



BM102

Algoritma ve Programlama II

Göstergeler (Pointer)



Ders Hakkında

- Her konu sonunda ödeviniz olacaktır.
- <http://akademik.duzce.edu.tr/arafatsenturk/Profil/DokAra>
- https://www.turnitin.com/login_page.asp?lang=tr_TR

Sınıf Numarası **24036213**

Kayıt anahtarı **BM102**

- Ödevler bireysel yapılacaktır.
- Yaklaşık 10 adet ödev olacaktır.
- Kodlar (ödevler) **öğrencino.pdf** şeklinde yüklenmelidir.
- Ödevler belirtilen gün ve saatte sisteme yüklenmelidir. Aksi halde sistem otomatik kapandığı için ödeviniz geçersiz olacaktır.

Ders Hakkında

- Ödevleriniz sistem üzerinde sınıf ve internet üzerinden benzerlik analizine bağlı tutulacaktır. Yüzde 30'dan fazla benzerlik raporu olan ödevler geçersiz sayılacaktır.
- Dersin puanlaması aşağıdaki şekilde olacaktır;
 - %40 Vize
 - %10 Ödev
 - %50 Final
- Her öğrenci yoklama kağıdına ancak kayıtlı olduğu gruba imza atabilir.
- Kayıtlı olduğunuz gruba mazeretinizden dolayı katılamazsanız diğer gruplarda derse katılabiliyorsunuz.
- Her dersin sonunda Quiz yapılacaktır.
- Quiz'den dersi alanların genelinden en yüksek ilk 3 puanı alan öğrenciye FİNAL notuna 5 Puan eklenecektir.

Ders Hakkında

- Müfredatımız şu şekildedir;

KONU
Göstergeler(Pointer)
Sınıf (Class)
Yapıcı ve Yıkıcı Fonksiyonlar
Bileşim, Arkadaş ve Diğer Araçlar
Operatör Yükleme
Miras
Çoklu işlev
Kural Dışı Durum Yönetimi
Dosya İşlemleri

