

# **Bilgisayar Bilimlerine Giriş-II**

**-4-**

**BIL 1002**

**Dokuz Eylül Üniversitesi, Fen Fakültesi,  
Bilgisayar Bilimleri Bölümü**

# Bellek ve Adresleme

- ▶ Bilgisayarın ana belleği (RAM) sıralı kaydetme gözlerinden oluşmuştur.
- ▶ Her göze bir adres atanmıştır.
- ▶ Bu adreslerin değerleri 0 ila belleğin sahip olduğu üst değere bağlı olarak değişebilir.
- ▶ Örneğin 1GB bir bellek,  
 $1024 * 1024 * 1024 = 1073741824$  adet gözden oluşur.

# Bellek ve Adresleme

- ▶ Bir programlama dilinde, belli bir tipte değişken tanımlanıp ve bir değer atandığında, o değişkene dört temel özellik eşlik eder:
  - ▶ *değişkenin adı*
  - ▶ *değişkenin tipi*
  - ▶ *değişkenin sahip olduğu değer (içerik)*
  - ▶ *değişkenin bellekteki adresi*

# Örnek: Bellek ve Adresleme

```
int yas = 25;  
float boy = 1.72;  
float kilo = 65.7;  
char cins = 'B';
```

yas	25	4 bayt	5400
Boy	1.72	4 bayt	5404
kilo	65.7	4 bayt	5408
cins	'B'	1 bayt	5412
			5413

Bellek Adresleri

# Örnek: Bellek ve Adresleme

- ▶ Atama deyimlerine göre hücrelerin adreslere yerleşimi otomatik olarak gerçekleşir.
  - ▶ yas değişkeni 5400, 5401, 5402, 5403 adreslerini kapsadığından
  - ▶ boy değişkeni 5404 adresinden başlar ve sırasıyla 5405, 5406 ve 5407 adreslerini işgal eder.
  - ▶ kilo değişkeni bu nedenle 5408 adresinden başlar ve 5409, 5410, 5411 adreslerini kapsar.
  - ▶ cins değişkeni ise 5412 adresini tutar.

# Bellek ve Adresleme

## ► Örnek:

```
int tam = 33;
```

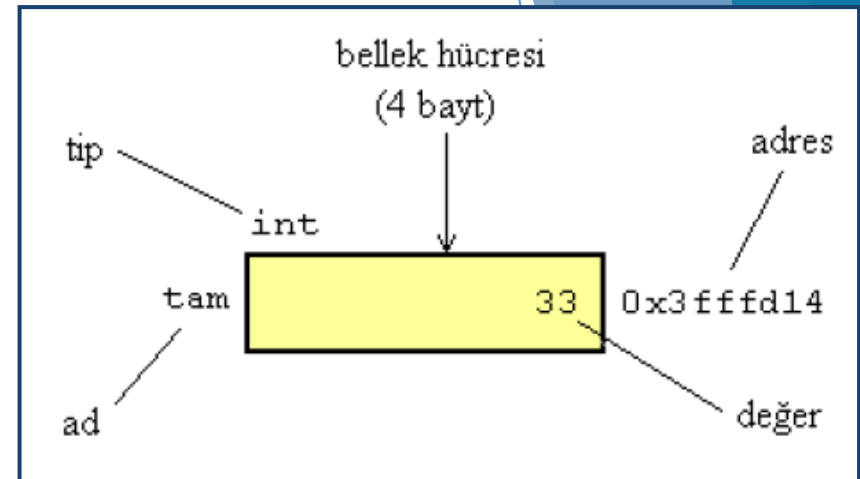
- Bu değişken için, int tipinde bellekte (*genellikle herbiri 1 bayt olan 4 bayt büyüklüğünde*) bir hücre ayrılır ve o hücreye 33 sayısı ikilik (binary) sayı sistemindeki karşılığı olan 4 baytlık (32 bitlik) karşılığı aşağıdaki gibi yazılır.

