



Programlamaya Giriş

HAFTA 11

İşaretçiler (Pointers)

Prof. Dr. Cemil ÖZ

Doç. Dr. Cüneyt BAYILMIŞ

Arş. Gör. Dr. Güluzar ÇİT

Konu & İçerik

- İşaretçiler (Pointers)
- İşaretçiler ve Diziler
- İşaretçi Aritmetiği
- İşaretçi Gösteren İşaretçiler
- Struct Gösteren İşaretçiler
- Fonksiyon Parametresi Olarak İşaretçiler
- Dinamik Bellek Kullanımı (Nesne Gösteren İşaretçiler)
- Dinamik Diziler
- void İşaretçi
- Kaynaklar



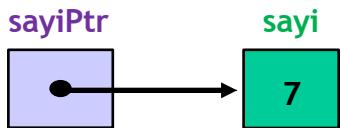
İşaretçiler (Pointers)

- Çok güçlü bir özellik fakat yönetimi zor.
- Belleğin dinamik olarak kullanımına olanak sağlar.
- Özellikle dizi ve karakter dizileriyle yakın ilişkisi vardır.
- Genel kullanım alanları
 - Dizi elemanlarına erişim
 - Fonksiyon giriş parametrelerinin orijinalinde de değişiklik gerektiğinde
 - Dizi ve karakter dizilerini fonksiyonlara gönderirken
 - Sistemden bellek isterken
 - Bağlı listeler, ağaç yapıları v.s. gibi veri yapıları oluştururken

İşaretçiler (Pointers)...

- İşaretçi değişkenler
 - Değer olarak bellek adresi içerir.
 - İşaretçinin içeriği **adres**, bir **değişken**, **fonksiyon**, **yapı** veya **nesne** gösterebilir.
- Normalde değişkenler belirli bir değeri içerir. (direct reference)
sayı

- İşaretçiler belirli değere sahip olan değişkenin bellek adresini içerir (indirect reference)



İşaretçiler (Pointers)...

➤ İşaretçi İşleçleri

İşaretçi İşleci	Anlamı	Görevi
&	Adres İşleci	Değişkenin bellek adresini döndürür.
*	Adres İçeriği	İlgili adressteki veriyi (değeri) döndürür.
& ve * birbirinin tümleyenidir (tersidir).		

```
int a = 5;
char b = 'z';
char *ptr_char = NULL;
ptr_char = &b;
```

Adres	Değişken	Değer
0100	a	5
0101		
0102	b	'z'
0103	ptr_char	0102
0107		
0108		
0109		



İşaretçiler (Pointers)...

➤ İşaretçi değişkenlerin bildirimi

```
tip * işaretçiAdı;
```

```
int *myPtr;  
int *myPtr1, *myPtr2;
```

- Alacağı değerler : **0**, **NULL**, ya da herhangi bir adres
- **0** or **NULL** hiçbir şeyi göstermez.



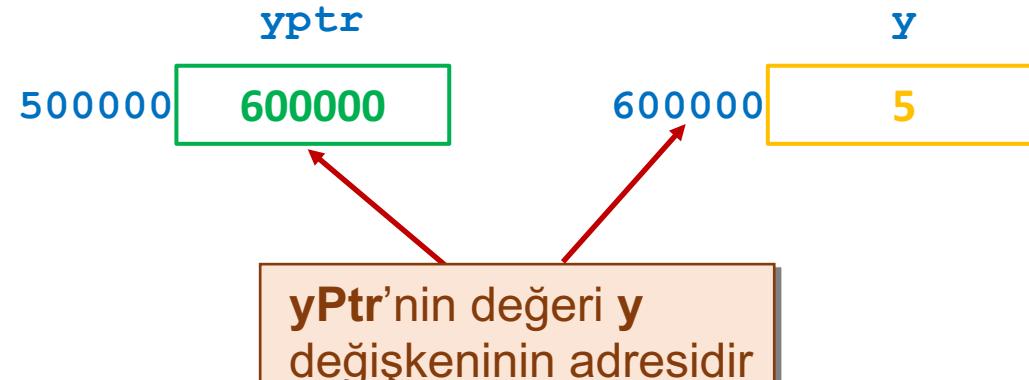
İşaretçiler (Pointers)...

