

BSM101 Programlama Dilleri I

Hafta 13 Fonksiyonlar

Doç. Dr. Caner ÖZCAN

Fonksiyonlar

► Fonksiyonlar

- C 'de modüller
- Programlar kullanıcı tanımlı fonksiyonları ve kütüphane fonksiyonlarını birlikte kullanırlar.
- C standart kütüphanesi zengin bir fonksiyon çeşitliliğine sahiptir.

Fonksiyonların Faydaları

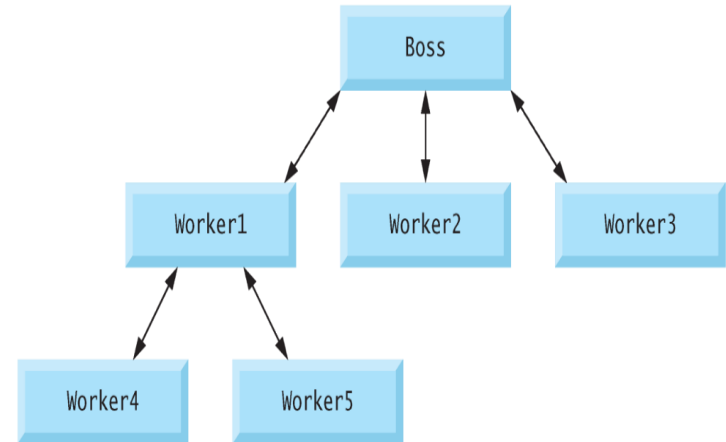
- Fonksiyonları faydaları
 - Böl ve yönet
 - Küçük parçalar veya bileşenlerden program oluştur.
 - Bu küçük parçalara modül denir.
 - Fonksiyonlar bir programı modüler hale getirir.
 - Her bir modül programın kendisinden daha yönetilebilirdir.
 - Yazılımın tekrar kullanılabilirliği
 - Mevcut fonksiyonlarınızı yeni bir program yapımında kullanabilirsiniz
 - Soyutlama – iç detaylar gizlenir(kütüphane fonksiyonları)
 - Kod tekrarı önlenir.

Fonksiyonlar

- Fonksiyonlar içinde tanımlanan tüm değişkenler yerel değişkenlerdir
 - Sadece tanımlandıkları fonksiyon içinde geçerlidirler
- Parametreler
 - Fonksiyonlar ile haberleşmede kullanılan bilgi
 - Yerel değişkendirler
- Fonksiyon çağırılması
 - Fonksiyon ismini ve argümanlarını (veri) belirt.
 - Fonksiyonlar birtakım işlemler gerçekleştirir.
 - Fonksiyonlar sonuç döndürürler

Fonksiyonlar

- Fonksiyon çağırma analogisi
 - Patron işçiden bir işi yapmasını ister
 - İşçi bilgi toplar, işi yapar ve sonucu patrona bildirir.
 - Bilgi gizleme: patron işin detaylarını bilmez.



Fonksiyon Tanımlama

- Fonksiyon tanımlama formatı:

```
geri_dönüş_değer_tipi fonksiyon_adı ( parameter_listesi )  
{  
    tanımlamalar_ve_ifadeler  
}
```

- fonksiyon_adı* herhangi bir geçerli tanımlayıcı olabilir.
- geri_dönüş_değer_tipi* fonksiyonu çağırana döndürülen sonucun veri tipidir.
- geri_dönüş_değer_tipi* void ise fonksiyon herhangi bir değer döndürmez.
- geri_dönüş_değer_tipi*, *fonksiyon_adı* ve *parametre_listesi* üçlüsü fonksiyon başlığı (header) olarak adlandırılır.

Fonksiyon Tanımlama

- *parametre_listesi* fonksiyonun çağırılırken aldığı parametreleri tanımlayan, virgülle birbirinden ayrılmış bir listedir.
- Eğer fonksiyon herhangi bir parametre almıyorsa, parametre listesi **void** olur.
- Her bir parametrenin tipi belirtilmelidir.

