkenine okul\_no nun bellek adresini atayabilmemiz için b yi int \*b şeklinde yani integer pointer olarak tanımlıyoruz.
- **b=&okul\_no** işleminde b’ye okul\_no değişkenin referans değerinin (yani adresi) atanması için \* işareti koyulmamıştır. Bu diğer değişkenlerde (int, char v.b.) yapılan işlemin bir benzeridir.



# Pointer Tipindeki Değişkenlerin Tanımlanması

- Bir pointer aşağıdaki gibi tanımlanır;

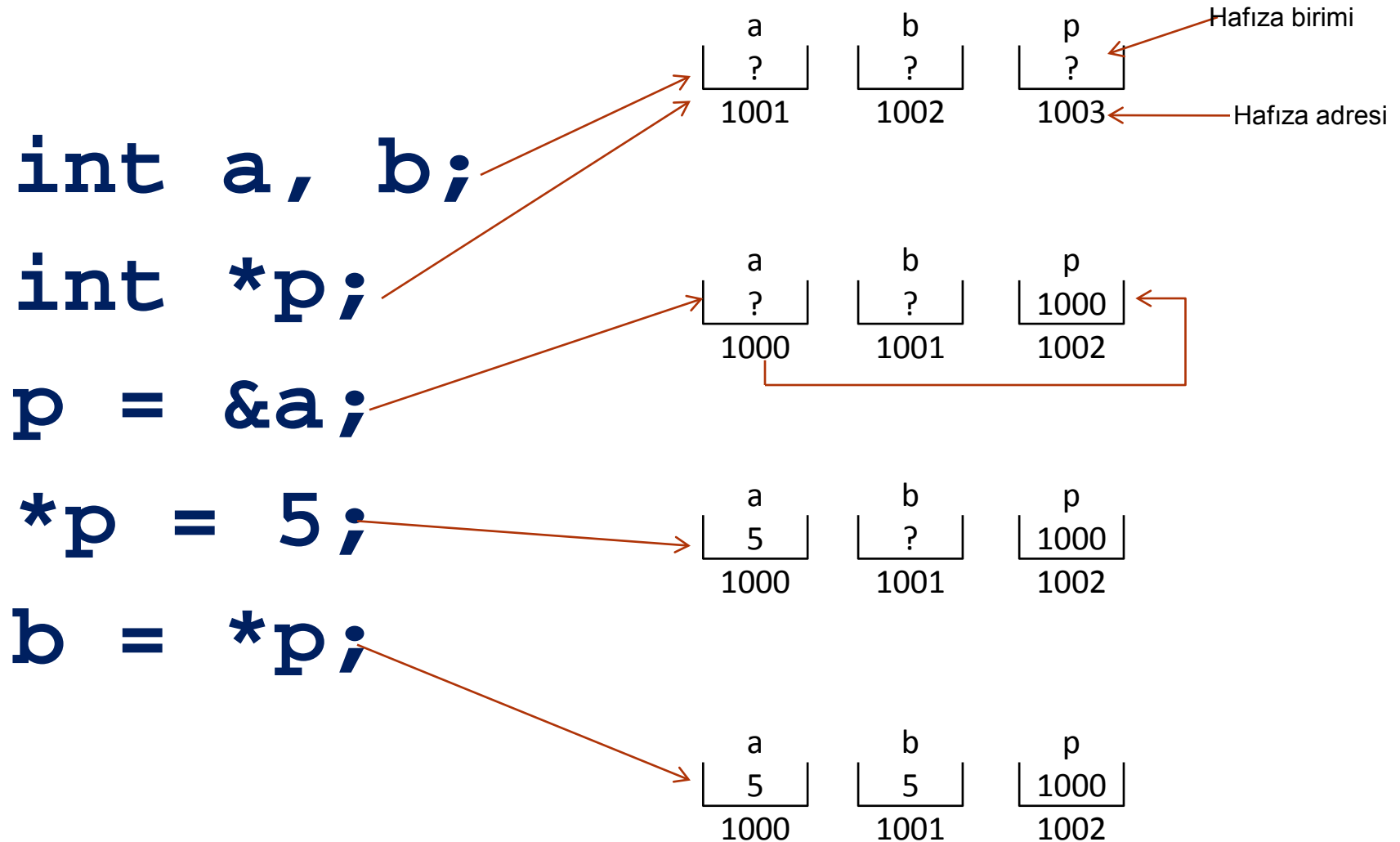
*<Değişken Türü> \*<Değişken Adı>*

*Değişken Türü:* int, double, char, string olabilir.