- & (Adres İşleci) ⇒ "address of"
- Bellek adresini geri döndürür

```
int y = 5;  
int *yPtr;  
yPtr = &y;
```

➤ yPtr, y'nin adresini alır, yani
yPtr, y'ye "işaret eder"

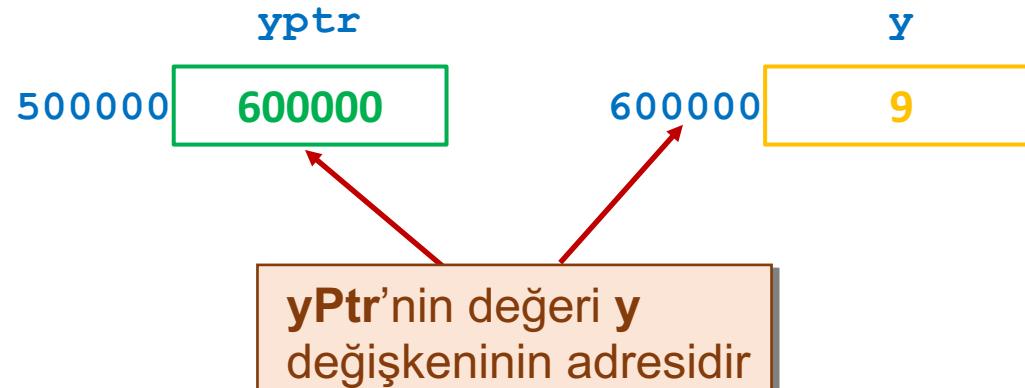
//İlk değer ataması
int y=5;
int *yPtr = &y;



İşaretçiler (Pointers)...

- * (Adres İçeriği) ⇒ "value pointed by"
 - Bellek adresinin içeriğini geri döndürür
- * ve & birbirlerinin tersidir.

```
int y = 5;  
int *yPtr;  
yPtr = &y;  
  
*yPtr = 9;
```



İşaretçiler (Pointers)...

➤ ÖRNEK: ⇒ [1]_isaretci.cpp

```
int x;
int *xptr;

x = 3;
xptr = &x;

cout << "x'in adresi      : " << &x << endl;
cout << "xptr'in icerigi : " << xptr << endl;
cout << "x'in degeri     : " << x << endl;
cout << "(*xptr)'nin degeri : " << *xptr << endl;

cout << "\n\n* ve & operatorleri birbirinin tümleyenidir \n";
cout << "&xptr: " << &xptr << endl;
cout << "*&xptr: " << *&xptr << endl;
```

```
x'in adresi      : 007BFC8C
xptr'in icerigi : 007BFC8C
x'in degeri     : 3
(*xptr)'nin degeri : 3

* ve & operatorleri birbirinin tümleyenidir
&xptr: 007BFC8C
*&xptr: 007BFC8C
Press any key to continue . . .
```

İşaretçi Aritmetiği

- İşaretçilerle kullanılabilecek aritmetik işlemler

- `++, --, +, +=, -, -=`

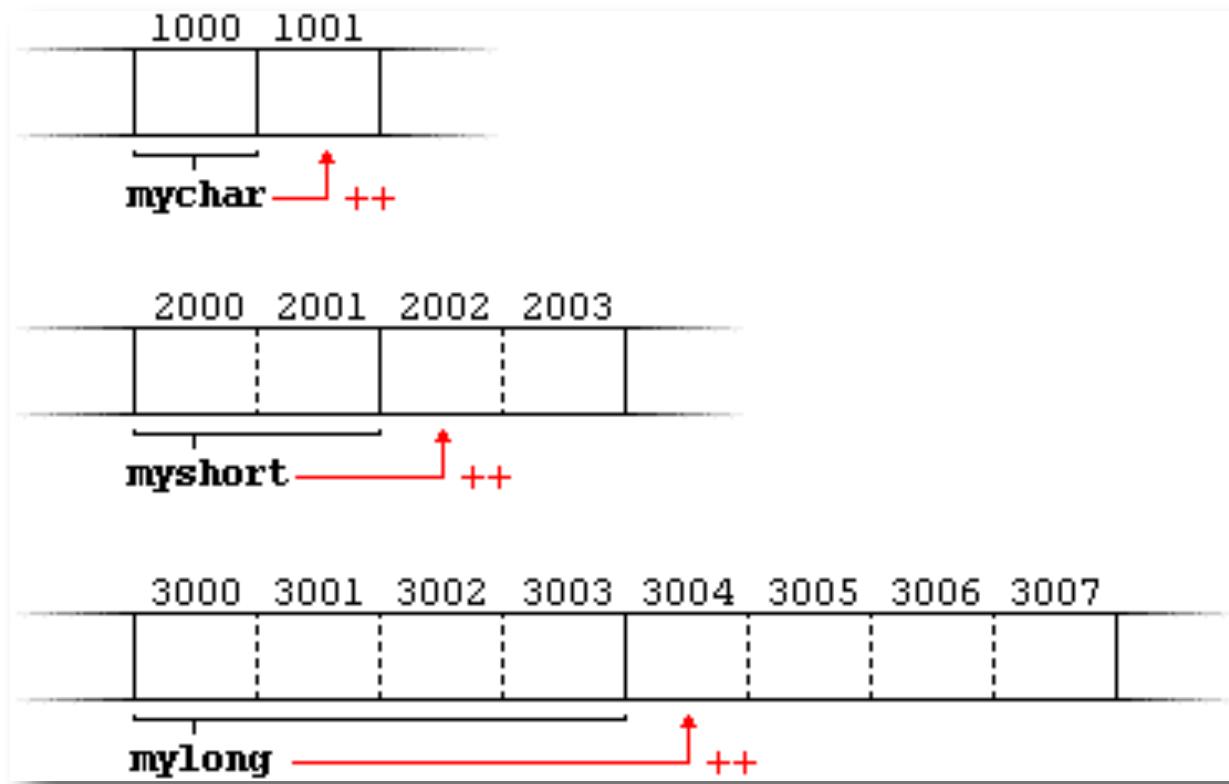
```
int a[5]={0};    // a dizisinin başlangıç adresi 0100 olsun.  
  
int *aPtr=a;    // aPtr, a dizisinin başlangıcını işaret ediyor  
                 // yani, aPtr=0100 där.  
  
aPtr++;         // aPtr=0102 olur.  
                 // a[1] elemanına işaret ediyor  
  
(*aPtr)++;     // a[1]=1 olur.
```

