BÖLÜM 4

FONKSİYONLAR

Fonksiyonlar programların etkinliğini, kolay yazılmasını, anlaşılmasını ve bellekte daha az yer kaplamasını sağlayan bloklardır.

Yazılan bir fonksiyon programın farklı yerlerinde defalarca çağrılarak çalıştırılabilir. Fonksiyonları bir programın bütününden bağımsız olarak yazmak yani programın global tanımlarını kendi iç bloğunda kullanmayan fonksiyonlar yazmak uygundur.

Bir C programı main() fonksiyonuyla çalışmaya başlar. Diğer fonksiyonlar ise main fonksiyonundan ya da çalışan başka bir fonksiyon içerisinden çağrılarak çalıştırılır.

Her C programı içinde main adı verilen bir fonksiyonun bulunması gerekir, çünkü program yürütülmeye başlandığında, program içinde bağlanmış bulunan bir başlangıç yordamı çalışmaya başlar, bu da kontrolü Main fonksiyonuna geçirir.

Eğer bağlayıcı (*linker*) main adı verilen bir fonksiyon bulamazsa yürütülebilir bir kod oluşturamayacaktır. main bir anahtar sözcük değildir, yani istenirse, main adında bir değişken de tanımlanabilir.

Fonksiyonların gücü, tekrarlanan kod yazımından kaçınmamızı sağlamalarında yatar.

Herhangi bir kod, fonksiyon şeklinde bir defa belirtilir, ona bir isim verilir. Daha sonra, bu kodu çalıştırma gereksinimi duyulduğunda, bu fonksiyon “çağrılır”.

Bu iyi, fakat yeterli değildir. Birçok durumda bu kod o anki gereksinimlere “uyarlanmalıdır”. Örneğin, bir diziyi sıraya sokan bir kod varsa, bu kod her çağrıldığında, değişik sayıda elemandan oluşan değişik bir dizi belirtecek şekilde hazırlanması daha iyi olur. Bu ise, *fonksiyon argümanları* kullanılarak yapılabilir.

Her fonksiyon diğerlerinden bağımsızdır. Yani bir fonksiyon içinde tanımlanmış değişkenler başka bir fonksiyonda kullanılamadığına göre, fonksiyonların birbirleriyle haberleşmelerini sağlayacak yöntemler geliştirilmiştir. Bir yöntem fonksiyonlara argüman geçirmektir. Başka bir yöntem global (*küresel) değişkenlerin* kullanılmasıdır.

Fonksiyon Tanımlama

Bir fonksiyonun genel yazım biçimi aşağıdaki gibidir.

```
dönüş_tipi  fonksiyon_ismi (parametre_bildirimleri)
                                {
                                tip tanımlamaları;
                                deyimler;
                                }
```

```
float deneme (int,float);
int en_kucuk(int,int);
void yaz (int,char);
void sonuc(void);
```

Dönüş_tipi: Fonksiyon tarafından döndürülen değerin tipini (int, double…gibi) belirtir. Bir fonksiyon için varsayılan dönüş tipi int'tir. Ancak, her durumda (yani int bile olsa) dönüş tipinin açıkça belirtilmesi önerilir. *Dönüş_tipi* olarak void kullanıldığında, fonksiyondan hiçbir değer döndürülmeyeceği belirtilmiş olur.

Fonksiyon_adı: Değişken adı tanımlama kuralları ile verilen herhangi bir isimdir.

Parametre_bildirimleri: Diğer bir fonksiyon tarafından çağrıldığında bu fonksiyona aktarılacak olan tipleri belirtilerek virgülle ayrılmış *biçimsel argüman* veya *parametre* listesidir. Aynı zamanda parametrelere, sadece fonksiyonun gövdesini oluşturan *blok*' un içinde anlamlı olan isimler verir. Hiç parametresi olmayan bir fonksiyon da olabilir. Bu durum, parantezler içine void yazılarak açıkça gösterilir.

```
void dene(void)
                {
                 printf (“\nbu bir fonksiyondur”);
                }


int dene(void)
                {
                 int a=10;
                 printf (“\nbu bir fonksiyondur”);
                 printf (“\n a=%d deneme fonksiyonunun bir değişkenidir.”,a);
                }


int mutlak(int n)
               {
                if (n >= 0) return n;
                else return -n;
                }
```

`return` deyimi, sonucun çağırana geri dönmesini sağlar. Return deyimi,
return ; veya **return ifade;**
şeklinde kullanılır.