- \* işaretinin iki kullanım amacı vardır.
  - Eğer tanımlama aşamasında değişkenin önüne getirilirse o değişkenin pointer olduğu belirtilir.
  - Eğer kod içerisinde bir işaretçi değişkenin önüne getirilirse o değişkende kayıtlı adres üzerindeki değeri gösterir.

*Değişken Adı:* c++ da geçerli herhangi bir değişken ismi.

# Pointer Tipindeki Değişkenlerin Tanımlanması



# Pointer Tipindeki Değişkenlerin Tanımlanması

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

int main()
{
    int a = 25;
    int b = a;
    int *c = &a;

    cout<<"a = "<<a<<endl;
    cout<<"b = "<<b<<endl;
    cout<<"c = "<<c<<endl;
    cout<<endl<<endl;
    cout<<"&a = "<<&a<<endl;
    cout<<"&b = "<<&b<<endl;
    cout<<"&c = "<<&c<<endl;

    return 0;
}
```

```
a = 25
b = 25
c = 0x28ff0c
```

```
&a = 0x28ff0c
&b = 0x28ff08
&c = 0x28ff04
```

```
Process returned 0 (0x0)   execution time : 0.075 s
Press any key to continue.
```

Değişken Adı :					
Adrese Kayıtlı Değerler					
Adresler :	0x28ff00	0x28ff04	0x28ff08	0x28ff0c	0x28ff0e

# Pointer Tipindeki Değişkenlerin Tanımlanması

```
int *b;  
int okul_no=453;  
b=&okul_no;  
cout << b    //1005  
cout << *b   //453
```

Bu durumda;

```
cout << *okul_no << endl;    //HATA  
cout << *&okul_no << endl;  //453  
cout << &*okul_no << endl;    //HATA  
cout << &okul_no << endl;    //1005
```

Atanmamış adrese  
çağrı yapılıyor.

Adrese atama  
yapılıp tekrar  
çağrılıyor.

Atanan adres  
çağrılıyor.

# Pointer Tipindeki Değişkenlerin Tanımlanması

```
int okul_no;  
int *b;  
okul_no =453;  
b=&okul_no;  
*b=500;  
cout <<okul_no; //500
```

burada 454 numruralı öğrencinin oturduğu sıranın adresi b değişkenine & referans operatörüyle aktarılıyor. b değişkeni ne 454 nolu öğrencinin oturduğu adres atanıyor. Daha sonra dereference (\*b=500) ile 453 nolu öğrenci bu sıradan kaldırılarak yerine 500 nolu öğrenci oturtuluyor. Artık okul\_no=500 oldu.

# Pointer Tipindeki Değişkenlerin Tanımlanması

- Pointer tipindeki değişkenler, aynı tipte veriyi gösteriyorsa kendi aralarında atama işlemi yapılabilir.

```
int x = 99;  
int *p1, *p2;  
p1 = &x;  
p2 = p1;
```

- Farklı tip veri gösteren pointerlar da atama işlemi yapılabilir ancak tip çevrimi yapılması gerekir.