00000000 00000000 00000000 00100001

# Bellek ve Adresleme

## ► Örnek:

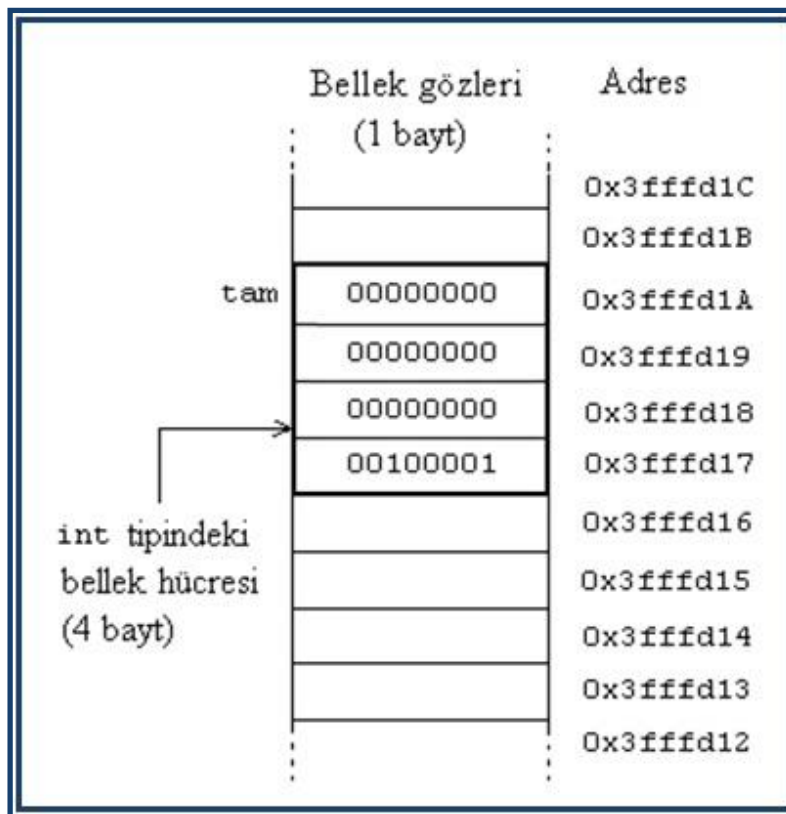
```
int tam = 33;
```



- Bellek adresleri genellikle onaltılık (hexadecimal) sayı sisteminde ifade edilir.
- 0x3fffd14 sayısı onluk (decimal) sayı sisteminde 67108116 sayısına karşılık gelir. Bunun anlamı, **tam** değişkeni, program çalıştığı sürece, bellekte **67108116 - 67108120** numaralı gözler arasındaki 4 baytlık hücreyi işgal edecek olmasıdır.

# Bellek ve Adresleme

- **tam** adlı değişkenin bellekteki gerçek konumu ve ikilik düzendeki içeriği aşağıdaki gibidir:





# Bellek ve Adresleme

- ▶ Değişkenin saklı olduğu adres, **&** karakteri ile tanımlı adres operatörü ile öğrenilebilir.
- ▶ Bu operatör bir değişkenin önüne konursa, o değişkenin içeriği ile değil adresi ile ilgileniliyor anlamına gelir.

# Örnek: Bellek ve Adresleme

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

int main()
{
    int tam = 33;
    printf("icerik: %d \n", tam);
    printf("adres: %p \n", &tam);
    return 0;
}
```

Değişkenin içeriği

Değişkenin adresi

```
icerik:33
adres:0022FEAC
```

# İşaretçi (Pointer) Kavramı

- ▶ C dili, bir değişkenin adresinin bir başka değişkende saklanmasına izin verir. Bu değişkene **işaretçi** denir.
- ▶ İşaretçi denmesinin sebebi ilgili değişkenin adresini işaret etmesinden yani göstermesinden kaynaklanır.
- ▶ Diğer bir deyişle işaretçi, bir değişkenin adresini içeren başka bir değişkendir.

# İşaretçi (Pointer) Kavramı

- ▶ Bir işaretçi, diğer değişkenler gibi, sayısal bir değişkendir.
- ▶ Bu sebeple kullanılmadan önce program içinde bildirilmelidir. İşaretçi tipindeki değişkenler aşağıdaki gibi tanımlanır:

**tip\_adı \*isaretci\_adı;**

- ▶ Burada **tip\_adı** herhangi bir C veri türü olabilir. Değişkenin önündeki **\*** karakteri **yönlendirme (indirection)** operatörü olarak adlandırılır ve bu değişkenin veri değil bir adres bilgisi tutacağını işaret eder.

# İşaretçi (Pointer) Kavramı

