

CENG 111 ALGORİTMALAR VE PROGRAMLAMA

Doç. Dr. Tufan TURACI

tturaci@pau.edu.tr

- Pamukkale Üniversitesi
- Mühendislik Fakültesi
- Bilgisayar Mühendisliği Bölümü
- Hafta 12

12. Hafta Konular

--- İşaretçi Değişkenler (Pointer'lar)

--- Dinamik Bellek Kullanımı

İŞARETÇİLER (Pointer)

- Adresleme kavramı
- İşaretçi kavramı
- İşaretçi Değişkenleri Bildirmek ve Değişkenlere Atama Yapmak
- NULL işaretçiler
- İşaretçi aritmetiği
- Diziler ve İşaretçiler
- C dilinde fonksiyonlar
 - Fonksiyonları adres ile çağırmak

Adresleme Kavramı

- Bilgisayarın ana belleği (**RAM**) sıralı kaydetme gözlerinden oluşmaktadır.
- Buradaki her bir göze adres atanmaktadır.
- Adreslerin değerleri sıfır ve belleğin sahip olduğu üst değere bağlı olarak değişmektedir.

Örnek:

1GB bir bellek,

--- $1024 * 1024 * 1024 = 1.073.741.824$ adet gözden oluşur. (byte)

1TB bir bellek,

--- $1024 * 1024 * 1024 * 1024 = 1.099.511.627.776$ adet gözden oluşur. (byte)

- Bir programlama dilinde tanımlanan bir değişkene değer atandığında, değişkende aşağıdaki 4 temel özellik bulunur.
- Değişkenlerin bellekte kapladığı alanlar sizeof komutu ile bulunabilir.

--- değişkenin adı

--- değişkenin tipi

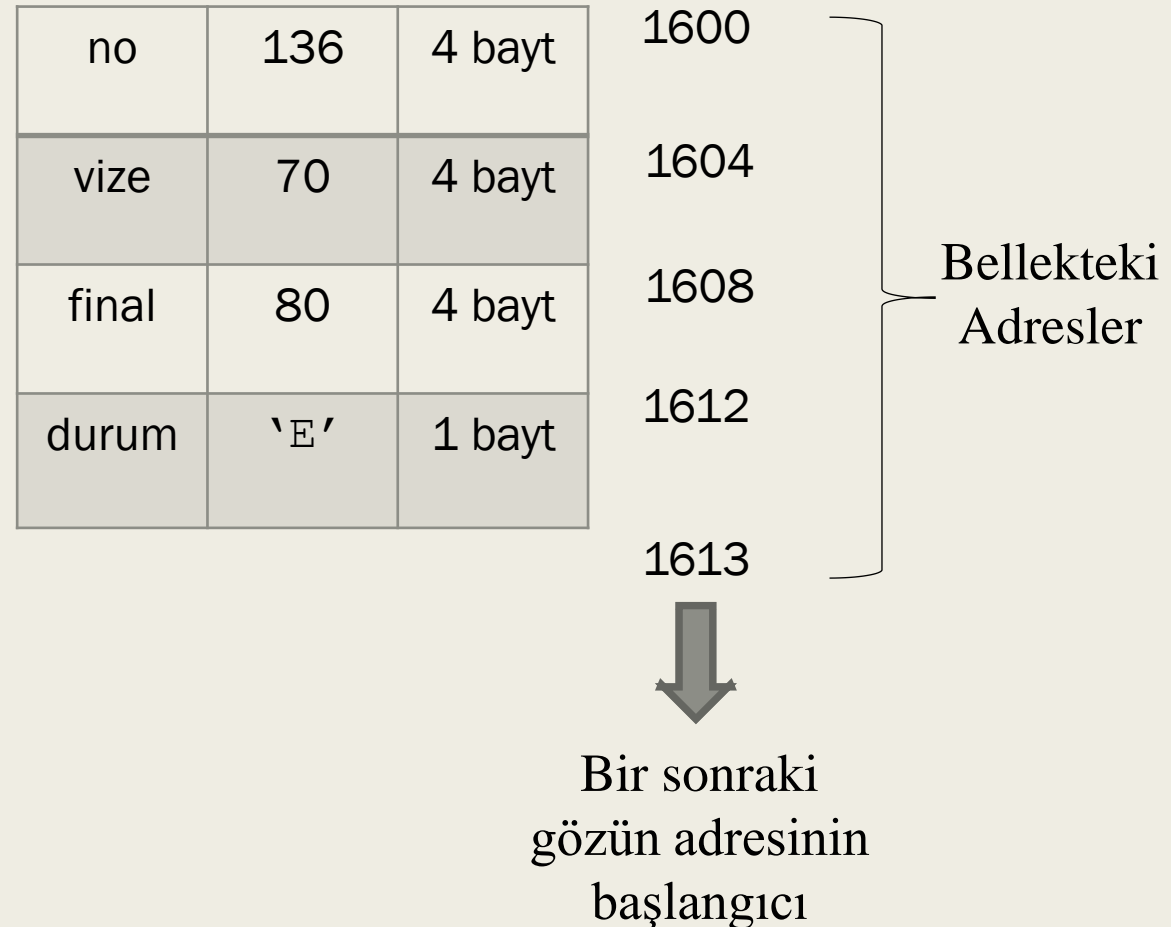
--- değişkenin değeri

--- **değişkenin bellekteki adresi**

- Değişken türlerinin bellekte kapladığı alanlar sizeof komutu ile bulunabilir.

Örnek: Bellek ve Adreslemeye bir örnek

```
int no = 136;  
int vize = 70;  
int final = 80;  
char durum = 'E';
```



- Atama işleminde adreslere yerleşim otomatik olarak yapılmaktadır.
 - no değişkeni 1600, 1601, 1602, 1603 adreslerini işgal eder. (4 byte)
 - vize değişkeni 1604, 1605, 1606, 1607 adreslerini işgal eder. (4 byte)
 - final değişkeni 1608, 1609, 1610, 1611 adreslerini işgal eder. (4 byte)
 - durum değişkeni ise 1612 adresini tutar. (1 byte)

- Örnekteki kod bloğunda değişkenler bilgisayar tarafından belirli bir adreste saklanır. Bu yönteme implicit (kapalı) adresleme denir.
- İşaretçi değişken kullanılarak, işaretçilere verilerin bellekte saklandığı bellek hücrelerinin başlangıç adresleri atanır. Bu yönteme ise explicit (açık) adresleme denir.