```
double d = 1213;  
int *i;  
i=(int *)&d; //tip dönüştürülüyor.  
cout << *(double *)i; //tip dönüştürülüyor.
```

# Pointer Tipindeki Değişkenlerin Tanımlanması

- Bir integer pointer'ına integer değerler, bir char pointer'ına char değerleri atayabiliyoduk. Void türünde bir pointer'a atama nasıl yapılır?
- Ayrıca bir float pointer üzerinde yürüyebiliyorduk veya bu pointer'ı başka bir float pointer ile kıyaslayabiliyorduk. Bu işlemleri void pointer üzerinde yapamayız fakat void pointer'ın en önemli faydası dilin esnekliğini artırmasıdır. **Bir void pointer'a değişken tipi ne olursa olsun istediğimiz başka bir pointer'ı atayabiliriz.**
- **Void pointer, üzerinde herhangi bir aritmetik işlem yapamayı ve indeksleyemeyiz, sadece casting (farklı tiplerdeki iki değişkeni birbirine atamak için kullanılan metottur) işlemlerini gerçekleştirmek için kullandığımız pointer türüdür.**

# Pointer Tipindeki Değişkenlerin Tanımlanması

```
#include<iostream>
using namespace std;
main()
{
    int *iP, i=0;
    float *fP, f=0.9999;
    double *dP, d=0.999999999999;
    char *cP, c=99;
    string *sP, s("bu bir string");
    void *vP;
    iP=&i; fP=&f; dP=&d; cP=&c; sP=&s;
    vP=iP; cout << vP << endl;
    vP=&iP; cout << vP << endl;
    vP=fP; cout << vP << endl;
    vP=&fP; cout << vP << endl;
    vP=dP; cout << vP << endl;
    vP=&dP; cout << vP << endl;
    vP=cP; cout << vP << endl;
    vP=&cP; cout << vP << endl;
    vP=sP; cout << vP << endl;
    vP=&sP; cout << vP << endl;}
```

```
0x28ff00
0x28ff04
0x28fef8
0x28fefc
0x28fee8
0x28fef4
0x28fee3
0x28fee4
0x28fed8
0x28fedc
```



# ÖRNEK

## Tip Dönüştürme

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

main()
{
    double d = 1213;
    int *i;

    i=(int *) &d; //tip dönüştürülüyor.

    cout << " d = " << d << endl;
    cout << " &d = " << &d << endl;
    cout << " i = " << i << endl;
    cout << " *i = " << *i << endl;
    cout << " *(double * )i = " << *(double *)i << endl;
    //tip dönüştürüldü aksi halde sonuç son satır için 0 dır.

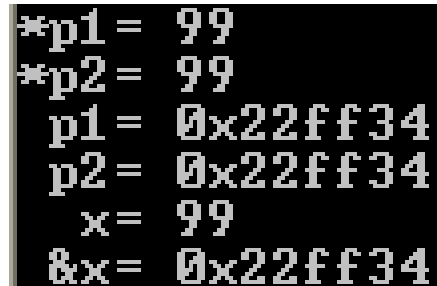
}
```

```
d = 1213
&d = 0x22ff30
i = 0x22ff30
*i = 0
*(double * )i = 1213
```

# ÖRNEK

```
#include <iostream>
using namespace std;
int main()
{
    int x = 99;
    int *p1, *p2;
    p1 = &x;
    p2 = p1;

    cout << "*p1= " << *p1 << endl;
    cout << "*p2= " << *p2 << endl;
    cout << " p1= " << p1 << endl;
    cout << " p2= " << p2 << endl;
    cout << "  x= " << x << endl;
    cout << " &x= " << &x << endl;
}
```



```
*p1= 99
*p2= 99
p1= 0x22ff34
p2= 0x22ff34
 x= 99
&x= 0x22ff34
```

# Pointer Aritmetiği

- Pointer üzerinde yapılabilecek iki tip aritmetik işlem vardır:
  - 1. Toplama
  - 2. Çıkarma.
- Ancak eklenecek yada çıkartılacak değer tamsayı olmalıdır
- Pointer değişkenin değeri 1 arttırıldığı zaman değişken bir sonraki veri bloğunu işaret eder. Değişkenin alacağı yeni değer pointer değişkenin ne tip bir veri bloğunu işaret ettiğine bağlıdır.
- Bir pointer her arttırıldığında, hafızada aynı tipteki bir sonraki değeri, azaltıldığında ise hafızada aynı tipteki bir önceki değeri gösterir.