```
char *kr;
```

```
int *x;
```

```
float *deger, sonuc;
```

```
/* tek bir karakter için */
```

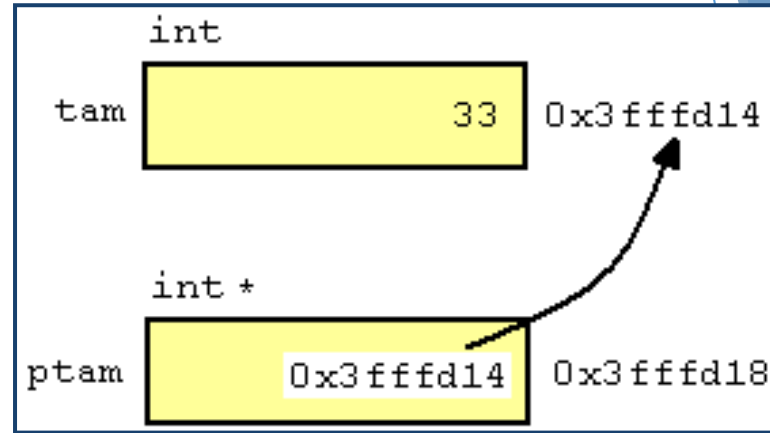
```
/* bir tamsayı için */
```

```
/* deger işaretçi tipinde,  
sonuc sıradan bir gerçel  
değişken */
```

Yukarıda bildirilen işaretçilerden; **kr** bir karakterin, **x** bir tamsayının ve **deger** bir gerçel sayının bellekte saklı olduğu yerlerin adreslerini tutar.

# İşaretçi (Pointer) Kavramı

```
int *ptam, tam = 33;  
.  
.  
.  
ptam = &tam;
```



- Bir işaretçiye, bir değişkenin adresini atamak için **& (adres)** operatörünü kullanırız.
- **ptam** işaretçisi **tam** değişkeninin saklandığı **adres**i tutacaktır.

# Değer/Adres

	DEĞER	ADRES
p (pointer)	*p	p
t (tamsayı)	t	&t

- Bir işaretçi değişkenin adının önüne \* operatörü konulursa, bu işaretçinin tuttuğu adres ile değil, işaret ettiği yerdeki **veri** ile ilgileniyoruz demektir.

# Örnek: İşaretçi Gösterimi

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

```
int main(void)
```

```
{
```

```
    int tam=33;
```

```
    int *ptam;
```

```
    ptam=&tam;
```

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

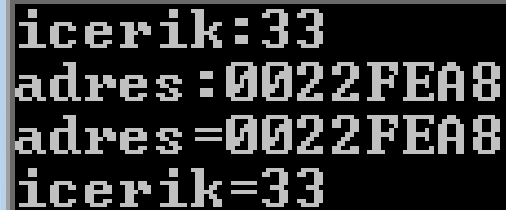
```
    printf("adres: %p \n",&tam);
```

```
    printf("adres: %p \n",ptam);
```

```
    printf("icerik=%d",*ptam);
```

```
    return 0;
```

```
}
```



```
icerik:33  
adres:0022FEA8  
adres=0022FEA8  
icerik=33
```



# Örnek: İşaretçi Gösterimi ve Değer Değiştirme

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

```
int main()
```

```
{
```

```
    int tam=33;
```

```
    int *ptam;
```

```
    ptam=&tam;
```

```
    printf("&tam=%p\n",&tam);
```

```
    printf("ptam=%p\n",ptam);
```

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

```
    printf("&tam=%d\n",tam);
```

```
    printf("*ptam=%d\n",*ptam);
```

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

**\*ptam** ve **tam**, **tam** adlı değişkenin içeriği ile ilgilidir.

**ptam** ve **&tam**, **tam** adlı değişkenin adresi ile ilgilidir.