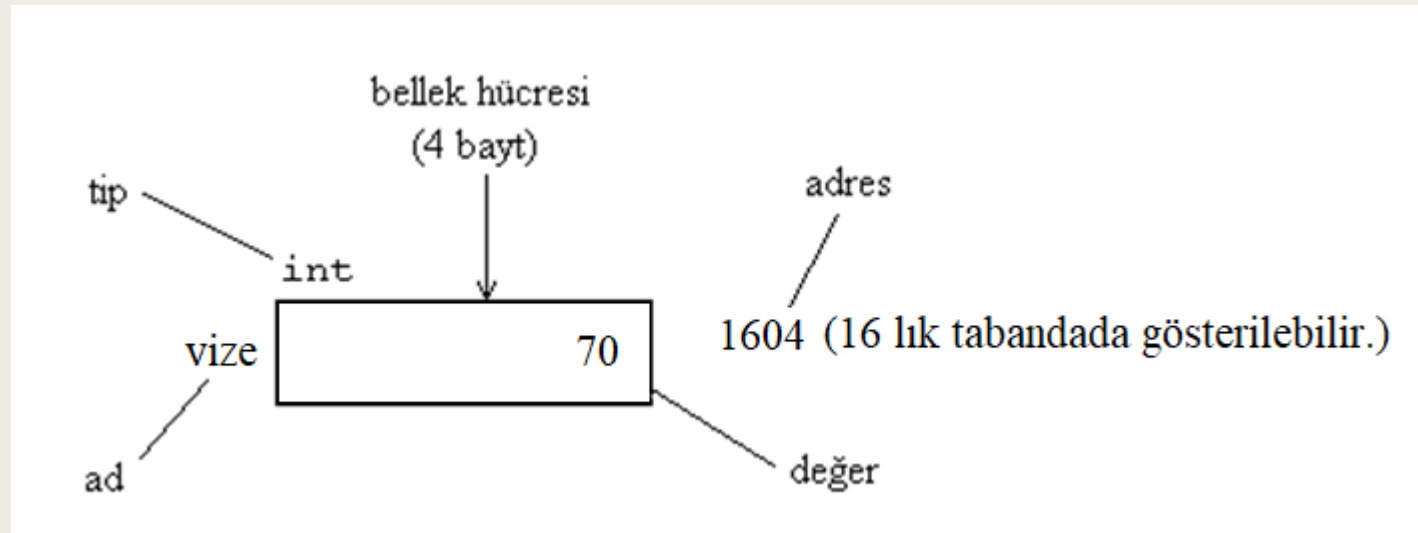
Örnek:

```
int vize = 70;
```

--- vize değişkeni için, bellekte int tipinde (4 bayt büyüklüğünde) bir hücre ayrılır ve o hücreye 70 sayısı ikilik (binary) sayı sistemindeki karşılığı olan 4 baytlık (32 bitlik) karşılığı aşağıdaki gibi yazılır.

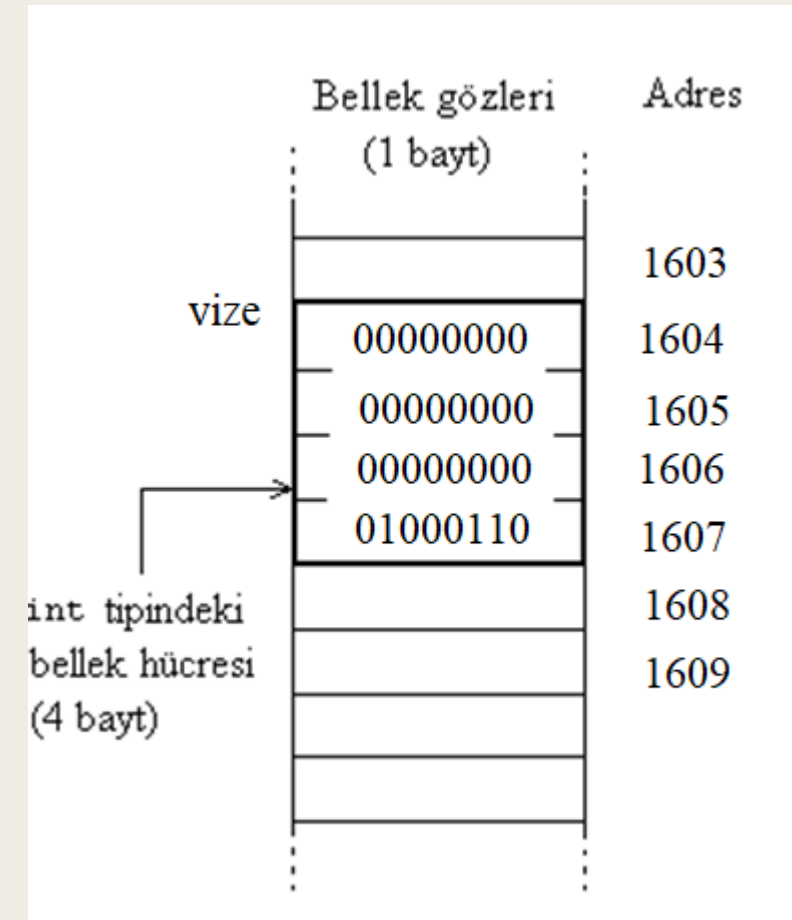
00000000 00000000 00000000 01000110

(70 sayısının ikilik tabandaki gösterimi)



--- Yukarıdaki şekilde, vize değişkeni, program çalıştığı sürece, bellekte 1604. - 1607. numaralı gözler arasındaki 4 baytlık hücreyi kullanmaktadır.

vize değişkeninin bellekteki gerçek konumu ve ikilik düzendeki içeriği aşağıdaki gibidir:



--- Değişkenin bulunduğu adres, & (ampersant) karakteri ile tanımlı adres operatörü ile öğrenilir.

--- Bu operatör bir değişkenin önüne konulduğunda, artık değişkenin adresi ile ilgileniliyor anlamındadır, değişkenin içeriği ile ilgilenilmemektedir.

Örnek:

```
#include<stdio.h>
int main()
{ int vize=70,final=80;
printf ("vize notu= %d\n", vize);
printf ("vize degiskeninin adresi= %d\n", &vize);

printf ("final notu= %d\n", final);
printf ("final degiskeninin adresi= %p\n", &final);
return 0;
}
```

```
#include<stdio.h>
int main()
{ int vize=70,final=80;
printf ("vize notu= %d\n", vize);
printf ("vize degiskeninin adresi= %d\n", &vize);

printf ("final notu= %d\n", final);
printf ("final degiskeninin adresi= %d\n", &final);
return 0;
}
```

```
vize notu= 70
vize degiskeninin adresi= 6487580
final notu= 80
final degiskeninin adresi= 000000000062FE18
-----
```

```
vize notu= 70
vize degiskeninin adresi= 6487580
final notu= 80
final degiskeninin adresi= 6487576
-----
```

İşaretçi (Pointer) Kavramı

- Bir değişkenin adresinin başka bir değişkende saklanması işaretçi değişkenler yardımıyla olur.
- İşaretçiler, değişkenin adresini içeren başka bir değişkendir.
- İşaretçi denmesinin sebebi ilgili değişkenin adresini işaret etmesinden yani göstermesinden kaynaklanır.
- İşaretçi değişkenler fonksiyonların referansa göre çağırma yapmasını sağlarlar.
- Bununla beraber, işaretçiler kullanılarak Bağlı listeler (Linked List), Yığınlar (Stack), Ağaçlar (Tree) gibi büyüyüp küçülebilen dinamik veri yapılarının oluşturulması ve yönetilmesi sağlanır.

- Bir işaretçi değişken, sayısal tipte bir değişkendir.
- İşaretçi değişkenler kullanılmadan önce program içinde tanımlanmalıdır.
- İşaretçi tipindeki değişkenler aşağıdaki şekilde tanımlanır:

tip_adı *isaretci_adı;

- tip_adı herhangi bir veri tipi şeklinde olabilir. Değişkenin önündeki * Karakteri (asterisk) yönlendirme (indirection) operatörü olarak bilinmektedir.
- Asterisk karakteri, değişkenin veri değil bir adres bilgisi tutacağını işaret eder.

Örnek: İşaretçi değişkenin tanımlanmasına bir örnek.

```
int *vize; // vize değişkeninin adres bilgisini saklar.
```

```
int final; // final değişkeninin içindeki değeri saklar.
```

```
float *ort; // ort değişkeninin adres bilgisini saklar.
```

```
float top; // top değişkeninin içindeki değeri saklar.
```

```
char *durum; // durum değişkeninin adres bilgisini saklar.
```


- Bir işaretçi değişkene, bir değişkenin adresini atamak için & (adres) operatörü kullanılır.
- Genel bir kural olmamakla beraber işaretçi değişkenler 'p' harfi ile başlar.
- Aşağıdaki örnekte, pvize işaretçi değişkeni vize değişkeninin saklandığı adresi tutmaktadır.