İçerik



1. Gösterge Tanımı
 2. Gösterge Operatörleri
 3. Dinamik Bellek Kullanımı
- Çözümlü Sorular***

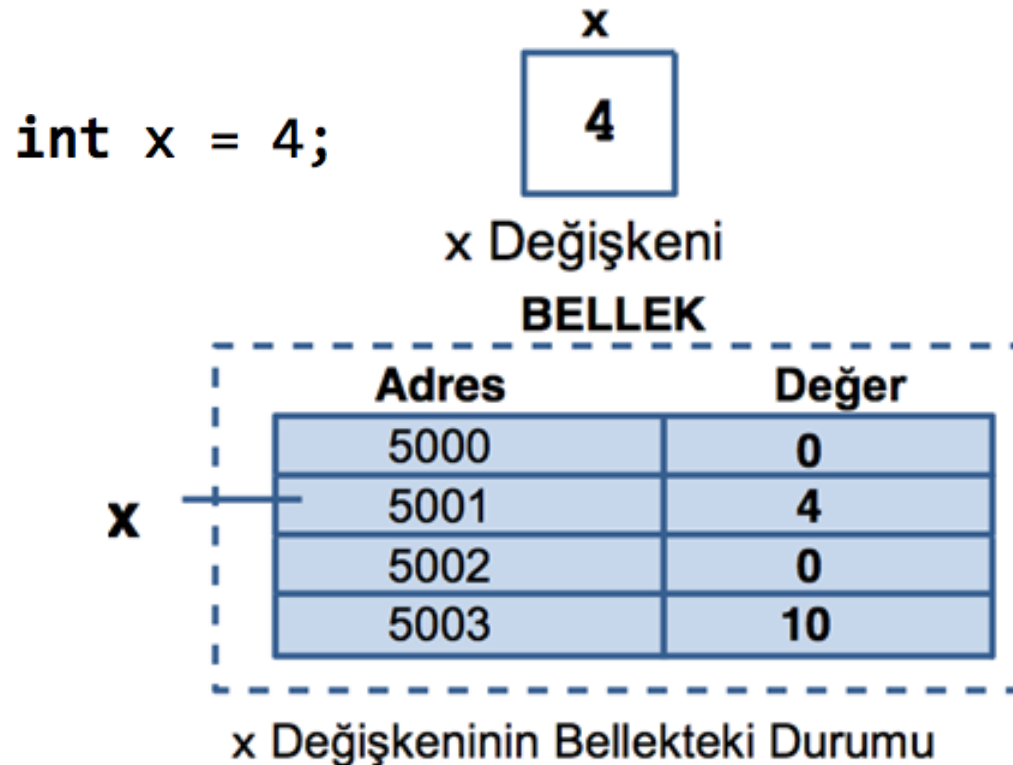
Hedefler



- Gösterge kavramını anlatma
- Göstergelerin kullanım amacını anlatma
- Farklı veri tiplerinde gösterge tanımlama
- Göstergeler ve bellek alanlarındaki durumu takip etme
- Gösterge operatörlerinin kullanımını anlama
- Verilen göstergenin gösterdiği bellek alanındaki değere erişme
- Göstergenin içerdiği bellek adresine erişme
- Dinamik bellek kullanımının önemini anlatma
- new ve delete komutlarının görevini açıklama
- Verilen bir programı dinamik bellek kullanarak yeniden yazma
- Yığın kavramını açıklama

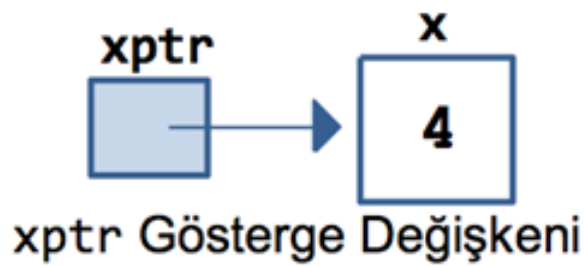
Göstergeler

- Değişkenler, verileri saklamak için kullanılan bellek hücrelerine verilen isimler olarak tanımlanır.



Göstergeler

- Değişkenlerin kapladıkları hücrelerin adreslerini saklamak için başka değişkenler de kullanabiliriz.
- Bu değişkenlere *gösterge (pointer)* adı verilir.



BELLEK

	Adres	Değer
x	5000	0
	5001	4
	5002	0
	5003	10
	5004	10
xptr	5005	5001
	5006	10

x Değişkeni ve xptr Gösterge Değişkeni

1. Gösterge Tanımı

- Genel olarak gösterge tanımını aşağıdaki sözdizimlerden birini kullanarak yapmamız gerekir.

*VeriTipi *GöstergeAdı;*

veya

VeriTipi GöstergeAdı;*

Örnek:

```
int *xptr;  
int*  xptr;  
float  *a, *b, *c;
```

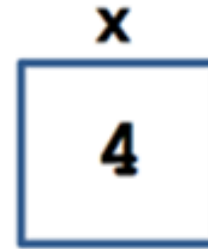


Göstergelerin tipleri gösterilen hücrelerde saklanan değerlerin veri tipleriyle uyumlu olmalıdır.

2. Gösterge Operatörleri (& ve *)

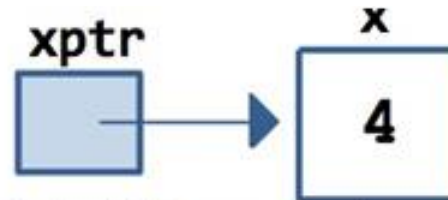
- Aşağıda kullanılan ‘&’, *adres* operatörüdür.
- Bu durumda, &x, x’in adresi anlamına gelmektedir ve bu komut sonucunda xptr değişkenine x’in adresi atanır.

```
int x = 4;
```