Adres	Değer
0100	0
0101	
0102	0 → 1
0103	
0104	0
0105	
0106	0
0107	
0108	0
0109	
010A	

İşaretçi Aritmetiği...

```
char *mychar;  
short *myshort;  
long *mylong;
```

```
mychar++;  
myshort++;  
mylong++;
```



İşaretçiler (Pointers)...

➤ ÖRNEK: ⇒ [2]_aritmetik.cpp

```
int sayilar[5];
int sayi = 15;

int* sayiPtr;
//cin>>sayi;
sayiPtr = &sayi;
*sayiPtr = 20;      //sayi = 20;

cout << "sayilar:" << sayilar << endl
    << " sayilar[0] adresi:" << &sayilar[0] << endl
    << "sayilar işaretçi sabitinin boyutu:" << sizeof(sayilar)

cout << "sayi:" << sayi << endl
    << "adres:" << sayiPtr << endl
    << "isaretçinin boyutu:" << sizeof(sayiPtr) << endl<< endl;

int a[5] = { 0,1,2,5,7 };

int *p;
p = a;

cout << "*(&a + 3) => " << *(a + 3) << endl;
cout << "*(&p++) => " << *(p++) << endl;
cout << "*(&p) => " << *(p) << endl<< endl;
//cout<<*(++a);      //hatalı, a işaretçi sabitidir, değeri değiştirilemez
```

```
sayilar:0055F794
sayilar[0] adresi:0055F794
sayilar işaretçi sabitinin boyutu:20

sayi:20
adres:0055F788
isaretçinin boyutu:4

*(a + 3) => 5
*(p++) => 0
*(p) => 1

31
54
77
52
93

Press any key to continue . . .
```

İşaretçiler ve Diziler

- Dizinin ilk elemanın adresini içeren dizi adı aslında işaretçi gibi düşünülebilir.
- İşaretçiler, dizi elemanları ile ilgili işlemlerde kullanılabilir.

```
int sayilar[5];
int * p;

p = sayilar;
```

```
int sayı;
int *p1 = &sayı;
```

İşaretçiler ve Diziler...

➤ ÖRNEK: ⇒ [3]_dizi.cpp

```
int sayilar[5];
int * p;

p = sayilar;
*p = 10;

p++;
*p = 20;
p = &sayilar[2];
*p = 30;
p = sayilar + 3;
*p = 40;
p = sayilar;
*(p + 4) = 50;

for (int n = 0; n < 5; n++)
    cout << sayilar[n] << ", ";
```

sayilar++	10	0xAB500
	20	0xAB504
sayilar[2]	30	
	40	
	50	
	AB500	p

Karakter Katarları

```
const char* katar = "SAKARYA";
```

```
char katar[] = "SAKARYA";
char* katarPtr = katar;
```