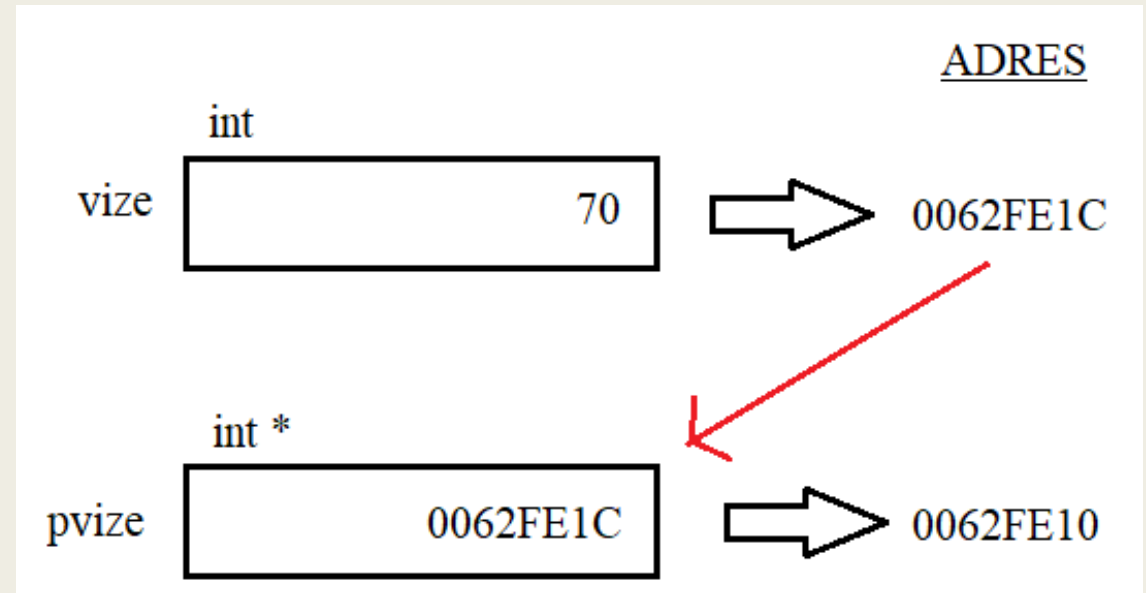
```
int *pvize, vize = 70;
```

```
.
```

```
.
```

```
.
```

```
pvize = &vize;
```



Örnek: `#include <stdio.h>`
`#include <conio.h>`

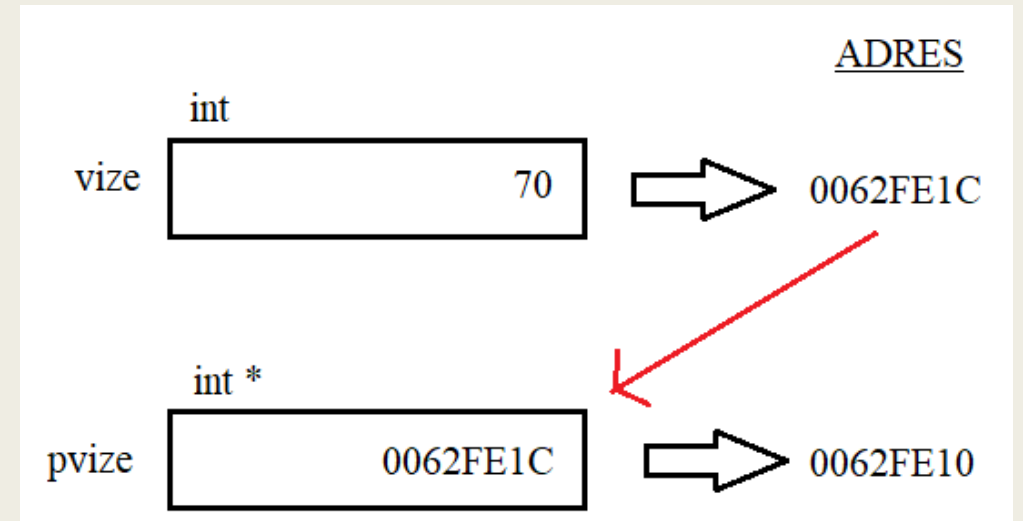
```
int main()
{ int vize=70;
  int *pvize;
  pvize=&vize;
  printf("&vize: %p\n",&vize);
  printf("pvize: %p\n",pvize);
  printf("&pvize: %p\n",&pvize);
  printf("\n");
  printf("vize: %d\n",vize);
  printf("*pvize: %d\n", *pvize);
  printf("\n");
  *pvize=85;
  printf("vize: %d\n",vize);
  printf("*pvize: %d\n", *pvize);
  printf("\n");
  printf("&vize: %p\n",&vize);
  printf("pvize: %p\n",pvize);
  getch();
  return 0;
}
```

```
&vize: 000000000062FE1C
pvize: 000000000062FE1C
&pvize: 000000000062FE10
```

```
vize: 70
*pvize: 70
```

```
vize: 85
*pvize: 85
```

```
&vize: 000000000062FE1C
pvize: 000000000062FE1C
```



```
&vize: 000000000062FE1C  
pvize: 000000000062FE1C  
&pvize: 000000000062FE10
```

```
vize: 70  
*pvize: 70
```

```
vize: 85  
*pvize: 85
```

```
&vize: 000000000062FE1C  
pvize: 000000000062FE1C  
-----
```

--- vize adlı değişkenin içeriğindeki değere, pvize işaretçi değişkeni yardımıyla da ulaşılabilir.

--- pvize değişkeninin önüne yönlendirme operatörü (*) konulduğunda *pvize, vize değişkeninin adresini değil içeriğini tutar.

--- Yani *pvize = 85 komutuyla pvize' nin adresini tuttuğu hücreye 85 atanır.

Sonuç olarak özetleyecek olursak:

- *pvize ve vize, vize adlı değişkenin içeriği ile ilgilidir.
- pvize ve &vize, vize adlı değişkenin adresi ile ilgilidir.
- * yönlendirme operatörüdür.
- & adres operatörüdür.

Örnek:

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int a;
    int *pa;
    a=7;
    pa=&a;
    printf("pa'nin adresi: %p\n",&pa); //16 lık tabanda
    printf("pa'nin adresi: %x\n",&pa); //16 lık tabanda
    printf("pa'nin adresi: %d\n",&pa); //10 luk tabanda
    printf("pa'nin degeri: %p\n", pa);
    printf("a'nin adresi: %p\n", &a);
    printf("a'nin degeri: %d\n", a);
    printf("*pa'nin degeri: %d\n", *pa);
    printf("* ve & birbirlerine tersidir...\n");
    printf("*&pa %p \n&*pa %p",&pa,&*pa);
    getch();
    return 0;
}
```

```
pa'nin adresi: 000000000062FE10
pa'nin adresi: 62fe10
pa'nin adresi: 6487568
pa'nin degeri: 000000000062FE1C
a'nin adresi: 000000000062FE1C
a'nin degeri: 7
*pa'nin degeri: 7
* ve & birbirlerine tersidir...
*&pa 000000000062FE1C
&*pa 000000000062FE1C
-----
```

pa nın değeri, a nın adresidir!!!

NULL İşaretçi

- NULL işaretçiler C programlama dilinde ve birçok kütüphanede sıklıkla kullanılırlar.
- NULL işaretçi kullanmanın amacı, işaretçiye herhangi bir değer atanıp atanmadığının kontrolünü yapmaktır.
- NULL işaretçi, sabit bir değerdir ve hemen hemen tüm dillerde sıfırdır.

```
#include <stdio.h>
```

```
#include <conio.h>
```

```
int main()
```

```
{ int *p;
```

```
  p=NULL;
```

```
  printf("p isaretcisinin degeri: %p\n", p);
```

```
  getch();
```

```
  return 0;
```

```
}
```

```
p isaretcisinin degeri: 0000000000000000
```

```
-----
```

--- NULL işaretçisinin gösterdiği yere bir değer atanamamaktadır.