x Değişkenine Değer Atama

```
xptr = &x;
```

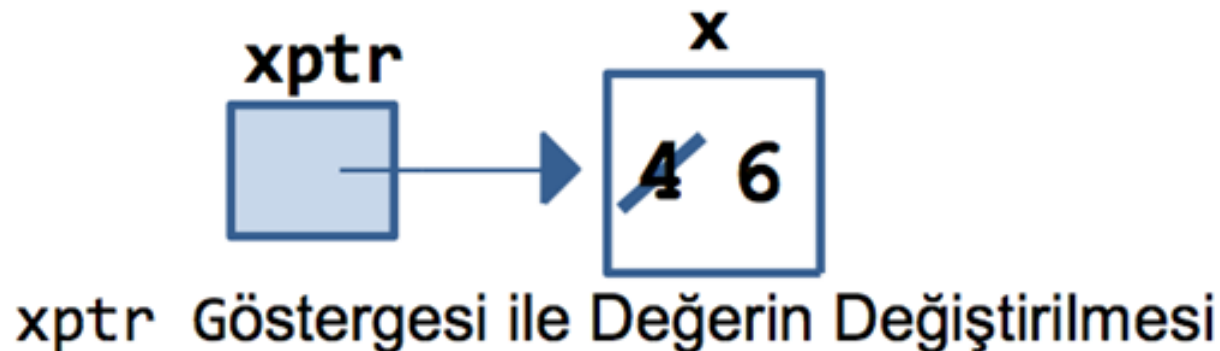


x değişkenini gösteren xptr göstergesi

2. Gösterge Operatörleri (& ve *)...

- ‘ * ’ , *yönlendirme* operatörünün göstergeyi tanımlamanın yanı sıra bir görevi daha vardır.
- *xptr ifadesi, xptr'nin gösterdiği hücrenin içeriğine ulaşmak için de kullanılır.

`*xptr = 6;`



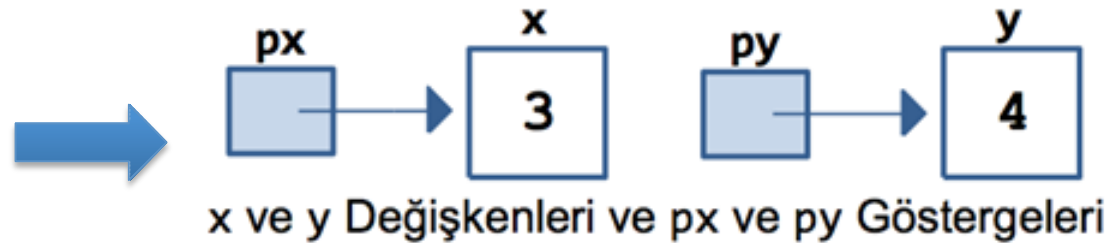
2. Gösterge Operatörleri (& ve *)...

- Aşağıdaki program parçasını inceleyelim.

```
1  int *px, *py;  
2  int x=3, y=4;  
3  px = &x;  
4  py = &y;  
5  py = px;  
6  *py = 5;  
7  cout<<*px;
```

2. Gösterge Operatörleri (& ve *)...

```
1 int *px, *py;  
2 int x=3, y=4;  
3 px = &x;  
4 py = &y;
```



```
5 py = px;
```



```
6 *py = 5;
```



```
7 cout<<*px;
```



2. Gösterge Operatörleri (& ve *)...

- Bir göstergenin bellekte herhangi bir hücreyi göstermesini istemediğimiz zaman **NULL** isimli sabiti kullanırız.
- **NULL** göstergenin bellekte hiç bir yeri göstermediğini belirten 0 değeridir.

```
int *gos = NULL;
```

3. Dinamik Bellek Kullanımı

- Programın çalışması esnasında özel komutlar kullanarak bellekten yer alınıp kullanılmasına dinamik bellek kullanımı adı verilir.
- Dinamik değişkenler için bellekte ayrılan yere **yığın (heap)** adı verilir.
- Dinamik bellek kullanımını **new** ve **delete** komutları ile gerçekleştiririz.

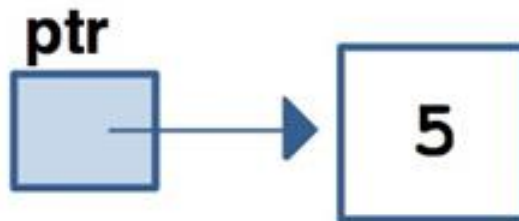
3. Dinamik Bellek Kullanımı...

- **new** komutu ile bellekten istediğimiz veri tipi boyutunda bir yer alabiliriz.