$*(\text{katar}+4) \Rightarrow 'R'$
 $\text{katar}[4] \Rightarrow 'R'$

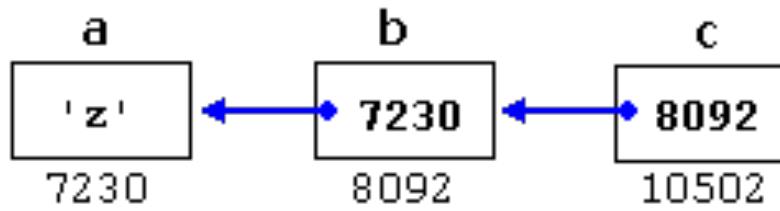


katar \Rightarrow "SAKARYA"
 $\&\text{katar} \Rightarrow 1000$



İşaretçi Gösteren İşaretçiler

```
char a;  
char *b;  
char **c;  
  
a = 'S';  
b = &a;  
c = &b;
```



Struct Gösteren İşaretçiler

➤ ÖRNEK: ⇒ [4]_struct.cpp

```
struct Olcu
{
    int metre;
    int cmetre;
};

int main()
{
    Olcu *d1 = new Olcu; // C++ tarzı yer ayırma
    //Olcu *d1 = (Olcu *)malloc(2*sizeof(int)); // C tarzı yer ayırma

    cout << "uzunluk(metre) giriniz: ";      cin >> d1->metre;
    cout << "uzunluk(cmetre) giriniz:: ";    cin >> d1->cmetre;

    cout << d1->metre << " m " << d1->cmetre << " cm" << endl;

    system("pause");
    return 0;
}
```

- **new** komutuyla ayrılan yerlerin **delete** ile boşaltılması gereklidir.
- Aksi takdirde bellek sızıntıları (memory leak) meydana gelir. Özellikle uzun süreli çalışacak ya da fazla miktarda dinamik bellek kullanan programlarda bellek sızıntıları çok büyük sorumlara neden olabilir.
- Program sonlandığında ayrılan yerlerin kontrolü işletim sistemine geçer.



Fonksiyon Parametresi Olarak İşaretçiler

- Bir fonksiyona parametre aktarmanın iki yolu vardır.
 - Değerle Çağırma
 - Adresle Çağırma
 - Adresle Çağırma İşaretçiler ile yapılır.

```
//Değer ile Çağırma
```

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

void arttir(int x) {
    x++;
}

int main() {
    int sayi = 7;
    cout << "sayi: " << sayi << endl;

    arttir(sayi);
    cout << "sayi: " << sayi << endl;

    system("pause");
    return 0;
}
```

```
//Adres ile Çağırma
```

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

void arttir(int *x) {
    (*x)++;
}

int main() {
    int sayi = 7;
    cout << "sayi: " << sayi << endl;

    arttir(&sayi);
    cout << "sayi: " << sayi << endl;

    system("pause");
    return 0;
}
```



Fonksiyon Parametresi Olarak İşaretçiler...

➤ ÖRNEK: ⇒ [5]_fonksiyon.cpp

```
void centimize(double* ptrd)
{
    /*ptrd= *ptrd * 2.54;

    for (int j = 0; j < MAX; j++) {
        *ptrd *= 2.54;
        ptrd++; //ptrd varray dizisi elemanlarına işaret eder
    }
}

double varray[MAX] = { 10.0, 43.1, 95.9, 59.7, 87.3 };

//double var=10.0;
//centimize(&var);

centimize(varray);

for (int j = 0; j < MAX; j++)
    cout << "vararray[" << j << "]=" << varray[j] << " cm" << endl;
```



Fonksiyon Parametresi Olarak İşaretçiler...

➤ Gönderilen adresin içeriğinin değiştirilememesini garanti etmek için **const** ifadesi kullanılabilir.

➤ ÖRNEK: ⇒ [6]_fonksiyon.cpp

- strcpy() tanımlamasında kullanılan const char* hedef argümanı, hedefin işaret ettiği karakterlerin strcpy() fonksiyonu tarafından değiştirilemeyeceğini belirtir.
- Hedef işaretçisinin kendisinin değiştirilemeyeceği anlamına gelmez. Bunu yapmak için argüman tanımlaması char* const hedef şeklinde yapılmalıdır.

```
void copystr(char* hedef, const char* kaynak)
{
    while (*kaynak)           //null karakterine kadar,
        *hedef++ = *kaynak++; //kaynaktaki karakterleri hedefe kopyalar
    *hedef= '\0';            //hedefi sonlandır
}

const char* str1 = "SAKARYA UNIVERSITESI";
char str2[80];

copystr(str2, str1);        //str1'i str2'ye kopyala
cout << str2 << endl;

system("pause");
return 0;
```



Fonksiyon Parametresi Olarak İşaretçiler...