```
#include <stdio.h>
#include <conio.h>
int main()
{ int *p;
  p=NULL;
  *p=5;
  printf("p isaretcisinin degeri: %p\n", p);
  printf("*p isaretcisinin degeri: %d\n", *p);
  getch();
  return 0;
}
```

```
-----
Process exited after 1.596 seconds with return value 3221225477
Press any key to continue . . .
```

--- Program yukarıdaki şekilde bir çıktı vermemektedir.

--- int main() ana fonksiyonu, 0 değerini döndürmemiştir.

İşaretçi Zinciri

- Genel olarak işaretçi bir değişken, bir değişkenin adresini gösterir.
- Bununla beraber, bir işaretçiye işaret eden başka bir işaretçi de tanımlanabilir.
- Bu duruma **işaretçi zinciri** denir.
- İşaret zincirinin tanımı aşağıdaki şekilde yapılmaktadır:

```
int **ppvize;
```



```
#include <stdio.h>
```

```
#include <conio.h>
```

```
int main()
```

```
{ int vize=70;
```

```
  int *pvize;
```

```
  int **ppvize;
```

```
  pvize=&vize;
```

```
  ppvize=&pvize;
```

```
  printf("vize: %d\n",vize);
```

```
  printf("*pvize: %d\n",*pvize);
```

```
  printf("**ppvize: %d\n",**ppvize);
```

```
  printf("\n");
```

```
  printf("\n");
```

```
  printf("&vize: %p\n",&vize);
```

```
  printf("&pvize: %p\n", &pvize);
```

```
  printf("&ppvize: %p\n", &ppvize);
```

```
  getch();
```

```
  return 0;
```

```
}
```

```
vize: 70
```

```
*pvize: 70
```

```
**ppvize: 70
```

```
&vize: 000000000062FE1C
```

```
&pvize: 000000000062FE10
```

```
&ppvize: 000000000062FE08
```

İşaretçi Aritmetiği

- Bir işaretçiye bir ekleme yapıldığında, o anda tuttuğu adres ile eklenen sayı doğrudan toplanmamaktadır.
- Bir işaretçiye bir ekleme yapıldığında işaretçinin gösterdiği yerdeki veriden hemen sonraki verinin adresini hesaplanmaktadır.
- Genel olarak, bir işaretçiye n sayısını eklemek (veya çıkarmak), bellekte gösterdiği veriden sonra (veya önce gelen) n . elemanın adresini hesaplamaktır.
- char tipinde ise 1 eklendiğinde işaretçi değeri 1 artar.
- int tipinde ise 1 eklendiğinde işaretçi değeri 4 artar.

Örnek:

```
#include <stdio.h>
#include <conio.h>
```

```
int main()
{ char *pc, c='E';
  int *ps, s=35;
  pc=&c;
  ps=&s;
  printf("ilk degerler: *pc= %c *ps= %d \n",*pc,*ps);
  printf("ilk degerlerin adresleri: &pc= %p &ps= %p\n",pc,ps);
  pc++;
  ps++;
  printf("yeni degerler: *pc= %c *ps= %d \n",*pc,*ps);
  printf("yeni degerlerin adresleri: &pc= %p &ps= %p\n",pc,ps);
  getch();
  return 0;
}
```

```
ilk degerler: *pc= E *ps= 35
ilk degerlerin adresleri: &pc= 000000000062FE0F &ps= 000000000062FE08
yeni degerler: *pc= ♀ *ps= 1157627904
yeni degerlerin adresleri: &pc= 000000000062FE10 &ps= 000000000062FE0C
-----
Process exited after 68.01 seconds with return value 0
```

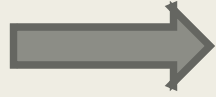
Adres değerleri 1 artırıldığı için bir sonraki hücrenin adresine gidiyor. Burada bir değer olmadığı için rastgele sayılar atanıyor!!!

Diziler ve İşaretçiler

- C programlama dilinde bir dizi ismi, sabit bir işaretçidir.
- Diziler ve işaretçiler, birbirleri yerine hemen hemen her yerde kullanılabilirler.
- Bir dizinin herhangi bir elemanına işaretçiler ile ulaşılabilir.

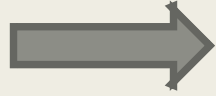
```
int vize[10], *pa, *pb;
```

```
pa = vize;
```



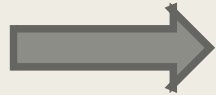
1. Elemanın adresi pa işaretçisine atanıyor.

```
pa = &vize[0];
```



1. Elemanın adresi pa işaretçisine atanıyor.

```
pb = &vize[9]
```



10. Elemanın adresi pb işaretçisine atanıyor

--- Dizi adı bir işaretçi olduğundan dolayı aynı tipteki işaretçiye atanabilir.

Dizilerde; i bir tamsayı olmak üzere aşağıdaki iki ifade aynı anlama gelmektedir.

--- vize[i];

--- *(pa + i);

Örnek 1:

```
#include <stdio.h>
#include <conio.h>
int main()
{ int vize[10]={70,80,98,90,55,35,20,78,65,15}, *pa, *pb, *pc;
  pa=vize;
  pb=&vize[9];
  pc=&vize[2];
  printf("isaretciler ile dizinin elemanlarinin yazdirilmesi:\n");
  printf("dizinin 1. elemani= %d\n", *pa);
  printf("dizinin 10. elemani= %d\n", *pb);
  printf("dizinin 3. elemani= %d\n", *pc);
  getch();
  return 0;
}
```

```
isaretciler ile dizinin elemanlarinin yazdirilmesi:
dizinin 1. elemani= 70
dizinin 10. elemani= 15
dizinin 3. elemani= 98
-----
```

Örnek 2:

```
#include <stdio.h>
```

```
#include <conio.h>
```

```
int main()
```

```
{ int vize[10]={ 70,80,98,90,55,35,20,78,65,15}, *pa;
```

```
  pa=vize;
```

```
  printf("isaretciler ile dizinin elemanlarinin yazdirilmesi:\n");
```

```
  printf("dizinin 1. elemani= %d\n", *pa);
```

```
  printf("dizinin 2. elemani= %d\n", *(pa+1));
```

```
  printf("dizinin 3. elemani= %d\n", *(pa+2));
```

```
  printf("dizinin 4. elemani= %d\n", *(pa+3));
```

```
  printf("dizinin 5. elemani= %d\n", *(pa+4));
```

```
  printf("dizinin 6. elemani= %d\n", *(pa+5));
```

```
  printf("dizinin 7. elemani= %d\n", *(pa+6));
```

```
  printf("dizinin 8. elemani= %d\n", *(pa+7));
```

```
  printf("dizinin 9. elemani= %d\n", *(pa+8));
```

```
  printf("dizinin 10. elemani= %d\n", *(pa+9));
```

```
  getch();
```

```
  return 0;
```

```
}
```

```
isaretciler ile dizinin elemanlarinin yazdirilmesi:
dizinin 1. elemani= 70
dizinin 2. elemani= 80
dizinin 3. elemani= 98
dizinin 4. elemani= 90
dizinin 5. elemani= 55
dizinin 6. elemani= 35
dizinin 7. elemani= 20
dizinin 8. elemani= 78
dizinin 9. elemani= 65
dizinin 10. elemani= 15
-----
```

2. örneğin for ile yapılışı:

```
#include <stdio.h>
#include <conio.h>
int main()
{ int vize[10]={ 70,80,98,90,55,35,20,78,65,15}, *pa, i;
  pa=vize;
  printf("isaretciler ile dizinin elemanlarinin for ile yazdirilmesi:\n");
  for(i=0;i<=9;i++)
  { printf("dizinin %d. elemani= %d\n", i+1,*pa+i);}
  getch();
  return 0;
}
```

```
isaretciler ile dizinin elemanlarinin for ile yazdirilmesi:
dizinin 1. elemani= 70
dizinin 2. elemani= 80
dizinin 3. elemani= 98
dizinin 4. elemani= 90
dizinin 5. elemani= 55
dizinin 6. elemani= 35
dizinin 7. elemani= 20
dizinin 8. elemani= 78
dizinin 9. elemani= 65
dizinin 10. elemani= 15
```


Örnek 3: 10 elemanlı bir dizinin ortalamasını bir fonksiyonda bulan C programı yazınız.
Dizinin elemanlarına işaretçiler ile ulaşınız...