```
*ptam=44;
```

```
printf("tam=%d\n",tam);
```

```
printf("*ptam=%d\n",*ptam);
```

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

```
printf("&tam=%p\n",&tam);
```

```
printf("ptam=%p\n",ptam);
```

```
return 0;
```

```
}
```

```
&tam=0023FEA8  
ptam=0023FEA8
```

```
&tam=33  
*ptam=33
```

```
tam=44  
*ptam=44
```

```
17 &tam=0023FEA8  
ptam=0023FEA8
```

# İşaretçi (Pointer) Dönüşümleri

- İşaretçilerin arasındaki tip dönüşümleri sadece adreste tutulan değişkenin değerini değiştirmez. Ancak derleyicinin aynı adrese yeni tipteki bir veri gibi davranmasını sağlar.

**(tip\_adı \*) isaretc\_i\_adı;**

```
#include <stdio.h>

int main(void){
    int tam = 33;
    printf("icerik: %d\n", tam);
    printf("adres: %p\n", &tam);
    char *ctam = (char *)&tam;
    printf("      adres\ticerik\n");
    printf("%p\t%d\n",&ctam[0],ctam[0]);
    printf("%p\t%d\n",&ctam[1],ctam[1]);
    printf("%p\t%d\n",&ctam[2],ctam[2]);
    printf("%p\t%d\n",&ctam[3],ctam[3]);
    return 0;
}
```

```
icerik: 21
adres: 0x7ffe97e0cbcc
      adres      icerik
0x7ffe97e0cbcc  33
0x7ffe97e0cbcd   0
0x7ffe97e0cbce   0
0x7ffe97e0cbcf   0
```

# İşaretçi Aritmetiği

- ▶ İşaretçiler kullanılırken, bazen işaretçinin gösterdiği adres taban alınıp, o adresten **önceki** veya **sonraki** adreslere erişilmesi istenebilir.
- ▶ Bu durum, işaretçiler üzerinde, aritmetik işlemcilerin kullanılmasını gerektirir.
- ▶ İşaretçiler üzerinde yalnızca
  - ▶ toplama (+),
  - ▶ çıkarma (-),
  - ▶ bir arttırma (++) ,
  - ▶ bir eksiltme (--)

operatörleri işlemleri yapılabilir.

# Örnek: İşaretçi Aritmetiği

- ▶ Aşağıdaki gibi üç tane gösterici bildirilmiş olsun:

**char** \*kar; → 10000 (0x2710)

**int** \*tam; → 20000 (0x4e20)

**double** \*ger; → 30000 (0x7530)

- ▶ Buna göre aşağıdaki atama işlemlerinin sonucu ne olmalıdır?
  - ▶ kar++;
  - ▶ tam++;
  - ▶ ger++;

# Örnek: İşaretçi Aritmetiği

- ▶ Bir işaretçiye ekleme yapıldığında, o anda tuttuğu adres ile eklenen sayı doğrudan toplanmaz.
- ▶ Böyle olsaydı, bu atamaların sonuçları sırasıyla **10001**, **20001** ve **30001** olurdu.
- ▶ Gerçekte, işaretçiye bir eklemek, işaretçinin gösterdiği yerdeki veriden hemen sonraki verinin adresini hesaplamaktır
- ▶ Buna göre atama işlemlerinin sonucu:
  - ▶ `kar++;` → 10001 (0x2711)
  - ▶ `tam++;` → 20004 (0x4e24)
  - ▶ `ger++;` → 30008 (0x7538)

# Örnek: İşaretçi Aritmetiği

- ▶ Genel olarak, bir işaretçiye  $n$  sayısını eklemek veya çıkarmak, *bellekte gösterdiği veriden sonra veya önce gelen  $n$ . elemanın adresini hesaplamaktır.*
- ▶ Buna göre aşağıdaki atamalar şöyle yorumlanır.
  - ▶ `kar++;`                      `// kar = kar + sizeof(char)`
  - ▶ `tam = tam + 5;`           `// tam = tam + 5*sizeof(int)`
  - ▶ `ger = ger - 3;`           `//ger = ger - 3*sizeof(double)`

# Fonksiyonları Referansa göre Çağırma

- ▶ Bu tip çağırmada gönderilecek *verilerin değerleri yerine*, değerlerin bulunduğu bellek gözünün adresi gönderilir. Yani, *fonksiyonlar arasındaki veri paylaşımı adresler aracılığı ile* yapılır.
- ▶ Bu durumda argüman ile buna karşılık gelen fonksiyon parametresi, aynı bellek alanını kullanır. Bu çağırma tipinde geriye 1'den fazla değer gönderilebilir ve fonksiyon kendisine gelen adresten gerçek değeri alıp kullanır.
- ▶ Çağırana değer göndermek için return deyimi kesinlikle kullanılmaz. Bunun yerine bellek adresi kullanılır.

# Örnek:

```
#include <stdio.h>
#include <conio.h>
int Kup(int a)
{
    return a*a*a;
}
int main()
{
    int sayi=5;
    printf("Sayinin esas degeri: %d",sayi);
    sayi=Kup(sayi);
    printf("\nYeni deger: %d",sayi);
    getch();
    return 0;
}
```

*Değere göre Çağırma*

```
#include <stdio.h>
#include <conio.h>
void Kup(int *aPtr)
{
    *aPtr=(*aPtr)*(*aPtr)*(*aPtr);
}
int main()
{
    int sayi=5;
    printf("Sayinin esas degeri: %d",sayi);
    Kup(&sayi);
    printf("\nYeni degeri: %d",sayi);
    getch();
    return 0;
}
```

*Referansa göre Çağırma*



## Örnek: 2 sayının sırasını degistir isimli fonksiyonda değiştirerek main() fonksiyonunda yazdıran program.

```
#include<stdio.h>
void degistir (int *a, int *b)
{
    int g;
    g=*a;
    *a=*b;
    *b=g;
}
```

```
int main()
{
    int x,y;
    printf ("bir sayı giriniz\n");
    scanf("%d",&x);
    y=23;
    printf ("girdiginiz sayi x=%d ve y=%d\n",x,y);
    degistir (&x,&y);
    printf ("simdi x=%d ve y=%d",x,y);
}
```

- Referans yoluyla aktarım olmasaydı, iki değişkenin değerlerini fonksiyon kullanarak değiştiremezdik.
- Eğer yazdığınız fonksiyon *birden çok değer* döndürmek zorundaysa, referans yoluyla aktarım zorunlu hale geliyor.
- Çünkü daha önce işlediğimiz *return* ifadesiyle sadece **tek bir** değer döndürebiliriz.

# Örnek: Bölme işlemini yapıp, bölüm sonucunu ve kalanı söyleyen fonksiyon

```
#include<stdio.h>

int bolme_islemi( int bolunen, int bolen, int *kalan )
{
    *kalan = bolunen % bolen;
    return bolunen / bolen;
}

int main()
{
    int bolunen, bolen;
    int bolum, kalan;
    bolunen = 13;
    bolen = 4;
    bolum = bolme_islemi( bolunen, bolen, &kalan );
    printf( "Bolum: %d Kalan: %d\n", bolum, kalan );

    return 0;
}
```

- Bu durumda, ***bölünen*** ve ***bölen*** fonksiyona gidecek argümanlar olurken; ***kalan*** ve ***bölüm*** geriye dönmelidir.
- ***return*** ifadesi geriye tek bir değer vereceğinden, ikinci değeri alabilmek için referans yöntemi kullanmamız gerekir.

# Örnek: Bir dairenin alanını ve çevresini hesaplayan fonksiyon

```
#include<stdio.h>
#include<conio.h>
const float pi=3.14;
void daire(float r,float *al,float *cev)
{
    *al=pi*(r)*(r);
    *cev=pi*(r)*2;
}
int main()
{
    float yaricap,alan,cevre;
    printf("Yaricapi giriniz:");
    scanf("%f",&yaricap);
    daire(yaricap,&alan,&cevre);
    printf("\Alan=%f ve Cevre=%f",alan,cevre);
    getch();
    return 0;
}
```

# İşaretçi ve Diziler Arasındaki İlişki

- ▶ C dilinde işaretçi ve diziler arasında yakın bir ilişki vardır.
- ▶ Dizi adı, aslında, dizinin ilk elemanının adresinin tutan bir işaretçidir.
- ▶ Bir dizinin herhangi elemanına işaretçiyle de erişebilir.

Örneğin,