`return` deyiminden sonra gelen deyimler yerine getirilmez ve fonksiyon hemen çağrıldığı yere "geri döner".

return; kullanımında akış geri döner, ancak yararlı hiç bir değer döndürülmez. Geri döndürülen değer tanımsız olduğu için, çağıran fonksiyon bunu kullanmaya çalışmamalıdır.

return ifade; şeklindeki kullanımda ise, ifadenin değeri fonksiyonun değeri olarak geri döndürülür. Eğer ifadenin değerinin tipi fonksiyonun dönüş tipiyle aynı değilse, otomatik tip dönüşümü yapılır.

"Dönüş tipi" `void` olan fonksiyonlar için return kullanımına gerek yoktur.
Küme parantezinin önünde bir `return` olduğu varsayılır; bundan dolayı belli bir değer döndürülmek istenmiyorsa `return` yazılmaz. Son deyim yerine getirildikten sonra, yararlı bir değer olmadan kontrol çağıran fonksiyona döndürülecektir.

Fonksiyon Çağrıları: Bir fonksiyonu çağırmak için fonksiyon ismi ve parantez içine alınmış *argüman*ların bir listesi belirtilir. Fonksiyonun argümanları olmasa dahi parantezler bulunmalıdır, böylece derleyici ismin bir değişkeni değil de bir fonksiyonu gösterdiğini anlayabilecektir.

Fonksiyon tarafından geri döndürülen değer kullanılabilir veya kullanılmayabilir. Örneğin,

```
mutlak(-10);
```

bir fonksiyon çağrısıdır. Bu noktada kontrol, çağıran fonksiyondan çağrılan fonksiyona aktarılır. -10 değeri mutlak fonksiyonunun n parametresine atanır. Çağrılan fonksiyon, ya bir **return** deyiminin yerine getirilmesi yada fonksiyon bitimini belirten paranteze ulaşıldığında, kontrol çağıran fonksiyondaki kaldığı yere geri döner.

Fonksiyon tarafından bir değer döndürülüyorsa, bu değer çağrının yerini alır. Örneğin,

```
x = mutlak(-127);
```

x ’in değerini 127 yapacaktır; oysa daha yukarıda aynı fonksiyona yapılan ilk çağrıda geri döndürülen değer (-10) kullanılmamıştı, bu yüzden bu değer kaybolmuştu.

Örnek: a^b işlevini yürüten fonksiyon

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

float us (float a, int  b){
                        int i; float r=1;
                        for (i = 1; i<=b; i++)
                        r = r * a;
                        return  r;
                        }

int main(void)
      {
      int x; float y, z;
      printf ("Taban ve kuvvet degerlerini giriniz: ");
      scanf ("%f,%d",&y,&x);
      z=us(y,x);
      printf (" %f sayisinin %d. kuvveti %f 'dir ",y,x,z);
      getche();
      }
```

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

float y,z;int x;

void us(){
            int i;z=1.0;
            for(i=1;i<=x;i++)z=z*y;
           }

main(){
        printf ("Taban ve kuvvet degerlerini giriniz: ");
        scanf ("%f%d",&y,&x);
        us();
        printf ("%f sayisinin %d. kuvveti %f 'dir ",y,x,z);
        getche();
       }
```

Bir argüman istenildiği kadar karmaşık bir ifade olabilir. Argüman değerinin tipi fonksiyon tanımındaki karşılık gelen parametre için beklenen tipe uymalıdır, aksi takdirde, aşağıda açıklanacağı gibi, bulunması zor olan bazı hatalar meydana çıkabilir.