```
dizideki elemanların ortalaması= 64.600  
dizideki elemanların ortalaması fonksiyonda işaretçiler yoluyla hesaplandı...  
-----
```

```
#include <stdio.h>
```

```
#include <conio.h>
```

```
float ortalama(int A[10]);
```

```
int main()
```

```
{ int vize[10]={ 100,80,98,90,55,35,30,78,65,15};
```

```
float sonuc;
```

```
sonuc=ortalama(vize);
```

```
printf("dizideki elemanların ortalaması= %.3f\n", sonuc);
```

```
printf("dizideki elemanların ortalaması fonksiyonda  
işaretçiler yoluyla hesaplandı...\n");
```

```
getch();
```

```
return 0;
```

```
}
```



```
float ortalama(int A[10])  
{ int *pvize, i, top=0; float ort;  
  pvize=A;  
  for(i=0;i<=9;i++)  
    { top=top+*(pvize+i);}  
  ort=top/10.0;  
  return ort;  
}
```



Fonksiyonlar: adres ile çağırma

- & operatörü ile argümanların adresleri fonksiyona geçirilir.
- Hafızadaki gerçek konumunda(adreste) değişiklik yapılmasına izin vermektedir.
- Diziler & ile geçirilemez. Dizi isimleri işaretçidir!!!

* Operatörü

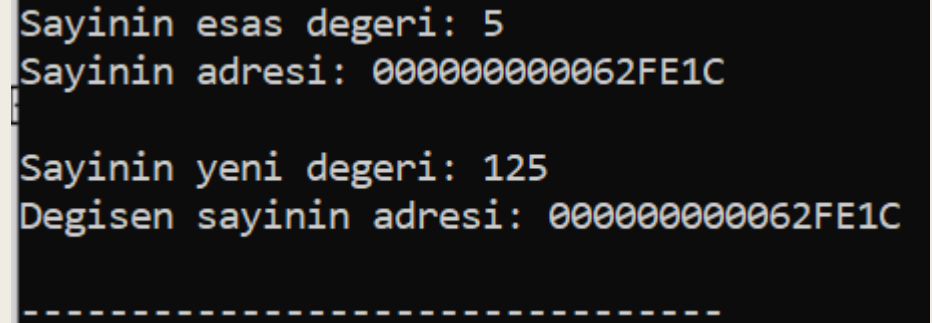
Örnek:

```
void ikikat( int *sayi )  
{  
    *sayi = 2 * ( *sayi );  
}
```

- ikikat fonksiyonuna gelen adresteki değişkenin değerin iki katı hesaplanmıştır.
- Adres değişmemiştir, içerideki değer değişmiştir.

Örnek 1: Ana programdan girilen bir tam sayının küpünü adres ile çağırma yöntemiyle bulan bir C programı yazınız.

```
#include <stdio.h>
#include <conio.h>
void Kup(int *psayi);
int main()
{
    int sayi=5;
    printf("Sayinin esas degeri: %d\n",sayi);
    printf("Sayinin adresi: %p\n",&sayi);
    Kup(&sayi);
    printf("\nSayinin yeni degeri: %d\n",sayi);
    printf("Degisen sayinin adresi: %p\n",&sayi);
    getch();
    return 0;
}
void Kup(int *psayi)
{
    *psayi=(*psayi)*(*psayi)*(*psayi);
}
```



```
Sayinin esas degeri: 5
Sayinin adresi: 000000000062FE1C

Sayinin yeni degeri: 125
Degisen sayinin adresi: 000000000062FE1C
-----
```

Adres değişmedi, adresteki değer değişti!!!

Örnek 2: Ana programdan girilen iki tamsayıyı bir fonksiyonda yer değiştiren (swap işlemi) C programını adres ile çağırma yöntemiyle yazınız.

```
a degerini giriniz:
78
a degerinin adresi: 000000000062FE1C
b degerini giriniz:
20
b degerinin adresi: 000000000062FE18

a degiskeninin yeni degeri: 20
a degiskeninin adresi: 000000000062FE1C
b degiskeninin yeni degeri: 78
b degiskeninin adresi: 000000000062FE18
-----
```

```
#include <stdio.h>
```

```
#include <conio.h>
```

```
void yerdegistir(int *pa, int *pb);
```

```
int main()
```

```
{
```

```
    int a,b;
```

```
    printf("a degerini giriniz: \n");
```

```
    scanf("%d",&a);
```

```
    printf("a degerinin adresi: %p\n",&a);
```

```
    printf("b degerini giriniz: \n", b);
```

```
    scanf("%d",&b);
```

```
    printf("b degerinin adresi: %p\n",&b);
```

```
    yerdegistir(&a,&b);
```

```
    printf("\n\n");
```

```
    printf("a degiskeninin yeni degeri: %d\n",a);
```

```
    printf("a degiskeninin adresi: %p\n",&a);
```

```
    printf("b degiskeninin yeni degeri: %d\n",b);
```

```
    printf("b degiskeninin adresi: %p\n",&b);
```

```
    getch();
```

```
    return 0;
```

```
}
```



```
void yerdegistir(int *pa, int *pb)
```

```
{    int gecici;
```

```
    gecici=*pa;
```

```
    *pa=*pb;
```

```
    *pb=gecici;
```

```
}
```

```
a degerini giriniz:
```

```
78
```

```
a degerinin adresi: 000000000062FE1C
```

```
b degerini giriniz:
```

```
20
```

```
b degerinin adresi: 000000000062FE18
```

```
a degiskeninin yeni degeri: 20
```

```
a degiskeninin adresi: 000000000062FE1C
```

```
b degiskeninin yeni degeri: 78
```

```
b degiskeninin adresi: 000000000062FE18
```

```
-----
```



Örnek 3: Ana programdan girilen string bir ifadenin tersini bulan C programını adres ile çağırma yöntemiyle yazınız.

```
String ifadeyi giriniz>> denizli pamukkale
String ifadenin tersi>> elakkumap ilzined
-----
```

```
#include <stdio.h>
#include <conio.h>
void ters(char *str);
```

```
int main()
{
    char str[100];
    printf("String ifadeyi giriniz>> ");
    gets(str);
    ters(str);
    getch();
    return 0;
}
```



```
void ters(char *str)
{
    int i,top=0 ;
    for(i=0;str[i]!='\0';i++)
        top+=1;
    printf("String ifadenin tersi>> ");
    for(int i=top-1;0<=i;i--)
        printf("%c",str[i]);
}
```



Fonksiyon Geri Dönüş Değeri Olan İşaretçiler

- Fonksiyonların geri dönüş değeri bir işaretçi olabilir.
- Bu durumda, fonksiyon bir değer değil adres döndürecek demektir.
- Fonksiyonun prototipi aşağıdaki şekildedir:

```
int *top(int a,int b)
```

Örnek: Ana programdan girilen iki tamsayının toplamını top fonksiyonunu işaretçi olarak tanımlandığı bir C programı yazınız.