```
char tablo[40], *p, *q;  
biçiminde bildirim yapılmış olsun.
```

```
p = &tablo[0];    // dizinin ilk elemanının adresi p'ye atanıyor  
p = tablo;        // dizinin baslangic adresi p'ye atanıyor  
q = &tablo[9];    // 10. elemanin adresi q'ya atanıyor  
q = &tablo[39];   // son elemanin adresi q'ya atanıyor28
```

# İşaretçi ve Diziler Arasındaki İlişki

- İlk iki satırdaki atamalar aynı anlamdadır. Dizi adı bir gösterici olduğu için, doğrudan aynı tipteki bir göstericiye atanabilir. Ayrıca,  $i$  bir tamsayı olmak üzere,

`tablo[i];` ile `*(p+i);`

aynı anlamdadır.

- Bunun sebebi, `p` göstericisi tablo dizisinin başlangıç adresini tutmuş olmasıdır.  $p+i$  işlemi ile  $i+1$ . elemanın adresi, ve `*(p+i)` ile de bu adresteki değer hesaplanır.

## NOT

Bir dizinin,  $i$ . elemanına erişmek için `*(p+i)` işlemi yapılması zorunludur.

Yani

`*p+i;` // `p` nin gösterdiği değere (dizinin ilk elemanına)  $i$  sayısını ekle

`*(p+i);` // `p` nin gösterdiği adresten  $i$  blok ötedeki sayıyı hesapla

anlamındadır.

Çünkü, `*` operatörü `+` operatörüne göre işlem önceliğine sahiptir.

## Örnek: 7 elemanlı bir tamsayı dizisinin hafızada yerleşimini yazdıran program

```
#include <stdio.h>
int main(void)
{
    int val[7] = { 11, 22, 33, 44, 55, 66, 77 } ;

    for ( int i = 0 ; i <= 6 ; i++ ){
        printf("\nval[%d]: degeri %d ve adresi %p", i, val[i], &val[i]);
    }
    return 0;
}
```

```
val[0]: degeri 11 ve adresi 0023FE80
val[1]: degeri 22 ve adresi 0023FE84
val[2]: degeri 33 ve adresi 0023FE88
val[3]: degeri 44 ve adresi 0023FE8C
val[4]: degeri 55 ve adresi 0023FE90
val[5]: degeri 66 ve adresi 0023FE94
val[6]: degeri 77 ve adresi 0023FE98
```

```
#include <stdio.h>
int main()
{
    int b[4] = { 10, 20, 30, 40 };
    int *bPtr = b;
    int i, offset;
    printf("Dizi belirtecleri yontemi\n");
    for ( i = 0; i < 4; i++ )
        printf( "b[ %d ] = %d\n", i, b[i]);
```

```
d dizi asagidaki metdolarla yazilmistir:
Dizi belirtecleri yontemi
b[ 0 ] = 10
b[ 1 ] = 20
b[ 2 ] = 30
b[ 3 ] = 40

Gosterici/offset yonetmi,
gosterici dizinin ismiyken
*( b + 0 ) = 10
*( b + 1 ) = 20
*( b + 2 ) = 30
*( b + 3 ) = 40

Gosterici belirtec yontemi
bPtr[ 0 ] = 10
bPtr[ 1 ] = 20
bPtr[ 2 ] = 30
bPtr[ 3 ] = 40

Gosterici/offset yontemi
*( bPtr + 0 ) = 10
*( bPtr + 1 ) = 20
*( bPtr + 2 ) = 30
*( bPtr + 3 ) = 40
```

```
printf("\nGosterici/offset yontemi - gosterici dizinin ismiyken\n");
for (offset = 0; offset < 4; offset++)
    printf("( b + %d ) = %d\n", offset, *(b + offset));
```

```
printf("\nGosterici belirtec yontemi\n");
for (i = 0; i < 4; i++)
    printf( " bPtr[ %d ] = %d\n", i, bPtr[i]);
```

```
printf("\nGosterici/offset yontemi\n");
for (offset = 0; offset < 4; offset++)
    printf("( bPtr + %d ) = %d\n", offset, *(bPtr + offset));
return 0;
}
```

**Örnek:** Bir dizinin ilk elemanının başlangıç adresini kullanarak dizi elemanlarının toplamını bulan program