Dikkat: Fonksiyon argümanlarının hesaplanma sırası belirtilmemiştir. Örneğin, iki **`int`** argümanının toplamını veren topla isminde bir fonksiyonumuzun olduğunu varsayalım. Ayrıca, değeri 5 olan, i adında bir değişkenimiz olsun.

```
toplam = topla(i--, i);
```

yazdığımızda toplam ’ın değeri ne olacaktır? 9 mu 10 mu? Bu, argümanların hesaplanma sırasına bağlıdır.

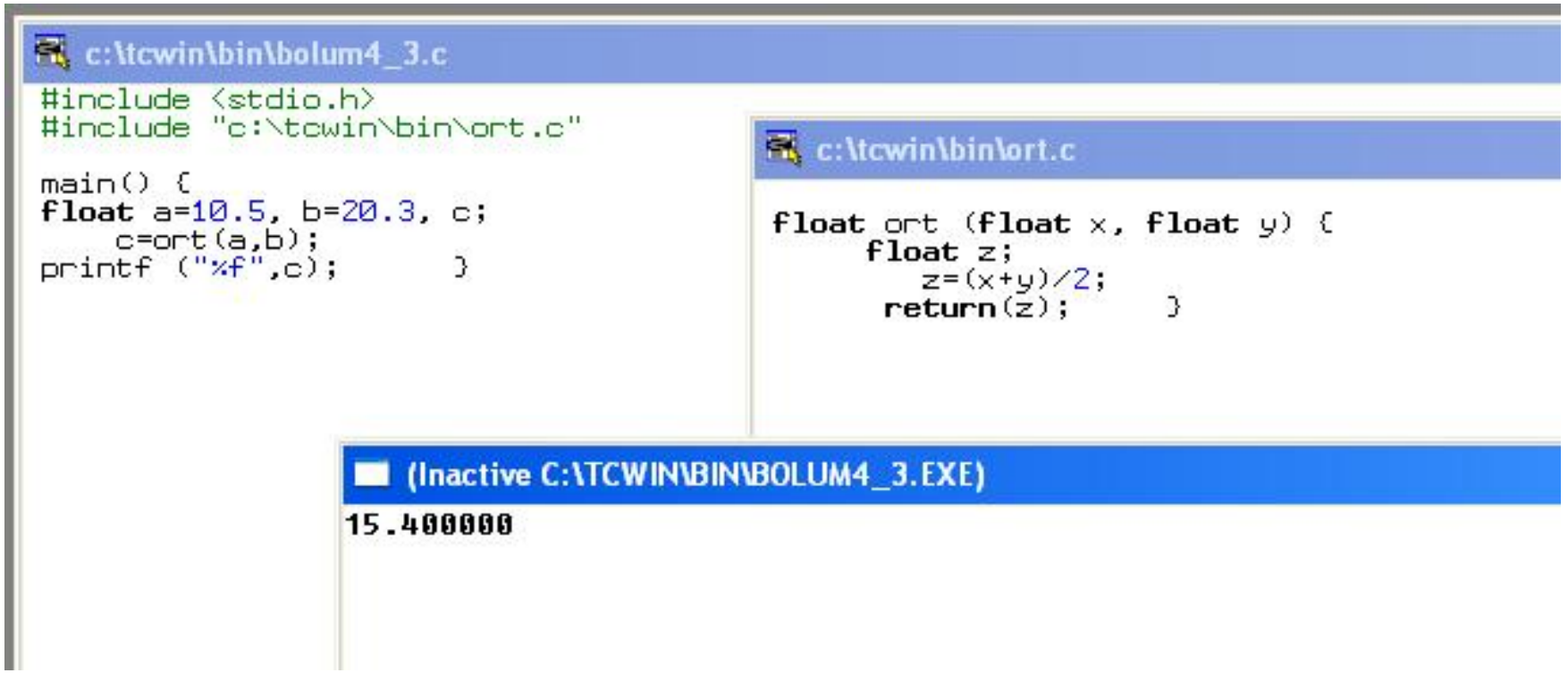
```
#include <stdio.h>
#include <conio.h>

int i,toplam;

int topla(int a,int b){
        return (a+b);
             }

main(){
           i=5;
           toplam=topla(i--,i);
           printf ("toplam=%d ",toplam);
           getche();
          }
```

Fonksiyonlar main fonksiyonundan ya da başka bir fonksiyondan çağrıldığında, çağrıldığı fonksiyondan sonra hazırlanmış ise önceden sembolik tanımı gerekebilir.