➤ ÖRNEK: ⇒ [7]_fonksiyon.cpp

```
void buyukHarfeCevir(char *sPtr)
{
    while (*sPtr != '\0')          // Metni son karaktere kadar tara
    {
        if (islower(*sPtr))       // Karakter küçük harf ise,
            *sPtr = toupper(*sPtr); // Büyük harfe çevir
        //else
        //    *sPtr = tolower(*sPtr);

        sPtr++;
    }
}
```

```
char metin[] = "Bilgisayar Muhendisligi";

cout << "Cevirme isleminden onceki metin : " << metin << endl;

buyukHarfeCevir(metin);

cout << "Cevirme isleminden sonraki metin : " << metin << endl;
```

Fonksiyon İşaretçileri

- Bir fonksiyona işaret ederler
- Fonksiyon işaretçisi, bir fonksiyona parametre olabilir
- Fonksiyon işaretçisi bir fonksiyondan geri döndürülebilir
- Birden fazlası dizi olarak tanımlanabilir
- Başka bir fonksiyon işaretçisine atanabilir

```
// int parametresi alan ve int döndüren bir fonksiyon işaretçisi  
int(*fonk1) (int);  
  
// iki adet int parametre alan ve int işaretçisi döndüren fonksiyon  
int* fonk2(int, int);
```

Fonksiyon İşaretçileri...

➤ ÖRNEK: ⇒ [8]_fonksiyon.cpp

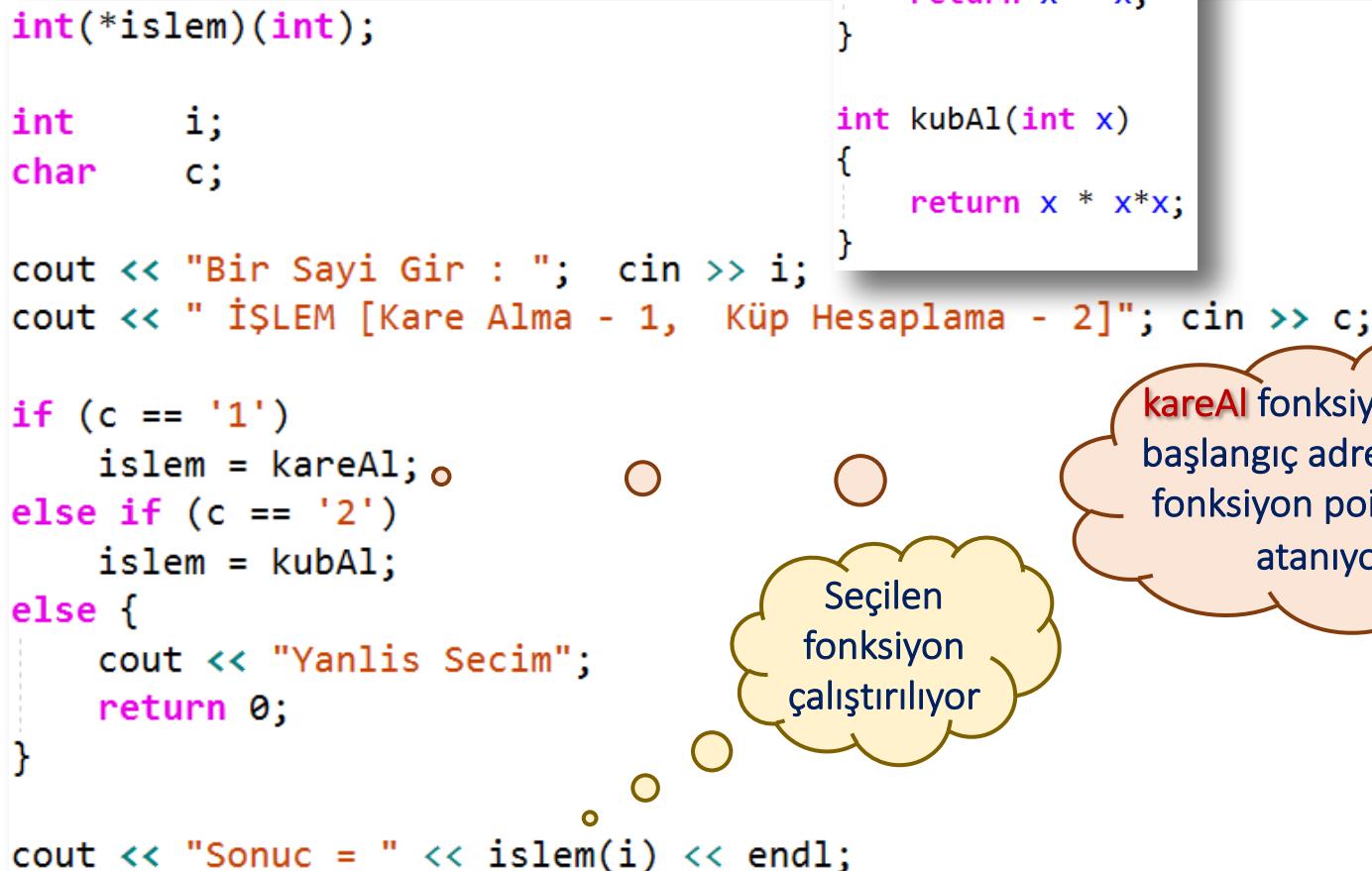
```
int(*islem)(int);

int      i;
char     c;

cout << "Bir Sayı Gir : "  cin >> i;
cout << " İŞLEM [Kare Alma - 1,  Küp Hesaplama - 2]"  cin >> c;

if (c == '1')
    islem = kareAl; ○
else if (c == '2')
    islem = kubAl;
else {
    cout << "Yanlış Seçim";
    return 0;
}

cout << "Sonuc = " << islem(i) << endl;
```