Fonksiyon Tanımlama

- Kırılmaç parantezler arasındaki ***tanımlamalar_ve_ifadeler*** fonksiyon gövdesini oluşturur.
- Fonksiyon gövdesinin diğer bir adı **blok**'tur.
- Değişkenler herhangi bir blok içerisinde tanımlanabilir ve bloklar iç içe olabilir.
- **Bir fonksiyon diğer bir fonksiyonun içerisinde tanımlanamaz.**

Fonksiyon Tanımlama

- Çağırılan bir fonksiyondan, fonksiyonun çağırıldığı noktaya kontrolün iade edilmesinin üç yöntemi vardır.
- Eğer fonksiyon herhangi bir sonuç döndürmüyorsa,
 - Sağ kurlangıç parantez ile kontrol iade edilmiş olunur.
 - Ya da basitçe `return`; ifadesi çalıştırılır.
- Eğer fonksiyon bir sonuç döndürüyorsa,
`return ifade;`
İfadenin değerini fonksiyonu çağırana döndürür.

Fonksiyon Prototipi

- Bir fonksiyonun künyesidir.
- Eğer fonksiyon tanımlaması çağırımdan sonra ise prototip tanımlanmalıdır.
- Aşağıdaki prototipe sahip fonksiyon
 - `int maximum( int x, int y, int z );`
 - 3 tamsayı parametre alır.
 - Geriye tamsayı döndürür.

Fonksiyon Prototipi

- Bir fonksiyon çağırımı prototipi ile uyuşmuyorsa derleme hatası oluşur.
- Eğer fonksiyon prototipi ile fonksiyon tanımlaması uyuşmuyorsa da hata oluşur.
- Fonksiyon prototiplerinin diğer bir önemli özelliği ise argümanların uygun bir veri tipine zorlanmasıdır.
- Örneğin, matematik fonksiyonu `sqrt` <math.h> içerisinde yer alan prototipinde `double` belirtilmiş olsa da `integer` argümanla da çağırılabilir. Fonksiyon yine de doğru çalışacaktır
 - `printf( "%.3f\n", sqrt( 4 ) );`
 - İfadesindeki `sqrt( 4 )` doğru bir şekilde değerlendirilir ve 2.000 değeri yazdırılır.

Fonksiyon Tanımlama

```
1  /* Fig. 5.4: fig05_04.c
2     Finding the maximum of three integers */
3  #include <stdio.h>
4
5  int maximum( int x, int y, int z ); /* function prototype */
6
7  /* function main begins program execution */
8  int main( void )
9  {
10     int number1; /* first integer */
11     int number2; /* second integer */
12     int number3; /* third integer */
13
14     printf( "Enter three integers: " );
15     scanf( "%d%d%d", &number1, &number2, &number3 );
16
17     /* number1, number2 and number3 are arguments
18        to the maximum function call */
19     printf( "Maximum is: %d\n", maximum( number1, number2, number3 ) );
20     return 0; /* indicates successful termination */
21 } /* end main */
22
```

Fonksiyon Tanımlama

```
23  /* Function maximum definition */
24  /* x, y and z are parameters */
25  int maximum( int x, int y, int z )
26  {
27      int max = x; /* assume x is largest */
28
29      if ( y > max ) { /* if y is larger than max, assign y to max */
30          max = y;
31      } /* end if */
32
33      if ( z > max ) { /* if z is larger than max, assign z to max */
34          max = z;
35      } /* end if */
36
37      return max; /* max is largest value */
38  } /* end function maximum */
```

Fig. 5.4 | Finding the maximum of three integers. (Part 3 of 4.)

Başlık Header Dosyaları

- Kütüphane fonksiyonlarının prototiplerini barındırırlar.
- `<stdlib.h>` , `<math.h>` , vs
- `#include <dosya_adı>` ile yüklenir.
 - `#include <math.h>`
- Özel başlık dosyaları
 - Fonksiyonlar içeren bir dosya oluştur.
 - `dosya_adı.h` şeklinde isim ile kaydet.
 - Başka dosyalar içerisinde `#include “dosya_adı.h”` olarak yükle.
 - Fonksiyonları tekrar kullan.