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

int topla(int,int);          /*Fonksiyonun sembolik tanımlanması. Topla fonksiyonu
                               main den sonra hazırlandığı için burada ön bildirimi yapıldı*/

main() {  int a=10, b=20, c;
    c=topla(a,b);                    // Fonksiyonun çağırılması
    printf ("%d",c);      }

int topla (int x, int y){            /*Fonksiyonun hazırlanması: Bu fonksiyon iki tam
                                       sayıyı toplar ve sonucu çağırıldığı yere döndürür. */
        int s;
        s=x+y;
        return(s);
                }
```

Fonksiyonların kaydedilerek çağırılması

Belirli bir amacı gerçekleştirecek şekilde hazırlanan bir fonksiyon diske fonksiyon adı ile kaydedilerek başka bir programdan çağırılabilir. Ancak bu durumda çağırılacağı programa include ile tanıtılmalıdır.



Örnek:

```
#include<stdio.h>

void sonuc (void);

int main(void) {
                    int x=10;
                    printf("x=%d",x);
                    sonuc();
          }


    void sonuc (void){
        printf ("Değer almayan ve üretmeyen fonksiyon");
        }
```

Main fonksiyonu **genellikle** hiçbir değer almadığı ve değer üretmeyeceği için void ile kullanılması uygundur.

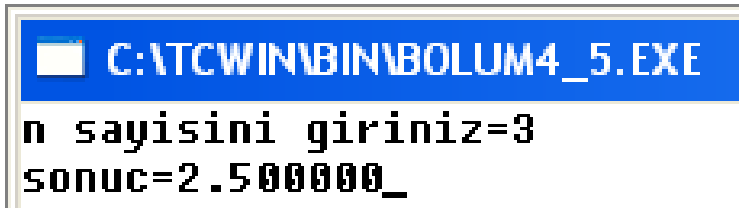
Örnek: İstenen para miktarını 20, 10 ve 5 'lik birimlere bölen ve sonucu ekrana gösteren C programı (Bankamatik Simülasyonu)

```
#include <stdio.h>
#include <conio.h>
void bankamatik(int para){
       int yirmilik,onluk,beslik,a = para; ;
                     if(a%5==0){
                            yirmilik = a/20; a -= yirmilik*20;
                            onluk = a/10; a -= onluk*10;
                            beslik = a/5; a -= beslik*5;
                            printf("\nYirmilik = %d",yirmilik);
                            printf("\nOnluk    = %d",onluk);
                            printf("\nBeslik   = %d\n",beslik);
                            }
else   printf("Girilen miktar 5 YTL ve katlari olmali!\a\n");
                          }
main(){
       int miktar;
       printf("Cekilecek para miktari (YTL) = ");
       scanf("%d",&miktar);   bankamatik(miktar);
            }
```

Örnek: Sayının tek veya çift olduğunu denetleyen program

```
#include<stdio.h>
void tek_mi_cift_mi( int sayi )
      {
      if( sayi%2 == 0 )
      printf( "%d, cift bir sayidir.\n", sayi );
            else
            printf( "%d, tek bir sayidir.\n", sayi );
                        }
 int main( void )
      {
  int girilen_sayi; printf( "Lütfen bir sayi giriniz> " );
      scanf( "%d",&girilen_sayi );
      tek_mi_cift_mi( girilen_sayi );
      return 0;
      }
```

Örnek: $\sum_{k=1}^{n} \frac{k}{k!}$ n tane terimin toplamı=?




```
# include <stdio.h>
# include <conio.h>
int faktoriyel (int a);
main(){
int n,k;     float seri=0;
     printf("n sayisini giriniz="); scanf("%d",&n);
       for (k=1;k<=n;k++)
         seri=seri + (float) k / faktoriyel (k ) ;
           printf("sonuc=%f",seri);
             getche();          }

faktoriyel (int a) {
int f=1,i;
          for(i=2;i<=a;i++) f=f*i;
return(f);              }
```

Kendi Kendini Çağıran Fonksiyonlar (Özyinelemeli-Rekürsif)

Kendini çağıran fonksiyonlara *özyinelemeli* fonksiyonlar adı verilir.