# Pointer Aritmetiği

```
#include<iostream>
```

```
using namespace std;
```

```
main()
```

```
{
```

```
int *iP1, i;
```

```
iP1 = &i;
```

```
double *iP2, j;
```

```
iP2 = &j;
```

```
cout << "i=" << sizeof(i) << " byte" << endl;
```

```
cout << "iP1=" << sizeof(iP1) << " byte" << endl;
```

```
cout << "iP1=" << iP1 << endl;
```

```
iP1++;
```

```
cout << "iP1=" << iP1 << endl;
```

```
cout << "\n\n";
```

```
cout << "j=" << sizeof(j) << " byte" << endl;
```

```
cout << "iP2=" << sizeof(iP2) << " byte" << endl;
```

```
cout << "iP2=" << iP2 << endl;
```

```
iP2++;
```

```
cout << "iP2=" << iP2 << endl;
```

```
}
```

sizeof()'da \*iP1 veya \*iP2 şeklinde ederek yollanırsa int yada double değişkeninin kapladığı alanı verir. Aşağıdaki gibi iP1 ve iP2 şeklinde yollanırsa adresin kapladığı alanı verir.

```
sizeof(*iP1)=4 sizeof(*iP2)=8
```

```
sizeof(iP1) =4 sizeof(iP2) =4
```

```
i=4 byte
iP1=4 byte
iP1=0x28ff14
iP1=0x28ff18
```

# Pointer Aritmetiği

- Pointerlar ile tamsayı değerleri kullanılarak da işlem yapılabilir.

```
a1 = a1 + 8;
```

a1, ilk değerden sonraki aynı tipteki 8. değeri gösterir.

- Pointerlar da diğer değişkenler gibi değişkenler gibi karşılaştırılabilir.

```
if (a1 < b1)
```

```
cout << "a1 hafızada daha az yer kaplar.");
```

- Pointerlar arası karşılaştırmalar genelde aynı yapı içerisinde hafıza adreslerini gösteriyorsa kullanılır. Örneğin pointer bir dizi gösteriyorsa.

# Pointer ve Diziler

- Elemanları pointer'lardan oluşan pointer dizileri denir.

<Değişken Tipi> \*<Değişken ismi>[Dizi Boyutu]

`int *p[10]`

- p değişkeni 4 byte x 10 =40 byte hafızada yer kaplar.

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

```
int main()
```

```
{
```

```
    int i;
```

```
    int *p[10];
```

```
int i      = 4 byte
int *p[10] = 40 byte
```

```
    cout << "int i      = " << sizeof(i) << " byte" << endl;
```

```
    cout << "int *p[10] = " << sizeof(p) << " byte" << endl;
```

```
}
```

# Pointer ve Diziler

- C/C++ dillerinde pointerlar ve diziler birbirinin yerine kullanılabilir

- Pointerlar dizi elemanlarının işlemlerinde kullanılabilir.

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

```
int *ptr_dizi=NULL;
```

```
ptr_dizi=dizi; //veya ptr_dizi=&dizi[0];
```

- 4 numaralı elemana 2 değerini aktarmak istersek:

```
dizi[4]=2; //veya *(ptr_dizi+4)=2;
```

- Statik değişkenler ve diziler için programın ilk çalışma anında bellekte yer ayrılır ve bu yer program bitine kadar bırakılmaz.

```
char kelime[100];
```

- Bu şekilde statik dizi kullanmanın sakıncaları vardır.

# Pointer ve Diziler

- Pointerlar bellekte kaplanacak yerin derleme aşamasında değil çalışma sırasında belirlenmesini sağlar. Bu şekilde kullanılan değişkenlere **dinamik değişken** denir.
- **new** ve **delete** operatörleri ile C++ dinamik dizi oluşturulabilir.
- Dinamik dizi oluşturmak için **new** operatörü kullanılır.

```
new <veri_tipi> [eleman_sayisi]
```

```
int *odenen;
```

```
odenen = new int [10]
```

odenen adında 10 elemanlı bir int dizi oluşturur.