`new VeriTipi` // Tek bir yer almak için kullanılır

`new VeriTipi[ElemanSayisi]` // Bir dizi yer almak için kullanılır

```
int *ptr;           // ptr adlı tamsayı gösterecek bir gösterge tanımlanır.  
ptr = new int;      // Bellekten tamsayı kadar yer ayrılır, yerin adresi ptr'ye atanır.  
*ptr = 5;           // Bellekten alınan yeni yere 5 atanır.
```



ptr Göstergesinin Dinamik Bellek ile Kullanımı

3. Dinamik Bellek Kullanımı...

- **Gösterge** (pointer) tanımı ve **new** komutunu tek bir komut halinde de yazabiliriz.

```
int *ptr = new int;
```

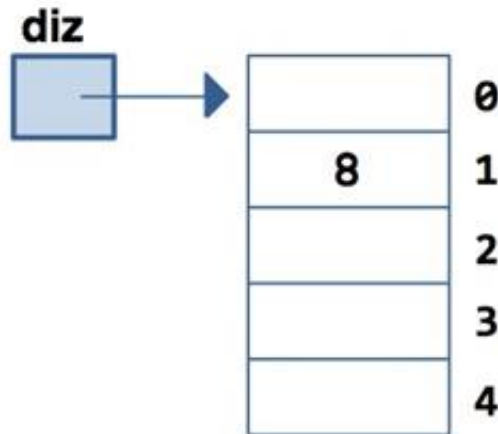
- Dinamik bir değişkenin tanımlanması sırasında ilk ataması da yapılabilir.

```
int *ptr=new int(5);
```

3. Dinamik Bellek Kullanımı...

- Dinamik bir dizinin nasıl oluşturulduğunu bir örnekle inceleyelim.

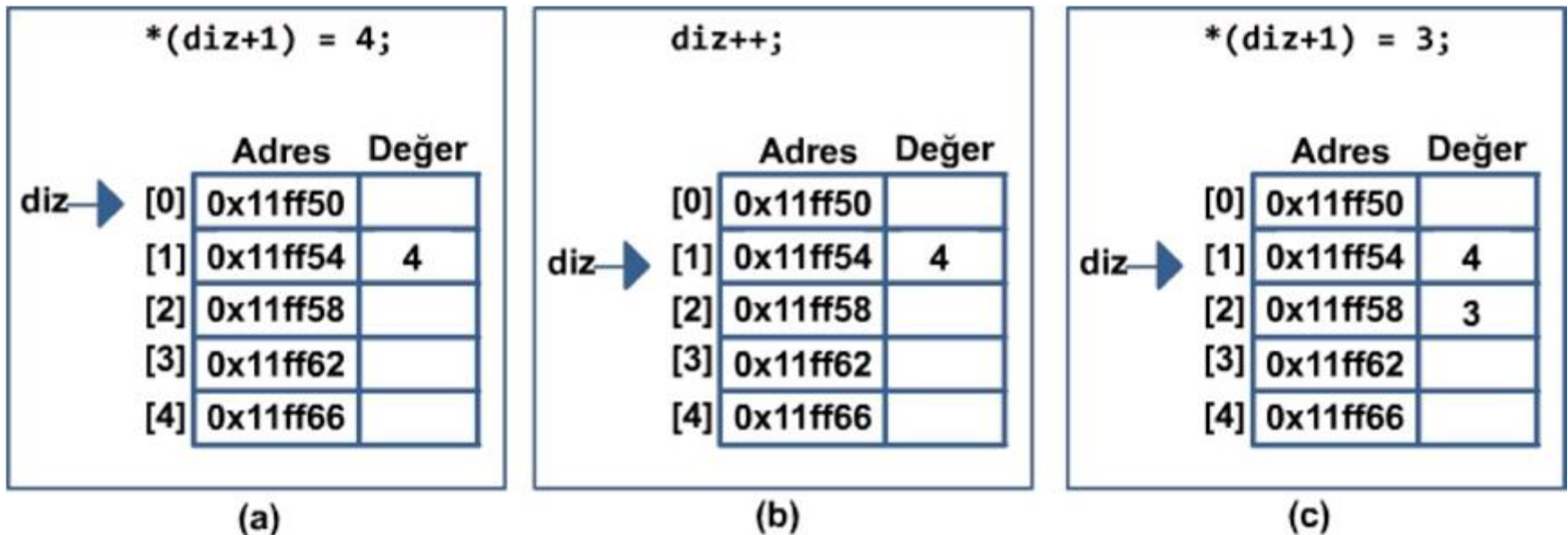
```
int *diz;           // diz isimli tamsayı gösterecek bir gösterge tanımlanır.  
diz=new int[5];     // Bellekten 5 tamsayı dizisi için yer ayrılır ve  
                    // dizinin adresi diz'e atanır.  
diz[1]= 8;          // Bellekten alınan dizinin 1. indeksteki elemanına 8 atanır.
```