```
main()  {
          fonksiyon();
          }


fonksiyon () {
                ……….
                fonksiyon();
                ……..
                }
```

Özyinelemeli bir fonksiyon kendini dolaylı veya dolaysız olarak çağırabilir.
Özyinelemeli fonksiyonlar özel işlem gerektirirler.
Özyinelemeli fonksiyonları desteklemek için derleyicinin özel bir bellek düzeni kullanması; özyinelemeli fonksiyonlar yazmak için ise programcının biraz farklı bir düşünme tarzına sahip olması gerekir.
Yine de, özyinelemeli fonksiyonlar o kadar garip değildir; ve bazı durumlarda, eşdeğer özyinelemeli olmayan fonksiyon yerine özyinelemeli olanı kodlamak daha kolaydır. Örneğin,

Matematikte, *faktöriye*lin (!) tanımı şöyledir:
$n! = n \times (n\text{-}1) \times \cdots \times 2 \times 1.$

```
fakt (int n)
        {
        int i = 1;
        while (n) i *= n--;
        return i;
        }
```

Faktöriyelin başka bir eşdeğer tanımı da şöyledir:

$0! = 1$

$n! = n \times (n\text{-}1)!$

Bu değerlerden fonksiyonun başka değerlerin nasıl elde edileceğini gösteren *özyinelemeli yapı kuralıdır*.

Özyinelemeli tanım, özyinelemeli yapı kuralının temel fonksiyon değerleri üzerinde *sınırlı* sayıda uygulama sonucu sona eren bir yöntem tarif eder. Her $n \geq 0$ için yukarıdaki tanımın doğru olduğu gösterilebilir. Örneğin,

$3! = 3 \times 2! = 3 \times (2 \times 1!) = 3 \times 2 \times (1 \times 0!) = 3 \times 2 \times (1 \times 1) = 6$

Faktöriyel fonksiyonunun özyinelemeli uyarlaması şöyledir:

```
fakt (int n)
{
  return (n==0) ? 1 : n*fakt(n-1);
}
```

Bu kodlama ise tamamen matematik dilinden bilgisayar diline bir çeviridir. Argümanın, temel deyimin gereğine uyup uymadığı, yani sıfır olup olmadığı, kontrol edilir. Eğer öyle ise, temel fonksiyon değeri, yani 1, döndürülür. Aksi takdirde, özyinelemeli yapı kuralı uygulanır. Bu kural (3! örneğinde olduğu gibi) sınırlı sayıda tekrarlanarak doğru değer hesaplanır. Böylece temel deyim sonsuz sayıda özçağrıya engel olur.

```
#include<stdio.h>
#include<conio.h>
int faktor(int n);
main() {
int a,b;
       scanf ("%d",&a);
           b=faktor(a); printf("%d",b);
           getche();   }

int faktor (int n) {
           if (n==1)
                   return (n);
          else
             return (n*faktor(n-1));
 }
```

Örnek Verilen bir pozitif sayıya kadar olan sayıların toplamını özyinelemeli olarak bulan fonksiyon.

```
#include<stdio.h>
#include<conio.h>
int topla(int n);
main() {
int a,b;
        scanf (“%d”,&a);
            b=topla(a); printf(“%d”,b);
            getche();       }

int topla (int n) {
            if (n==1)
                    return (n);
           else
              return (n+topla(n-1));
 }
```