- Oluşturulan diziyi silmek ve dizinin kullandığı bellek alanını serbest bırakmak için ise **delete** operatörü kullanılır.

```
delete odenen;
```



# Pointer ve Diziler

```
#include <iostream>

using namespace std;

main()
{
    const char *cC=new char[10];
    cC="Ankara";
    cout << cC[2] << endl;

    int *nN;
    nN = new int [10];
    nN[0]=5;
    nN[1]=5;
    nN[2]=7;
    nN[3]=5;
    cout << nN[2] << endl;
    delete cC, nN;
}
```

0 da verilebilir

# Pointer ve Diziler

```
#include <iostream>
```

```
using namespace std;
```

```
main()
```

```
{
```

```
    int b[5]={11,22,33,44,55};
```

```
    const int *bP;
```

```
    bP=b; // bP=&b[0]; gibi.
```

```
1. Eleman b[0]=*bP    -> 11
2. Eleman &b[1]=*(b+1)-> 22
3. Eleman &b[2]=bP+2   -> 0x28ff00
4. Eleman b[3]=*(bP+3)-> 44
5. Eleman &b[4]=bP[4]  -> 55
```

```
    cout <<"1. Eleman b[0]=*bP    -> " << *bP << endl;
```

```
    cout <<"2. Eleman &b[1]=*(b+1)-> " << *(b+1) << endl;
```

```
    cout <<"3. Eleman &b[2]=bP+2   -> " << bP+2 << endl;
```

```
    cout <<"4. Eleman b[3]=*(bP+3)-> " << *(bP+3) << endl;
```

```
    cout <<"5. Eleman &b[4]=bP[4]  -> " << bP[4] << endl;
```

```
}
```

# Pointer ve Diziler

```
#include <iostream>

using namespace std;

main()
{
    int b[4][5]={ {1,2,3,4,5},
                  {6,7,8,9,10},
                  {11,12,13,14,15},
                  {16,17,18,19,20}};
    cout << *(&b[0][0])+16 << endl;
    //b matrisinin 16. elemanı =17 dir.
}
```

17

# Pointer ve Diziler

Pointer değişken aracılığı ile dizi üzerinde dolaşılması.

```
#include <iostream>
```

```
using namespace std;
```

```
main()
```

```
{
```

```
int array[5] = {11,22,33,44,55};
```

```
int *pInt = array;
```

```
for (int i = 0; i < 5; i++)
```

```
    cout << "array[" << i << "]=" << array[i] << endl;
```

```
    cout << "Pointer degisken araciligiyla dizi uzerinde dolasilmasi." << endl;
```

```
    cout << "&array=" << &array<<endl;
```

```
    cout << "&pInt =" << &pInt<<endl;
```

```
    cout << endl;
```

```
    for (int i = 0; i < 5; i++)
```

```
        cout << "* (pInt+" << i << ")=" << *(pInt + i) << " &pInt[" << i << "]=" << &pInt[i] << endl;
```

```
    cout << endl;
```

```
    cout << "Array " << sizeof(array) << " byte yer kaplar."<< endl;
```

```
    cout << "pInt " << sizeof(pInt) << " byte yer kaplar."<< endl;
```

```
}
```

```
array[0]=11
array[1]=22
array[2]=33
array[3]=44
array[4]=55
Pointer degisken araciligiyla dizi uzerinde dolasilmasi.
&array=0x28fef4
&pInt =0x28fef0

*(pInt+0)=11 &pInt[0]=0x28fef4
*(pInt+1)=22 &pInt[1]=0x28fef8
*(pInt+2)=33 &pInt[2]=0x28fefc
*(pInt+3)=44 &pInt[3]=0x28ff00
*(pInt+4)=55 &pInt[4]=0x28ff04

Array 20 byte yer kaplar.
pInt 4 byte yer kaplar.
```

# Pointerın Pointeri

- Şu ana dek incelediğimiz tüm pointerlar bir değişkene yada bir sabite işaret ediyordu. Yani bir değişken yada sabitin adresini içeriyordu.
- Bir pointerın bir başka pointere işaret etmesi mümkündür. Bu durumdaki pointerların başında \*\* işleci getirilir. Örneğin, \*\*p1 pointerı bu tür bir pointerdır.
- Pointer tanımında \* işlecinin sayısal olarak sınırı yoktur ancak bu tür tanımlar işi karıştıracığından sayısının belirlenmesi konusunda yazılımdaki amaç dikkate alınmalıdır.