```
#include<stdio.h>
int dizi1(int a[],int n)
{
    int *p,toplam=0,sayac;
    p=&a[0];
    for(sayac=0;sayac<n;sayac++)
        toplam+=*(p+sayac);
    return toplam;
}
int main(void)
{
    int sayi[10]={1,2,3,4,5,6,7,8,9,10};
    int f;
    f=dizi1(sayi,10);
    printf("Sayilarin toplami=%d",f);
    return 0;
}
```



# Gösterici Dizileri

- ▶ Diziler göstericiler içerebilir.
- ▶ Örneğin; takım string dizisini ele alalım

Char

```
*takim[4]={ "kupa","karo","sinek","maca" };
```

- ▶ C'de string'ler, gerçekte ilk karakteri gösteren göstericilerdir.
- ▶ char\*- takım'ın her bir elemanı char gösteren bir göstericidir.
- ▶ string'ler aslında takım dizisinde tutulmazlar sadece bu string'leri gösteren göstericiler tutulur.

ÖRNEKLER

# Soru 1

► *sizeof* operatörü kullanarak sırası ile

- char
- short
- int
- long
- float
- double
- long double,
- int [20] ve bir int diziyi gösteren int\*'i için

gerekli olan byte sayısını ekrana yazdıran programı yazınız.

```
#include <stdio.h>
#include <conio.h>
int main( )
{
    char c;
    short s;
    int i;
    long l;
    float f;
    double d;
    long double ld;
    int dizi[ 20 ], *ptr = dizi;
```

```
printf( " sizeof c = %d"
"\tsizeof(char) = %d"
"\n sizeof s = %d"
"\tsizeof(short) = %d"
"\n sizeof i = %d"
"\tsizeof(int) = %d"
"\n sizeof l = %d"
"\tsizeof(long) = %d"
"\n sizeof f = %d"
"\tsizeof(float) = %d"
"\n sizeof d = %d"
"\tsizeof(double) = %d"
"\n sizeof ld = %d"
"\tsizeof(long double) = %d"
"\n sizeof dizi = %d"
"\n sizeof ptr = %d\n",
sizeof c, sizeof ( char ), sizeof s,
sizeof ( short ), sizeof i, sizeof ( int ),
sizeof l, sizeof ( long ), sizeof f,
sizeof ( float ), sizeof d, sizeof ( double ),
sizeof ld, sizeof ( long double ),
sizeof dizi, sizeof ptr );
}
```

```
sizeof c = 1      sizeof(char) = 1
sizeof s = 2      sizeof(short) = 2
sizeof i = 4      sizeof(int) = 4
sizeof l = 4      sizeof(long) = 4
sizeof f = 4      sizeof(float) = 4
sizeof d = 8      sizeof(double) = 8
sizeof ld = 12    sizeof(long double) = 12
sizeof dizi = 80
sizeof ptr = 4
```

## Soru 2

- ▶ **main()** fonksiyonunda verilen bir  $x$  değerini **f** isimli bir fonksiyona aktarıp,
  - ▶  $f(x)=3x-1$  değerini hesaplatan ve bu değeri **g** isimli bir fonksiyona gönderen,
  - ▶ **g** isimli fonksiyonda,  $g(x)=x^2+2x-3$  değerini hesaplatan
  - ▶ sonucu **main()** fonksiyonunda yazdıran

C programı yazınız.

*Not: program adres ile çağırmaya göre yazılacaktır.*

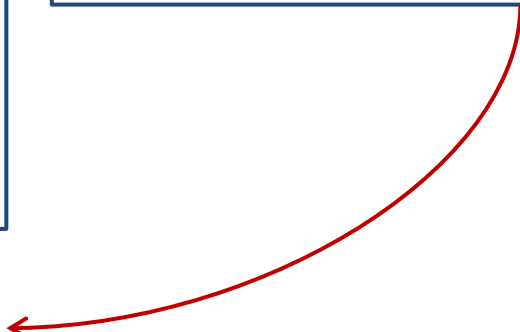
```
#include<stdio.h>
void g(int *t1, int *k1)
{
    int z;
    z=*t1;
    *k1=z*z+2*z-3;
}

void f(int *x1, int *y1)
{
    int c,t,k;
    c=*x1;
    t=3*c-1;
    g(&t,&k);
    *y1=k;
}
```