Başlık Header Dosyaları

- **math.h** → Matematik kütüphanesi
- **ctype.h** → Karakter özellikleri, küçük büyük harfe çevirme vs.
- **stdio.h** → Standart giriş / çıkış fonksiyonları
- **stdlib.h** → Sayıyı metne metni sayıya dönüştürme, hafıza yönetimi, rasgele sayılar ve bazı diğer faydalı fonksiyonlar.
- **string.h** → String işlemleri
- **time.h** → Zaman ve tarih fonksiyonları

Matematik Fonksiyonları

- Matematik kütüphane fonksiyonları
 - Temel matematik hesaplamaları yapar.
 - `#include <math.h>`
- Fonksiyonları çağırmak için kullanılacak format
 - `FonksiyonAdı( argümanı );`
- Eğer birden fazla argüman varsa, aralarında virgül kullan
- Tüm matematik fonksiyonlar double veri tipi döndürür
- Argümanlar sabit, değişken veya ifade olabilir

Matematik Fonksiyonları

Function	Description	Example
<code>sqrt(x)</code>	square root of x	<code>sqrt(900.0)</code> IS 30.0 <code>sqrt(9.0)</code> IS 3.0
<code>exp(x)</code>	exponential function e^x	<code>exp(1.0)</code> IS 2.718282 <code>exp(2.0)</code> IS 7.389056
<code>log(x)</code>	natural logarithm of x (base e)	<code>log(2.718282)</code> IS 1.0 <code>log(7.389056)</code> IS 2.0
<code>log10(x)</code>	logarithm of x (base 10)	<code>log10(1.0)</code> IS 0.0 <code>log10(10.0)</code> IS 1.0 <code>log10(100.0)</code> IS 2.0
<code>fabs(x)</code>	absolute value of x	<code>fabs(13.5)</code> IS 13.5 <code>fabs(0.0)</code> IS 0.0 <code>fabs(-13.5)</code> IS 13.5
<code>ceil(x)</code>	rounds x to the smallest integer not less than x	<code>ceil(9.2)</code> IS 10.0 <code>ceil(-9.8)</code> IS -9.0
<code>floor(x)</code>	rounds x to the largest integer not greater than x	<code>floor(9.2)</code> IS 9.0 <code>floor(-9.8)</code> IS -10.0

Matematik Fonksiyonları

Function	Description	Example
<code>pow(x, y)</code>	x raised to power y (x^y)	<code>pow(2, 7)</code> is 128.0 <code>pow(9, .5)</code> is 3.0
<code>fmod(x, y)</code>	remainder of x/y as a floating-point number	<code>fmod(13.657, 2.333)</code> is 1.992
<code>sin(x)</code>	trigonometric sine of x (x in radians)	<code>sin(0.0)</code> is 0.0
<code>cos(x)</code>	trigonometric cosine of x (x in radians)	<code>cos(0.0)</code> is 1.0
<code>tan(x)</code>	trigonometric tangent of x (x in radians)	<code>tan(0.0)</code> is 0.0

Örnek: Kare alan fonksiyon

```
#include <stdio.h>
float kareAl(float);

void main()
{
    int sayac;
    for(sayac = 1; sayac<=10; sayac++)
    {
        printf("Sayi:%d  Karesi:%d\n", sayac, kareAl(sayac));
    }

    printf("\n%.2f", kareAl(4.5));
}

float kareAl(float a)
{
    return a*a;
}
```