```
#include <stdio.h>
#include <conio.h>
int *top(int x, int y);
int main()
{
    int a,b, *ptop;
    printf("a degerini giriniz: ");
    scanf("%d",&a);
    printf("b degerini giriniz: ");
    scanf("%d",&b);
    ptop=top(a,b);
    printf("\n");
    printf("iki sayinin toplami= %d\n",*ptop);
    printf("ptop isaretcisi degiskeninin sakladigi adres=
%p\n",ptop);
    getch();
    return 0;
}
```





```
int *top(int x, int y)
{
    int sonuc,*psonuc;
    sonuc=x+y;
    psonuc=&sonuc;
    printf("psonuc isaretci degiskeninin sakladigi adres=
%p\n",psonuc);
    return psonuc;
}
```

```
a degerini giriniz: 20
b degerini giriniz: 35
psonuc isaretci degiskeninin sakladigi adres= 000000000062FDD4
iki sayinin toplami= 55
ptop isaretci degiskeninin sakladigi adres= 000000000062FDD4
-----
```

Dinamik Bellek Yönetimi

Adresleme Kavramı

- Bilgisayarın ana belleği (**RAM**) sıralı kaydetme gözlerinden oluşmaktadır.
- Buradaki her bir göze adres atanmaktadır.
- Adreslerin değerleri sıfır ve belleğin sahip olduğu üst değere bağlı olarak değişmektedir.

Örnek:

1GB bir bellek,

- $1024 * 1024 * 1024 = 1.073.741.824$ adet gözden oluşur. (byte)

1TB bir bellek,

- $1024 * 1024 * 1024 * 1024 = 1.099.511.627.776$ adet gözden oluşur. (byte)

- Bir programlama dilinde tanımlanan bir değişkene değer atandığında, değişkende aşağıdaki 4 temel özellik bulunur.
- Değişkenlerin bellekte kapladığı alanlar sizeof komutu ile bulunabilir.

--- değişkenin adı

--- değişkenin tipi

--- değişkenin değeri

--- değişkenin bellekteki adresi

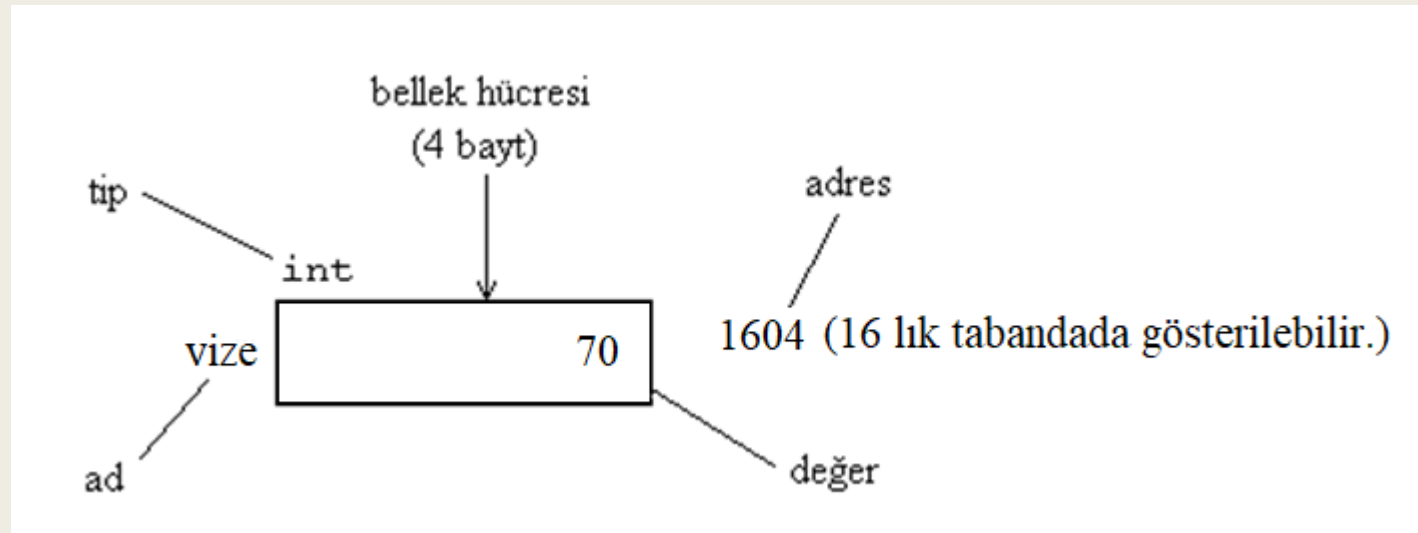
Örnek:

```
int vize = 70;
```

--- vize değişkeni için, bellekte int tipinde (4 bayt büyüklüğünde) bir hücre ayrılır ve o hücreye 70 sayısı ikilik (binary) sayı sistemindeki karşılığı olan 4 baytlık (32 bitlik) karşılığı aşağıdaki gibi yazılır.

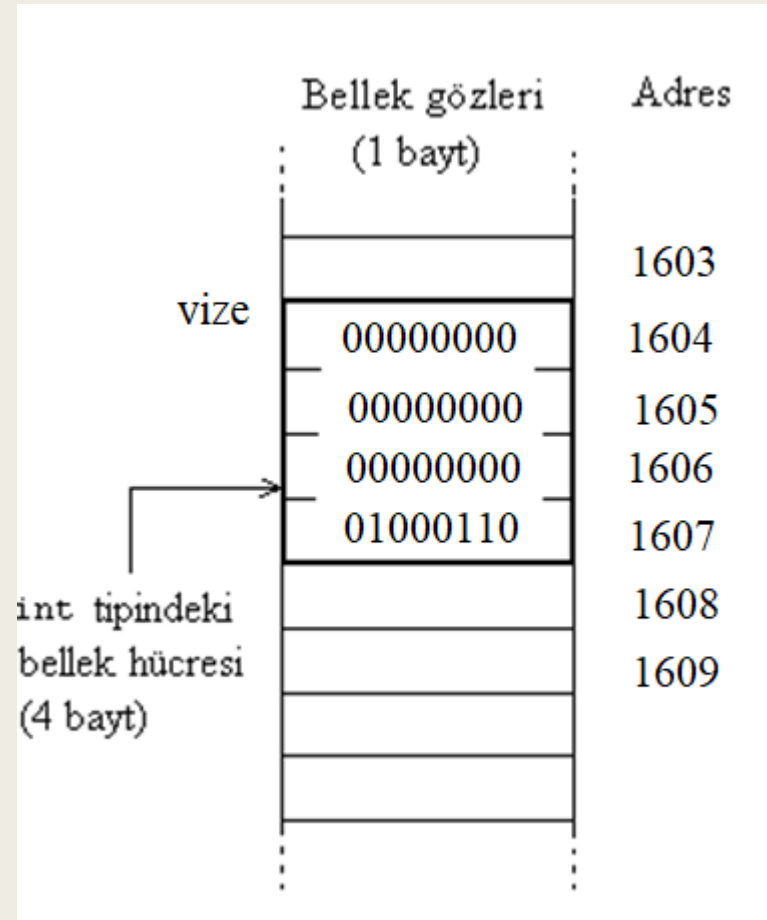
00000000 00000000 00000000 01000110

(70 sayısının ikilik tabandaki gösterimi)



--- Yukarıdaki şekilde, vize değişkeni, program çalıştığı sürece, bellekte 1604. - 1607. numaralı gözler arasındaki 4 baytlık hücreyi kullanmaktadır.

vize değişkeninin bellekteki gerçek konumu ve ikilik düzendeki içeriği aşağıdaki gibidir:



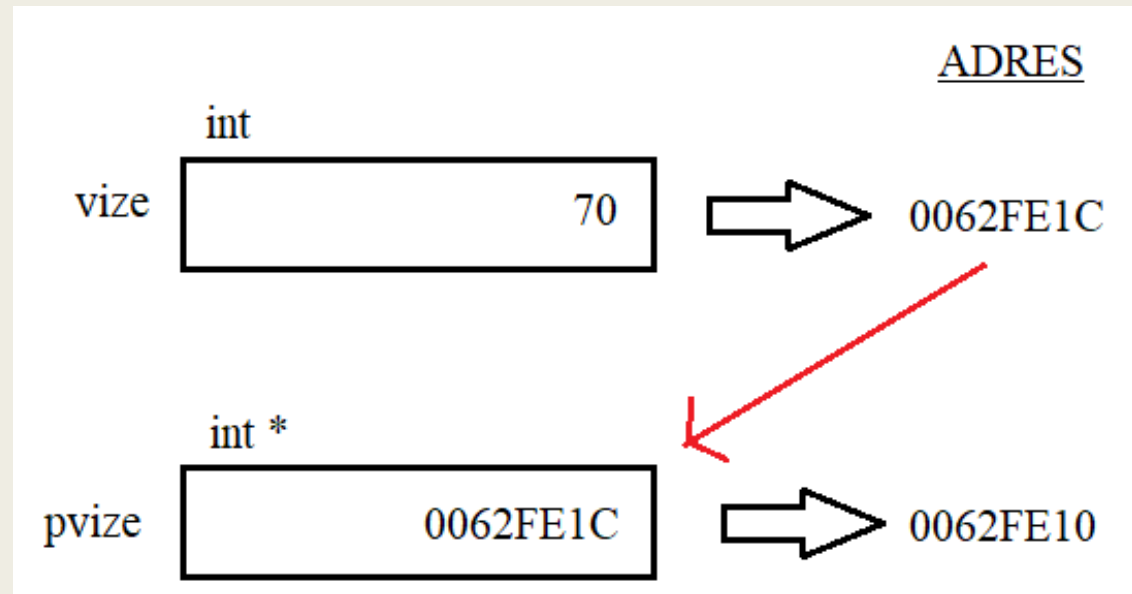
İşaretçiler

- Bir işaretçi değişkene, bir değişkenin adresini atamak için & (adres) operatörü kullanılır.
- Genel bir kural olmamakla beraber işaretçi değişkenler 'p' harfi ile başlar.
- Aşağıdaki örnekte, pvize işaretçi değişkeni vize değişkeninin saklandığı adresi tutmaktadır.