The diagram illustrates the state of memory for the code above. A pointer variable `islem` contains the address of either `kareAl` or `kubAl`. The `kareAl` function is shown with its code: `int kareAl(int x){ return x * x;}`. The `kubAl` function is also shown with its code: `int kubAl(int x){ return x * x*x;}`. Below the functions, the variable `i` contains the value 5. A yellow thought bubble says "Seçilen fonksiyon çalıştırılıyor". A red thought bubble says "kareAl fonksiyonunun başlangıç adresi islem fonksiyon pointerına atanıyor".

Dinamik Bellek Yönetimi

- **new** ve **delete** operatörleri
- new komutu ile tahsis edilen yerin başlangıç adresi döndürür

new veri_tipi [uzunluk]

```
int *ptr1;
int *ptr2 = new int[3];      // 3*sizeof(int) kadar yer tahsis et

delete ptr1;                // ptr için tahsis edilen yeri serbest bırakır (boşaltır)
delete[] ptr2;              // elemanlar dizisi için tahsisli yeri boşaltır
```



Dinamik Bellek Yönetimi...

➤ Bellek tahsisinin başarılı olup olmadığı C++'da iki yol ile belirlenir.

① Exception (istisna) kullanımı

- Yer tahisi başarısız olduğunda **bad_alloc** tipinde bir istisna yollanır.
- **bad_alloc** istisnası yollandığında programın çalışması sonra erdirilir.

② Nothrow

- Program sonlandırılmaz ve istisna yollanmaz.
- **new** operatörü tarafından boş (null) bir işaretçi döndürülür.
- Program çalışmayı devam ettirir.

```
int *ptr;
ptr = new (nothrow) int[3];

if (ptr)
    cout << "Bellek Tahisi Gerçekleştirildi";
else
    cout << "Bellek Tahisi Gerçekleştirilmedi";
```



Dinamik Bellek Yönetimi...

➤ ÖRNEK: ⇒ [9]_dinamikdizi.cpp

```
long *dizi;

int n;
cout << "Kac sayi gireceksiniz?"; cin >> n;

dizi = new long[n];
//dizi=NULL;

if (dizi == NULL)
    return 0;

for (int i = 0; i < n; i++)
{
    cout << "sayiyi giriniz ";
    cin >> dizi[i];
}

cout << "Girdiginiz degerler :   ";

for (int i = 0; i < n; i++)
    cout << dizi[i] << ", ";

delete[] dizi;
```

- Program çalışırken dinamik olarak bellek ayırmak(**new**)/silmek(**delete**).
- Bellek ayrılamaz ise NULL değeri döndürülür.



Dinamik Diziler...

➤ ÖRNEK: ⇒ [10]_dinamikdizi.cpp

```
int *ptrInt = new int[n];           // heap bölgesinde n kadar tamsayılık yer açıldı

cout << ptrInt << endl;           // heap bölgesinde açılan n kadar tamsayılık
                                  // yerin başlangıç adresi

ptrInt[0] = 10;
cout << *(ptrInt + 0);           // 10 değerini yazar...

char *mesaj = new char[10];

cout << mesaj << endl;           // mesaj için ayrılan yerin başlangıç adresini
                                  // yazdırmasını bekleriz ama karakter katarı gösteren
                                  // bir işaretçi değişkeni olduğundan karakter katarı yazdırılır.

int *ptr2 = (int*)mesaj;
cout << ptr2 << endl;           // mesaj için ayrılan yerin başlangıç adresi yazdırılır

delete[] ptrInt;
delete[] mesaj;

ptrInt = NULL;
mesaj = NULL;
```

Dinamik Diziler...