Örnek: Dört İşlem

```
#include <stdio.h>
int toplam(int, int);
int cikar(int, int);
int carp(int, int);
float bol(int, int);

void main()
{
    int secim,s1,s2;
    while(1)
    {
        printf("1-Topla\n2-Cikar\n3-Carp\n4-Bol\n5-Cikis\n");
        scanf("%d", &secim);
        printf("Sayilari gir:");
        scanf("%d %d", &s1, &s2);

        if(secim == 1)
            printf("Sonuc = %d", toplam(s1,s2));
        else if(secim == 2)
            printf("Sonuc = %d", cikar(s1,s2));
        else if(secim == 3)
            printf("Sonuc = %d", carp(s1,s2));
        else if(secim == 4)
            printf("Sonuc = %.2f", bol(s1,s2));
        else if(secim == 5)
            exit(0);
        else printf("Yanlis giris");
    }
}
```

```
int toplam(int a, int b)
{
    return a+b;
}

int cikar(int a, int b)
{
    return a-b;
}

int carp(int a, int b)
{
    return a*b;
}

float bol(int a, int b)
{
    return (float)a/b;
}
```

Örnek: Üs alma

```
#include <stdio.h>
double usAl(double, double);

void main()
{
    double a,b;
    printf("Taban ve us degeri gir:");
    scanf("%lf %lf", &a, &b);
    printf("%.2f", usAl(a,b));
}

double usAl(double x, double y)
{
    int sayac;
    double sonuc=1.0;
    for(sayac=0;sayac<y;sayac++)
    {
        sonuc *= x;
    }
    return sonuc;
}
```

Dizilerin Fonksiyonlara Gönderilmesi

- Bir diziye bir fonksiyona parametre olarak göndermek için parantez kullanmadan sadece dizinin ismi belirtilir.
 - **int** myArray [**24**];
 - **myFunction** (myArray, 24);
- Char dizilerinin aksine diğer türdeki diziler her hangi bir sonlandırma karakteri içermezler.
- Bu sebeple fonksiyonlara dizideki eleman sayısında parametre olarak gönderilir ki, fonksiyon uygun sayıda eleman üzerinde işlem yapsın.

Dizilerin Fonksiyonlara Gönderilmesi

- Dizilerin fonksiyonlara gönderilmesi referans ile çağırma işlemidir (call by reference).
- Dizinin adı aslında ilk elemanının adresidir.
- Fonksiyon böylece dizinin ilk elemanının hafıza nerede olduğunu bilir.
 - Orijinal hafıza bölgesinde işlem yapılır.
- Dizideki her hangi bir elemanın fonksiyona gönderilmesi ise değer ile çağırmadır (call by value).
 - Fonksiyona herhangi bir indisteki elemanın değeri gönderilir
 - `myArray [3]`
- Bir int dizi ve bir int değeri parametre olarak alan fonksiyon prototipi;
 - `void myArray (int [ ], int )`

Dizilerin Fonksiyonlara Gönderilmesi

```
1  /* Fig. 6.13: fig06_13.c
2     Passing arrays and individual array elements to functions */
3  #include <stdio.h>
4  #define SIZE 5
5
6  /* function prototypes */
7  void modifyArray( int b[], int size );
8  void modifyElement( int e );
9
10 /* function main begins program execution */
11 int main( void )
12 {
13     int a[ SIZE ] = { 0, 1, 2, 3, 4 }; /* initialize a */
14     int i; /* counter */
15
16     printf( "Effects of passing entire array by reference:\n\nThe "
17           "values of the original array are:\n" );
18
19     /* output original array */
20     for ( i = 0; i < SIZE; i++ ) {
21         printf( "%3d", a[ i ] );
22     } /* end for */
23 }
```


Dizilerin Fonksiyonlara Gönderilmesi

```
24     printf( "\n" );
25
26     /* pass array a to modifyArray by reference */
27     modifyArray( a, SIZE );
28
29     printf( "The values of the modified array are:\n" );
30
31     /* output modified array */
32     for ( i = 0; i < SIZE; i++ ) {
33         printf( "%3d", a[ i ] );
34     } /* end for */
35
36     /* output value of a[ 3 ] */
37     printf( "\n\nEffects of passing array element "
38           "by value:\n\nThe value of a[3] is %d\n", a[ 3 ] );
39
40     modifyElement( a[ 3 ] ); /* pass array element a[ 3 ] by value */
41
42     /* output value of a[ 3 ] */
43     printf( "The value of a[ 3 ] is %d\n", a[ 3 ] );
44     return 0; /* indicates successful termination */
45 } /* end main */
46
```