```
int *pvize, vize = 70;
```

.
. .
. .

```
pvize = &vize;
```



Dinamik Bellek Yönetimine Giriş

- Bir programcının geliştirdiği yazılım, sisteminin kaynaklarını en verimli şekilde kullanmayı amaçlamalıdır.
- En önemli sistem kaynaklarından biri istemci / sunucu tarafından kullanılan Bellektir. Disk Türü, Disk Alanı Boyutu, Toplam Sunucu Sayısı, vb. önemlidir.
- Sistem kaynakları sınırsız (sonsuz) değildir, bunlar en verimli şekilde kullanılmalıdır.
- Bellek yönetimi iyi yapılmazsa daima yeni bir belleğe ihtiyaç duyulur.
- Daha fazla bellek ihtiyacının olması maddi açıdan ve programın hızı açısından sorunlar yaratabilir.

Statik Dizi ve Dinamik Dizi

Statik Dizi

- Bir C programında, **dizilerin boyutu** program başında belirtildiğinde, derleyici tanımlanan dizi için gereken bellek alanını program sonlanıncaya kadar saklı tutmaktadır. Saklı tutulan bu alan başka bir amaç için kullanılamaz.
- Bu türdeki diziler statik dizi olarak adlandırılır ve Statik dizilerin boyutu programın çalışması esnasında değiştirilemez.

`int A[10]` → 10 eleman için saklı tutulan bellek alanı program sonlanıncaya kadar başka bir amaç için kullanılamaz!

Dinamik Dizi

- Bir Dizin boyutu bazı yöntemler (komutlar) yardımıyla programın çalışması esnasında değiştirilebilir. Bu tür dizilere Dinamik Diziler denir.
- Dinamik diziler için gereken bellek alanı, derleyici tarafından işletim sisteminden istenir, kullanılır ve gerekiyorsa bu alan boşaltılabilir.
- Örneğin, belirsiz sayıda yapılması gerekli bir işlem yapıyorsak Dinamik Dizi kullanmayı tercih ederiz. Statik Dizilerde bellek alanı başta tanımlandığından kullanılmaya ihtiyaç olmayan bellek alanı da boşuna işgal etmiş oluruz.

Dinamik Bellek Fonksiyonları

1-) `void *malloc(size_t eleman_sayısı)` fonksiyonu:

Bellekte her biri `size_t` tipinde olan `eleman_sayısı` kadar yer (bellek bloğu) ayırır.

Bu yer verilmezse `NULL` gönderir.

2-) `void *calloc(size_t eleman_sayısı, size_t nbayt)` fonksiyonu:

Bellekte her biri `nbayt` kadar yer işgal edecek `eleman_sayısı` kadar boş yer ayırır ve bütün bitleri sıfırlar. Bu yer ayrılamazsa geriye `NULL` gönderir.

Dinamik Bellek Fonksiyonları

3-) `void *realloc(void *ptr, size_t nbayt)` fonksiyonu:

`ptr` işaretçisi ile gösterilen bellek bloğunu, `nbayt` kadar büyüterek veya küçülterek değiştirir. Bu iş gerçekleşmezse geriye `NULL` gönderir.

4-) `void free(void *ptr)` fonksiyonu:

Daha önceden ayrılan adresi `ptr`' de saklanan bellek alanının boşaltır (serbest bırakır).

Fonksiyonlar `#include<stdlib.h>` ile kullanılır.

Örnek: Klavyeden girilen bir n değeri için, girilen n adet sayıyı A isimli bir diziye atayan ve dizinin elemanlarının toplamını bulan bir C programını Dinamik dizi kullanarak yazınız.

```
n degerini giriniz: 900000000000000000
Yetersiz bellek alanı istenmiştir...

-----
Process exited after 5.004 seconds with return value 1
```

```
n degerini giriniz: 4
1. elemani giriniz: 20
2. elemani giriniz: 35
3. elemani giriniz: 78
4. elemani giriniz: 90
Dizinin 1. elemani= 20
Dizinin 2. elemani= 35
Dizinin 3. elemani= 78
Dizinin 4. elemani= 90
Dizinin elemanlarının toplamı= 223

-----
```

C kodu:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

int main()
{
    int *A; // dinamik dizi tanımlanması
    int i,top=0,n;
    printf("n degerini giriniz: ");
    scanf("%d",&n);

    A = (int *) malloc( sizeof(int)*n );
    // n adet eleman için bellekten yer isteniyor.
    if( A == NULL )
    {printf("Yetersiz bellekalani istenmistir...\n"), exit(1);}

    for (i=0;i<n;i++)
    { printf("%d. elemani giriniz: ",i+1);
      scanf("%d",&A[i]);
    }
}
```



```
for (i=0;i<n;i++)
{ printf("Dizinin %d. elemani= %d\n",i+1,A[i]);}

for (i=0;i<n;i++)
{ top=top+A[i];}

printf("Dizinin elemanlarinin toplami= %d\n",top);

free(A); // istenilen alan serbest bırakılıyor.
getch();
return 0;
}
```



malloc() fonksiyonu ile calloc() fonksiyonun farkı

--- malloc() fonksiyonu istenilen eleman sayısı için gerekli olan bellek alanını ayırır, bu dizinin elemanlarına herhangi bir değer ataması yapmaz.