..... (\*\*\*)- > **Pointerın Pointeri** (\*\*) -> **Pointer** (\*) -> **Değişken (p1)**

# Pointerin Pointeri

```
#include <iostream>

using namespace std;

main()
{
    float *a, **b, c;

    a=&c;
    b=&a;

    **b=3.1416;

    cout <<"    c= " << c << endl;
    cout <<"    *a= " << *a << endl;
    cout <<"    **b= " << **b << endl;
    cout <<"    c= " << &c << endl;
    cout <<"    *&a= " << *&a << endl;
    cout <<"    *&b= " << *&b << endl;
    cout <<"    **&b= " << **&b << endl;
}
```

```
c= 3.1416
*a= 3.1416
**b= 3.1416
c= 0x28ff04
*&a= 0x28ff04
*&b= 0x28ff0c
**&b= 0x28ff04
```

# Fonksiyonlarda Pointer Kullanımı

- Bir fonksiyon tanımlanırken, parametreleri pointer olabileceği gibi, fonksiyon tipi de pointer olabilir.
- Fonksiyonlara değişik şekillerde parametre geçilebilir. Bunlar; değer geçilerek (Call By Value), referans parametreleri ile ve pointer parametreler ile referans geçerek.
- Parametresi pointer olan fonksiyonlar aşağıdaki gibi tanımlanabilir;

```
int fonk1(int *sayi)
```

# Fonksiyonlarda Pointer Kullanımı

Pointer tipinde referans geçilmesi ve değer geçerek parametre yollama yöntemi ile arasındaki fark aşağıdaki programda gösterilmiştir.

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

void DegerIleCagirma(int parametre)
{ parametre = 8;}

void PointerReferansIleCagirma(int *parametre)
{ *parametre = 8;}

main()
{
    int i=100;
    cout << "i = " << i << endl;
    DegerIleCagirma(i);
    cout << "DegerIleCagirma(i) fonksiyonundan sonra i = " << i << endl;
    PointerReferansIleCagirma(&i);
    cout << "PointerRefreransIleCagirma(i) fonksiyonundan sonra i = " << i << endl;
}
```

```
i = 100
DegerIleCagirma(i) fonksiyonundan sonra i = 100
PointerRefreransIleCagirma(i) fonksiyonundan sonra i = 8
```



# Fonksiyonlarda Pointer Kullanımı

- Tipi pointer olan fonksiyon kendisini çağırana bir adres gönderir. Bu tip uygulamalar özellikle kelime dizilerinde sık kullanılır.
- Fonksiyona İşaret Eden Pointerlar;
  - Bir fonksiyona işaret ederler,
  - Fonksiyon pointer'ı, bir fonksiyona parametre olabilir,
  - Fonksiyon pointer'ı bir fonksiyondan geri döndürülebilir,
  - Baska bir fonksiyon pointer'ına atanabilir.

```
int (*fonk1) (int);
```

// int parametresi alan ve int döndüren bir fonksiyon pointeri.

```
int *fonk2(int,int );
```

// iki adet int parametre alan ve int pointeri döndüren fonksiyon pointeri.

# Fonksiyonlarda Pointer Kullanımı

int parametresi alan ve int döndüren bir fonksiyon pointeri.

```
#include <iostream>
using namespace std;
int kareAl (int x){
    return x*x;
}
int kubAl (int x){
    return x*x*x;
}

main() {
    int (*islem) (int);
    int sayi, secim;
    cout<<"\nSayiyi giriniz:"; cin>>sayi;
    cout<<" karesi icin (1), kup icin (2) ";
    cin>>secim;
    if (secim == 1)
        islem = kareAl;
    else if (secim==2)
        islem = kubAl;
    else{
        cout<<"Yanlis secim\n";
    }
    cout<<"Sonuç ="<<islem(sayi);
}
```

kareAl ve kubAl fonksiyonunun ve başlangıç adresi işlem fonksiyon pointerına atanıyor

Seçilen fonksiyon çalıştırılıyor.