➤ ÖRNEK: ⇒ [11]_dinamikdizi.cpp

```
Ogrenci * ogrenciler[100];  
  
int ogrenciSayisi = 0; cin >> ogrenciSayisi;  
  
for (int i = 0; i < ogrenciSayisi; i++)  
    ogrenciler[i] = new Ogrenci();  
  
for (int i = 0; i < ogrenciSayisi; i++)  
{  
    cout << "NUMARA      : "; cin >> ogrenciler[i]->numara;  
    cout << "AD          : "; cin >> ogrenciler[i]->ad;  
    cout << "SOYAD       : "; cin >> ogrenciler[i]->soyad;  
    cout << "GENEL ORT. : "; cin >> ogrenciler[i]->genelOrtalama;  
    //getline(cin,ogrenciler[i].genelOrtalama);  
    cout << "\n-----\n";  
}
```

```
struct Ogrenci  
{  
    char numara[10];  
    char ad[10];  
    char soyad[10];  
    float genelOrtalama;  
};
```

Dinamik Diziler...

➤ ÖRNEK: ⇒ [12]_dinamikdizi.cpp

```
int ogrenciSayisi = 0;  cin >> ogrenciSayisi;

Ogrenci* ogrenciler = new Ogrenci[ogrenciSayisi];

for (int i = 0; i < ogrenciSayisi; i++)
{
    cout << "NUMARA      : "      >> ogrenciler[i].numara;
    cout << "AD          : "      >> ogrenciler[i].ad;
    cout << "SOYAD       : "      >> ogrenciler[i].soyad;
    cout << "GENEL ORT. : "      >> ogrenciler[i].genelOrtalama;
    cout << "\n-----\n";
}

for (int i = 0; i < ogrenciSayisi; i++)
{
    cout << "\n-----\n";
    cout << "NUMARA      : " << ogrenciler[i].numara << endl;
    cout << "AD          : " << ogrenciler[i].ad << endl;
    cout << "SOYAD       : " << ogrenciler[i].soyad << endl;
    cout << "GENEL ORT. : " << ogrenciler[i].genelOrtalama << endl;
}

delete[] ogrenciler;

ogrenciler = NULL;
```

```
struct Ogrenci
{
    string numara;
    string ad;
    string soyad;
    float genelOrtalama;
};
```

Dinamik Diziler...

➤ ÖRNEK: ⇒ [13]_dinamikdizi_isaretpci.cpp

```
Ogrenci* pOgr = new Ogrenci[ogrenciSayisi];

for (int i = 0; i < ogrenciSayisi; i++)
{
    cout << "NUMARA      : ";      cin >> pOgr->numara;
    cout << "AD          : ";      cin >> pOgr->ad;
    cout << "SOYAD       : ";      cin >> pOgr->soyad;
    cout << "GENEL ORT. : ";      cin >> pOgr->genelOrtalama;
    cout << "\n-----\n";
    pOgr++;
}

pOgr -= ogrenciSayisi;
cout << setw(20) << "AD" << setw(20) << "SOYAD"
    << setw(20) << "NUMARA" << setw(20) << "ORTALAMA" << endl;

for (int i = 0; i < ogrenciSayisi; i++)
{
    cout << setw(20) << pOgr->ad;
    cout << setw(20) << pOgr->soyad;
    cout << setw(20) << pOgr->numara;
    cout << setw(20) << pOgr->genelOrtalama;
    pOgr++;
    cout << endl;
}

pOgr -= ogrenciSayisi;
delete[] pOgr;
pOgr = NULL;
```

```
struct Ogrenci
{
    string numara;
    string ad;
    string soyad;
    float genelOrtalama;
};
```

Dinamik Diziler...

➤ ÖRNEK: ⇒ [14]_dinamikdizi_index.cpp

```
Ogrenci* pOgr = new Ogrenci[ogrenciSayisi];

for (int i = 0; i < ogrenciSayisi; i++)
{
    cout << "NUMARA      : ";      cin >> pOgr[i].numara;
    cout << "AD          : ";      cin >> pOgr[i].ad;
    cout << "SOYAD       : ";      cin >> pOgr[i].soyad;
    cout << "GENEL ORT. : ";      cin >> pOgr[i].genelOrtalama;
    cout << "\n-----\n";
}

cout << setw(20) << "AD" << setw(20) << "SOYAD"
    << setw(20) << "NUMARA" << setw(20) << "ORTALAMA" << endl;

for (int i = 0; i < ogrenciSayisi; i++)
{
    cout << setw(20) << pOgr[i].ad;
    cout << setw(20) << pOgr[i].soyad;
    cout << setw(20) << pOgr[i].numara;
    cout << setw(20) << pOgr[i].genelOrtalama;
    cout << endl;
}

delete[] pOgr;

pOgr = NULL;
```

```
struct Ogrenci
{
    string numara;
    string ad;
    string soyad;
    float genelOrtalama;
};
```