diz Dinamik Dizisi

3. Dinamik Bellek Kullanımı...

- Göstergeler ile bellekteki bir alana atama, okuma ve yazma yapmak için **gösterge aritmetiği** (pointer arithmetic) kullanılabilir.

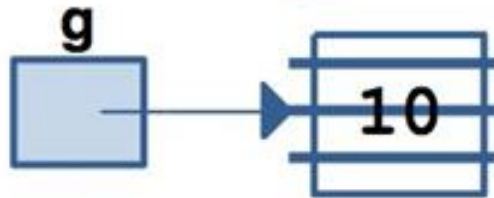


a) diz göstergesinin gösterdiği bellek adresinden bir sonraki adrese, yani dizinin 1. indeksteki elemanına 4 değeri atanır. **b)** diz göstergesi bir sonraki bellek adresini gösterecek şekilde kaydırılır. **c)** diz göstergesinin gösterdiği adresten bir sonraki adrese, yani dizinin 2. indeksteki elemanına 3 değeri atanır.

3. Dinamik Bellek Kullanımı...

- Bellekten **new** ile aldığımız yerler, kullanımları sona erse veya program bitse bile yeniden kullanılamazlar. Bu nedenle bellekten aldığımız hücreleri program bitmeden **delete** komutu ile silmemiz, bir başka deyişle yeniden kullanılabilmeleri için salıvermemiz gerekir.

```
int *g = new int; // g göstergesine tamsayı boyutunda yerin  
                  // adresi atanır  
*g=10;           // g göstergesinin gösterdiği yere 10 atanır.  
cout<<*g;        // Atanan 10 değeri yazdırılır.  
delete g;         // Ayrılan yer belleğe geri döndürülür
```

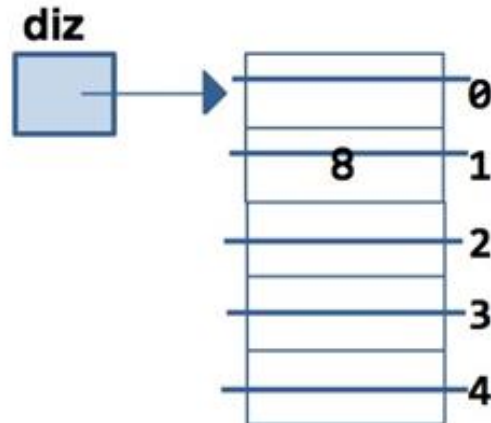


g Göstergesinin Gösterdiği Bellek Hücresi Silinir

3. Dinamik Bellek Kullanımı...

- Dinamik bir dizinin nasıl oluşturup yok edildiğini bir örnekle inceleyelim.

```
int *diz;           // diz isimli tamsayı gösterecek bir gösterge tanımlanır.  
diz=new int[5];     // Bellekten 5 tamsayı dizisi için yer ayrılır ve dizinin  
                    // adresi diz'e atanır.  
  
diz[1]= 8;          // Bellekten alınan dizinin 1. elemanına 8 atanır.  
delete [] diz;      // Yaratılan dizinin tüm elemanları belleğe geri döndürülür.
```