--- calloc() fonksiyonu istenilen eleman sayısı için gerekli olan bellek alanını ayırır, fakat bu ayrılan alandaki dizinin elemanlarına başlangıç değeri olarak sıfır değerini verir.

```

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
int main()
{
    int *A,*B;
    int i,n;
    printf("n degerini giriniz: ");
    scanf("%d",&n);
    printf("\n\n");
    printf("malloc fonksiyonu ile: \n");
    A = (int *) malloc( sizeof(int)*n );
    for (i=0;i<n;i++)
    { printf("A dizinin %d. elemani= %d\n",i+1,A[i]);}
    printf("\n\n");
    printf("calloc fonksiyonu ile: \n");
    B = (int *) calloc( n, sizeof (int) );
    for (i=0;i<n;i++)
    { printf("B dizinin %d. elemani= %d\n",i+1,B[i]);}

    free(A);
    free(B);
    getch();
    return 0;
}

```

```
n degerini giriniz: 8
```

```
malloc fonksiyonu ile:
```

```

A dizinin 1. elemani= 9966560
A dizinin 2. elemani= 0
A dizinin 3. elemani= 9961808
A dizinin 4. elemani= 0
A dizinin 5. elemani= 0
A dizinin 6. elemani= 0
A dizinin 7. elemani= 0
A dizinin 8. elemani= 0

```

```
calloc fonksiyonu ile:
```

```

B dizinin 1. elemani= 0
B dizinin 2. elemani= 0
B dizinin 3. elemani= 0
B dizinin 4. elemani= 0
B dizinin 5. elemani= 0
B dizinin 6. elemani= 0
B dizinin 7. elemani= 0
B dizinin 8. elemani= 0

```

```
-----
```

```
Process exited after 3.908 seconds with return value 0
```


Örnek: Bir sınıftaki 15 kişi, belirsiz sayıda resimden en güzelini seçmek için oylama yapacaktır. En çok oyu alan resmi bulan ve sonucu ekrana yazdıran C programını dinamik bellek kullanımı ile yapınız.

```
Yarisacak resim edetini giriniz: 7
1. ogrencinin en begendi resim degerini giriniz: 2
2. ogrencinin en begendi resim degerini giriniz: 3
3. ogrencinin en begendi resim degerini giriniz: 4
4. ogrencinin en begendi resim degerini giriniz: 3
5. ogrencinin en begendi resim degerini giriniz: 3
6. ogrencinin en begendi resim degerini giriniz: 4
7. ogrencinin en begendi resim degerini giriniz: 6
8. ogrencinin en begendi resim degerini giriniz: 7
9. ogrencinin en begendi resim degerini giriniz: 6
10. ogrencinin en begendi resim degerini giriniz: 7
11. ogrencinin en begendi resim degerini giriniz: 1
12. ogrencinin en begendi resim degerini giriniz: 3
13. ogrencinin en begendi resim degerini giriniz: 3
14. ogrencinin en begendi resim degerini giriniz: 4
15. ogrencinin en begendi resim degerini giriniz: 3
1. resmin aldigı oy adeti: 1
2. resmin aldigı oy adeti: 1
3. resmin aldigı oy adeti: 6
4. resmin aldigı oy adeti: 3
5. resmin aldigı oy adeti: 0
6. resmin aldigı oy adeti: 2
7. resmin aldigı oy adeti: 2
Yarismayi 3 numarali resim kazanmistir...
-----
Process exited after 17.31 seconds with return value 0
```

C kodu:



```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
```

```
int main()
{
    int *A;
    int i,n,x,enb,sakla;
    printf("Yarisacak resim adetini giriniz: ");
    scanf("%d",&n);
```

```
A = (int *) calloc( n, sizeof (int) );
for (i=1;i<=15;i++)
{
    printf("%d. ogrencinin en begendi resim
degerini giriniz: ",i);
    scanf("%d",&x);
    A[x-1]=A[x-1]+1;
}
```



```
for (i=0;i<n;i++)
{ printf("%d. resmin aldigi oy adeti: %d\n",i+1,A[i]);}
```

```
for (i=0;i<n;i++)
{ if (i==0) { enb=A[i]; sakla=i+1;}
  if (A[i]>enb) { enb=A[i]; sakla=i+1;}
}
printf("Yarismayi %d numarali resim
kazanmistir...\n",sakla);
```

```
free(A);
getch();
return 0;
}
```

Örnek: Bir sınıfta 10 kişi vardır. malloc() fonksiyonunu kullanarak öğrencilerin vize notlarını saklayan bir A dizisi tanımlayınız. Daha sonra sınıfa yeni katılan öğrencilerin notlarını da A isimli diziye atayınız. (Yeni eklenen öğrenciler için dizinin boyutu realloc() fonksiyonu ile artırılmalıdır.)

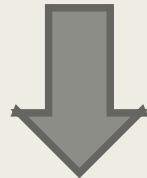
```
1. ogrencinin vize notunu giriniz: 33
2. ogrencinin vize notunu giriniz: 45
3. ogrencinin vize notunu giriniz: 78
4. ogrencinin vize notunu giriniz: 90
5. ogrencinin vize notunu giriniz: 43
6. ogrencinin vize notunu giriniz: 23
7. ogrencinin vize notunu giriniz: 67
8. ogrencinin vize notunu giriniz: 87
9. ogrencinin vize notunu giriniz: 45
10. ogrencinin vize notunu giriniz: 66
Sinifa yeni katilan ogrenci sayisini giriniz: 3
11. ogrencinin vize notunu giriniz: 55
12. ogrencinin vize notunu giriniz: 15
13. ogrencinin vize notunu giriniz: 68
Tum sinifin notlar2:
-----
1. ogrencinin vize notu= 33
2. ogrencinin vize notu= 45
3. ogrencinin vize notu= 78
4. ogrencinin vize notu= 90
5. ogrencinin vize notu= 43
6. ogrencinin vize notu= 23
7. ogrencinin vize notu= 67
8. ogrencinin vize notu= 87
9. ogrencinin vize notu= 45
10. ogrencinin vize notu= 66
11. ogrencinin vize notu= 55
12. ogrencinin vize notu= 15
13. ogrencinin vize notu= 68
```

C kodu:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
int main()
{
    int *A;
    int i,n,x;

    A = (int *) malloc( sizeof(int)*10);
    for (i=0;i<10;i++)
    { printf("%d. ogrencinin vize notunu giriniz: ",i+1);
      scanf("%d",&A[i]);
    }

    printf("Sinifa yeni katilan ogrenci sayisini giriniz: ");
    scanf("%d",&n);
```



```
A = (int *) realloc( A, 10+n);
// Dizinin boyutu arttırılıyor...
```

```
for (i=10;i<10+n;i++)
{ printf("%d. ogrencinin vize notunu giriniz: ",i+1);
  scanf("%d",&A[i]);
}
printf("Tum sinifin notlari:\n");
printf("-----\n");
for (i=0;i<10+n;i++)
{ printf("%d. ogrencinin vize notu= %d\n",i+1,A[i]);}

free(A);
getch();
return 0;
}
```

realloc() fonksiyonunun kullanımına bir örnek:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
```

```
int main()
{   char *A;
```

```
    A = (char *) malloc(sizeof(char) * 10);
    strcpy(A, "Algoritma");
```

```
    printf("Deger = %s, Adres = %p\n", A, A);
```

```
    A = (char *) realloc(A, 40);
```

```
    strcat(A, " ve Programlama Laboratuvari");
```

```
    printf("Deger = %s, Adres = %p\n", A, A);
```

```
    free(A);
```

```
    getch();
```

```
    return 0;
```

```
}
```

```
Deger = Algoritma, Adres = 00000000007613E0
Deger = Algoritma ve Programlama Laboratuvari, Adres = 00000000007613E0
-----
Process exited after 31.67 seconds with return value 0
```

Kaynaklar

- C: How to Program Third Edition Harvey M. Deitel ; Paul J. Deitel.
- C Programlama Dili Dr. Rıfat Çölkesen Papatya Yayıncılık.
- Problem Solving and Program Design in C, 7/E Jeri R. Hanly; Elliot B. Koffman.
- C Programlama dili; İbrahim Güney; Nobel Yayıncılık.
- Algoritma Geliştirme ve Programlamaya Giriş, Fahri Vatansever, Seçkin yayıncılık
- C Programlama Ders Notları, A. Kadir YALDIR, Pamukkale Üniversitesi ders notları.