Dinamik Nesneler

➤ ÖRNEK: ⇒ [15]_OBS.cpp, ogrenci.cpp, ogrenci.h

```
int ogrenciSayisi = 0;
cin >> ogrenciSayisi;

string gecici;
getline(cin, gecici);

Ogrenci* ogrenciler = new Ogrenci[ogrenciSayisi];

Ogrenci ogr1;

Ogrenci* ogr2 = new Ogrenci();

for (int i = 0; i < ogrenciSayisi; i++)
{
    ogrenciler[i].bilgiGirisi();
    //cout<<(ogrenciler+i)->getAd();
}

for (int i = 0; i < ogrenciSayisi; i++)
{
    ogrenciler->bilgiYazdir();
    ogrenciler++;
}

ogr1.bilgiGirisi();
cout << ogr1.getAd();

ogr2->bilgiGirisi();
```

```
class Ogrenci
{
private:
    string ad;
    string soyad;
    string numara;
    int notOrtalaması;
public:
    Ogrenci();
    ~Ogrenci();
    void bilgiGirisi();
    void bilgiYazdir();
    void setAd(string ad);
    void setSoyad(string soyad);
    void setNumara(string numara);
    void setNotOrtalaması(int notOrtalaması);
    string getAd() const;
    string getNumara() const;
    string getSoyad() const;
    int getNotOrtalaması() const;
};
```

Dinamik Nesneler

➤ ÖRNEK: ⇒ [16]_bolum.cpp, bolum.h, bolum.cpp, personel.h, tarih.h

```
Bolum *bilgisayar = new Bolum("Bilgisayar Muhendisligi");
Bolum emk("Endüstri Müh.");

Personel *ayse = new Personel();
ayse->bilgiGir();

bilgisayar->setBolumBaskani(ayse);

Personel* ahmet = new Personel();
emk.setBolumBaskani(ahmet);
ahmet->bilgiGir();
emk.setBolumKoordinatoru(ayse);

bilgisayar->yazdir();
emk.yazdir();

delete ahmet;
delete ayse;
```

delete komutuyla nesne yok edilirken yıkıcısı çalışır.

void İşaretçi

- Herhangi bir tipi gösterebilir
- * işaretini doğrudan kullanılamaz
- Tip dönüşümü gerektirebilir.

```
int      intvar;           //integer değişken
float    flovar;          //float değişken

int*    ptrint;           //int gösteren işaretçi
float*  ptrflo;          //float gösteren işaretçi

ptrflo = (float*)intvar;
```

void İşaretçi...

➤ ÖRNEK: ⇒ [17]_void.cpp

```
int      intvar;           //integer değişken
float    flovar;          //float değişken

int*     ptrint;          //int gösteren işaretçi
float*   ptrflo;          //float gösteren işaretçi
void*   ptrvoid;          //void gösteren işaretçi

ptrint = &intvar;         //int* to int*
//ptrint = &flovar;        //Hata, float* to int*
//ptrflo = &intvar;        //Hata, int* to float*
ptrflo = &flovar;          //float* to float*

ptrvoid = &intvar;         //int* to void*
ptrvoid = &flovar;          //float* to void*
```



void İşaretçi...

➤ ÖRNEK: ⇒ [18]_void.cpp

```
char ch = 'A';
int i = 5;

arttir(&ch, sizeof(char));
arttir(&i, sizeof(int));

cout << "Karakter : " << ch << endl;
cout << "Sayı : " << i << endl;
```

```
void arttir(void *veri, int pSize)
{
    if (pSize == sizeof(char))
    {
        char *pChar;
        pChar = (char*)veri;
        ++(*pChar);
    }
    else if (pSize == sizeof(int))
    {
        int *pInt;
        pInt = (int*)veri;
        ++(*pInt);
    }
}
```



KAYNAKLAR

- Deitel, C++ How To Program, Prentice Hall
- Horstmann, C., Budd,T., Big C++, Jhon Wiley&Sons, Inc.
- Robert Lafore, Object Oriented Programming in C++, Macmillan Computer Publishing
- Prof. Dr. Celal ÇEKEN, Programlamaya Giriş Ders Notları
- Prof. Dr. Cemil ÖZ, Programlamaya Giriş Ders Notları