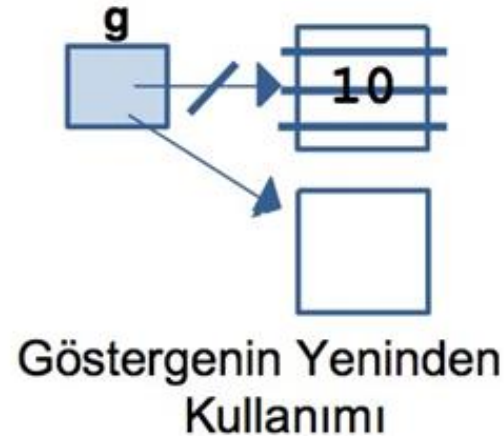


diz Dinamik Dizisinin Yok Edilmesi

3. Dinamik Bellek Kullanımı...

- Daha önce belirttiğimiz gibi **delete** komutu ile geri döndürülen yer, gerektiğinde tekrar kullanılabilir hale gelir. Bir yeri salıverdikten sonra, göstergeyi aşağıdaki örnekte görüldüğü gibi tekrar kullanabiliriz.

```
int *g = new int;  
*g=10;  
delete g;  
g = new int;
```



- new** komutu ile bellekten aldığımız tüm yerleri mutlaka program bitmeden **delete** ile salıvermemiz gerektiğini unutmamalıyız. Aksi takdirde bu yerler program tarafından kullanılamaz.

Örnek

```
#include <iostream>
using namespace std;
void oku(int[],int);
int main()
{
    int mevcut, toplam=0;
    int enYuksek=0;
    int *notlar;
    cout<<"Sinif mevcudunu giriniz: ";
    cin>>mevcut;           // Sınıfın mevcudu okunur
    notlar=new int[mevcut]; // Sınıfın mevcudu kadar dinamik bir dizi yaratılır
    oku(notlar,mevcut);    // Sınıfın notlarını okuyan fonksiyon çağırılır

    for (int i=0;i<mevcut;i++){ // Döngüde en yüksek not ve notların toplamı bulunur
        toplam+=notlar[i];
        if (notlar[i]>enYuksek)
            enYuksek=notlar[i];
    }
    cout<<"Sinif ortalamasi= "<<toplam/mevcut<<endl;
    cout<<"En yuksek not = "<<enYuksek<<endl;
    delete [] notlar;
    return 0;
}
void oku(int notDizi[],int boyut) // Sınıfın notlarını okuyan fonksiyon
{
    cout<<"Notlari giriniz:";
    for (int i=0;i<boyut;i++)
        cin>>notDizi[i];
}
```

Çıktı

Sinif mevcudunu giriniz: 5
Notlari giriniz:30 60 48 92 75
Sinif ortalamasi = 61
En yuksek not = 92

Çözümlü Sorular

Soru

Aşağıdaki fonksiyonları yazınız.

- a. İki tamsayı gösterge parametresi alan ve göstergeler bellekte aynı hücreyi gösteriyorsa `true`, farklı hücreleri gösteriyorsa `false` döndüren bir fonksiyon
- b. İki tamsayı gösterge parametresi alan ve göstergelerin gösterdiği hücrelerdeki değerler aynı ise `true`, değilse `false` döndüren bir fonksiyon.




Çözümlü Sorular...

Cevap

```
a. #include <iostream>
    using namespace std;
    bool fun(int* a, int* b)
    {
        return a==b;
    }
    int main()
    {
        int *x=new int;
        int *y=new int;
        cin>>*x>>*y;
        cout<<fun(x,y);
        return 0;
    }
```

```
b. #include <iostream>
    using namespace std;
    bool fun(int* a, int* b)
    {
        return *a==*b;
    }
    int main()
    {
        int *x=new int;
        int *y=new int;
        cin>>*x>>*y;
        cout<<fun(x,y);
        return 0;
    }
```



Çözümlü Sorular...

Soru

Bir dizinin kopyasını çıkarıp kopyayı döndüren bir fonksiyon yazınız. Fonksiyon kopyalanacak diziyi ve boyutunu parametre olarak alır ve oluşturduğu kopyanın göstergesini döndürür.

Örnek Çıktı

Dizinin boyutunu giriniz: 5

Dizinin elemanlarını giriniz: 2 4 6 8 10

Dizinin kopyası: 2 4 6 8 10



Çözümlü Sorular...

Cevap

```
#include <iostream>
using namespace std;
int* kopya (int* a, int boyut)
{
    int* b = new int[boyut];
    for (int i=0;i<boyut;i++) b[i]=a[i];
    return b;
}
int main()
{
    int a[5]={2,4,6,8,10};
    int *b=kopya(a,5);
    for (int i=0;i<5;i++) cout<<b[i]<<" ";
    return 0;
}
```