```
#include<stdio.h>
#include<conio.h>
int us(int a,int b){
          if (b==1)
                  return (a);
         else
            return (a*us(a,b-1));
 }
main() {
int a,b;
       scanf ("%d %d",&a,&b);
           printf("%d",us(a,b));
           getche();      }
```

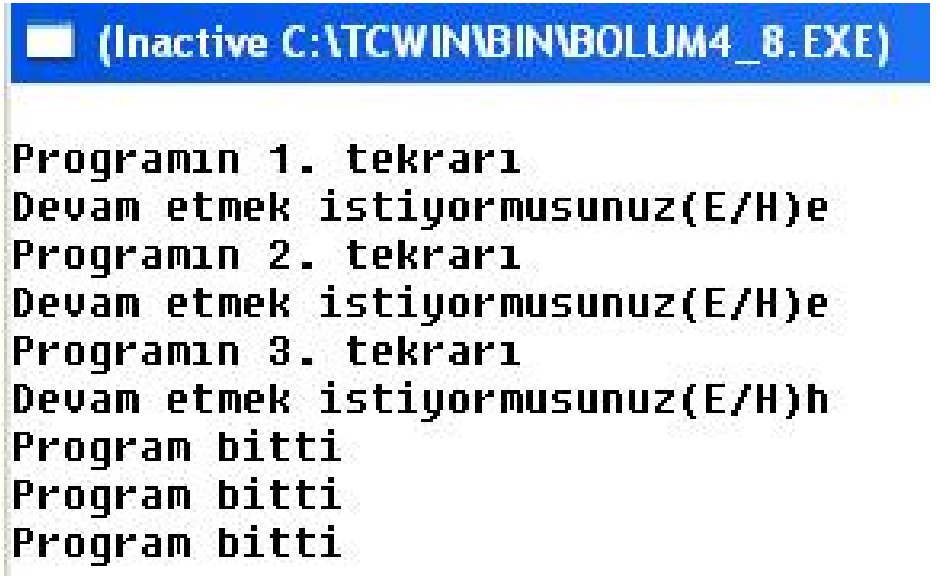
MAİN FONKSİYONU VE ARGÜMAN GİRİŞİ

Her C programında 1 tane main fonksiyonunun bulunması zorunludur. Main () de bir fonksiyon olduğu için kendi kendini çağırabilir veya başka bir fonksiyon tarafından çağrılabilir

```
main() {…….
        …… }


alt fonk() {……
        main();
                        ……}
```

Örnek

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

char c; int sayac=0;

main() {
            sayac++;
    printf("\nProgramın %d. tekrarı ",sayac);
    printf("\nDevam etmek istiyormusunuz(E/H)");
    c=getch();
    if (c=='e') main();
        printf("\nProgram bitti");
 }
```



Main fonksiyonunun parametreleri

Alt fonksiyonlar çağrıldıkları zaman değer alabildiklerine göre main() fonksiyonu da değer alabilmektedir. Main() fonksiyonu değerlerini sadece DOS ortamından alabilir. Dolayısıyla hazırlanan c programı derlenip çalıştırıldıktan sonra oluşan *.exe dosyası DOS ortamından main parametreleri ile çalıştırılmalıdır.

Main() fonksiyonu 2 tane parametre almaktadır

Argc(Argument Count): main()'e gönderilen komut satırındaki argümanların sayısını içerir. Komut satırında bulunan program ismi de bu sayıya dahildir. main(int argc, char *argv[]) parametreleri ile hazırlanan bir C dosyası çalıştırılırsa,

C:\>deneme.exe 4 17 36
argc=4

Argv(Argument Vektor): Komut satırında bulunan bilgilerin her biri bir string olacak şekilde bir pointer dizisi olarak argv değişkenine aktarılır.

Örnek

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


 main(int argc, char  *argv[] ){
      int i;

      printf("\n%d  ",argc);

      printf("\n argv0=%s",argv[0]);
      printf("\n argv1=%d",atoi(argv[1]));
      printf("\n argv2=%d",atoi(argv[2]));
      printf("\n argv3=%d",atoi(argv[3]));
getche();  }
```

Örnek Klavyeden argüman olarak girilen n tane sayının en küçüğünü ve en büyüğünü bulan programı yazınız.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
 main(int argc, char  *argv[] ){
            int i,ek,eb;
            ek=atoi(argv[1]);  eb=atoi(argv[1]);
     for (i=2;i<argc;i++){
                        if(ek>atoi(argv[i]))  ek=atoi(argv[i]);
              if(eb<atoi(argv[i]))  eb=atoi(argv[1]);
                                                  }
            printf("enküçük=%d,enbüyük=%d",ek,eb); getche();  }
```