```
int main(void)
{
    int x, y;

    printf("Bir sayi giriniz:");
    scanf("%d",&x);
    f(&x,&y);

    printf("\nSonuc=%d",y);
    return 0;
}
```



# Soru 3

- ▶ 50 adet (a,b,c) değer grubu veriliyor.
  - ▶ Verilen her bir (a,b,c) grubunu *hesap* isimli bir fonksiyona aktaran ve burada  $y=a!/(bc)$  değerini hesaplatıp, main() fonksiyonuna geri gönderen
  - ▶ Her bir y değerini ve hangi (a,b,c)'den oluştuğunu *enbhesap* adlı fonksiyona gönderen ve bu fonksiyonda en büyük y değerini ve hangi (a,b,c)'lerden oluştuğunu bularak main() fonksiyonuna sonuçları gönderen bir program yazınız.

*Not: program adres ile çağırmaya göre yazılacaktır.*

```

int main(void)
{
    int a,b,c,i;
    float y;

    for(i=1;i<=5;i++){
        printf("\n3 sayi giriniz:");
        scanf("%d%d%d",&a,&b,&c);
        y=hesap(a,b,c);
        printf("\nFonksiyon degeri=%f",y);
        enbhsp(&y,&a,&b,&c);
    }
    printf("\nSonuc=%f, a=%d, b=%d, c=%d",y,a,b,c);
    return 0;
}

```

```

#include<stdio.h>
float hesap(int a1,int b1,int c1)
{
    int carp=1,j;
    float y1;

    for(j=1;j<=a1;j++)
        carp=carp*j;
    y1=(float)(carp/(b1*c1));
    return y1;
}

```

```

void enbhsp(float *y2,int *a2,int *b2,int *c2)
{
    static int a3,b3,c3;
    static float enb;
    static int k=0;
    k++;
    if(k==1) {enb=*y2;a3=*a2;b3=*b2;c3=*c2;}

    if(*y2>=enb)
    {enb=*y2;a3=*a2;b3=*b2;c3=*c2;}
    if(k==5) {*y2=enb;*a2=a3;*b2=b3;*c2=c3;}
}

```



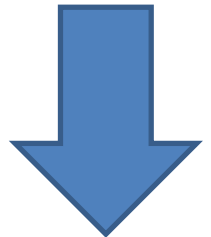
## Soru 4

- 1-54 arası 6 adet birbirinden farklı sayıyı rasgele olarak üretip bir diziye alan ve bunları sıralı olarak ekrana yazdıran bir program yazınız. Programda sıralama işlemi için argüman olarak gösterici alan void bir fonksiyon kullanılsın.

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

```
#include <time.h>
```

```
void secmeliSiralama(int *dizi, int boyut)
{
    int enkucuk, yedek;
    for (int i = 0; i < boyut-1; i++)
    {
        enkucuk = i;
        for (int j = i + 1; j < boyut; j++)
            if (dizi[j] < dizi[enkucuk])
                enkucuk = j;
        if(enkucuk!=i){
            yedek = dizi[i];
            dizi[i] = dizi[enkucuk];
            dizi[enkucuk] = yedek;
        }
    }
}
```



```

int main(void)
{
    srand(time(NULL));
    int c=1,i=0,a[6],kontrol=0,r;
    while(c==1){
        while(i<6){
            a[i]=1+rand()%54;
            for(int j=i-1;j>=0;j--){
                if(a[j]==a[i]){
                    a[i]=1+rand()%54;
                    j=i-1;
                }
            }
            i++;
        }
        secmeliSiralama(a,6);
        for(i=0;i<6;i++) printf("%d ",a[i]);
        printf("\nYeni sayi(Evet=1 Hayir=0):");
        scanf("%d",&c);
    }
    return 0;
}

```

```

4 15 20 26 30 39
Yeni sayilar turetmek istermisiniz?(Evet=1 Hayir=0):1
10 11 18 32 47 50
Yeni sayilar turetmek istermisiniz?(Evet=1 Hayir=0):1
10 16 24 34 47 51
Yeni sayilar turetmek istermisiniz?(Evet=1 Hayir=0):1
9 10 34 38 43 45
Yeni sayilar turetmek istermisiniz?(Evet=1 Hayir=0):1
7 20 27 35 44 50
Yeni sayilar turetmek istermisiniz?(Evet=1 Hayir=0):0

```

SON