Dizilerin Fonksiyonlara Gönderilmesi

```
47  /* in function modifyArray, "b" points to the original array "a"
48     in memory */
49  void modifyArray( int b[], int size )
50  {
51      int j; /* counter */
52
53      /* multiply each array element by 2 */
54      for ( j = 0; j < size; j++ ) {
55          b[ j ] *= 2;
56      } /* end for */
57  } /* end function modifyArray */
58
59  /* in function modifyElement, "e" is a local copy of array element
60     a[ 3 ] passed from main */
61  void modifyElement( int e )
62  {
63      /* multiply parameter by 2 */
64      printf( "Value in modifyElement is %d\n", e *= 2 );
65  } /* end function modifyElement */
```

Dizilerin Fonksiyonlara Gönderilmesi

Effects of passing entire array by reference:

The values of the original array are:

0 1 2 3 4

The values of the modified array are:

0 2 4 6 8

Effects of passing array element by value:

The value of a[3] is 6

Value in modifyElement is 12

The value of a[3] is 6

Çok Boyutlu Dizileri Fonksiyonlara Gönderme

- Tek boyutlu dizileri fonksiyona göndermekten farklı değildir.
- Her bir boyut için köşeli parantez kullanın, ilk boyut hariç diğerleri için büyüklük belirtin.
 - `void writeMatrice (int [ ] [ 4 ], int rowNumber);`
 - Bu tanımlama 4 sütuna sahip her matris için farklı satır numaralarına sahip olsalar da geçerli olur.
 - `void writeMatrice (int [ ] [ 3 ] [ 4 ], int rowNumber);`

Çok Boyutlu Dizileri Fonksiyonlara Gönderme

```
1  /* Fig. 6.21: fig06_21.c
2     Initializing multidimensional arrays */
3  #include <stdio.h>
4
5  void printArray( const int a[][ 3 ] ); /* function prototype */
6
7  /* function main begins program execution */
8  int main( void )
9  {
10     /* initialize array1, array2, array3 */
11     int array1[ 2 ][ 3 ] = { { 1, 2, 3 }, { 4, 5, 6 } };
12     int array2[ 2 ][ 3 ] = { 1, 2, 3, 4, 5 };
13     int array3[ 2 ][ 3 ] = { { 1, 2 }, { 4 } };
14
15     printf( "Values in array1 by row are:\n" );
16     printArray( array1 );
17
18     printf( "Values in array2 by row are:\n" );
19     printArray( array2 );
20
21     printf( "Values in array3 by row are:\n" );
22     printArray( array3 );
23     return 0; /* indicates successful termination */
24 } /* end main */
```

Çok Boyutlu Dizileri Fonksiyonlara Gönderme

```
25
26 /* function to output array with two rows and three columns */
27 void printArray( const int a[][ 3 ] )
28 {
29     int i; /* row counter */
30     int j; /* column counter */
31
32     /* loop through rows */
33     for ( i = 0; i <= 1; i++ ) {
34
35         /* output column values */
36         for ( j = 0; j <= 2; j++ ) {
37             printf( "%d ", a[ i ][ j ] );
38         } /* end inner for */
39
40         printf( "\n" ); /* start new line of output
41     } /* end outer for */
42 } /* end function printArray */
```

Values in array1 by row are:
1 2 3
4 5 6
Values in array2 by row are:
1 2 3
4 5 0
Values in array3 by row are:
1 2 0
4 0 0

Saklama Sınıfları

- Nesne kendi bloğu içinde oluşturulur ve yok edilir
 - **auto**: yerel değişkenler için ön tanımlıdır
 - `auto double x, y;`
 - **register**: değişkeni yüksek hızlı kaydedicilere yerleştirmeye çalışır
 - `register int counter= 1;`

Saklama Sınıfları

- **Statik Saklama**

- Değişken tüm programın çalışması sürecinde var olur.
- Ön tanımlı değeri sıfırdır.
- **static**: fonksiyonlar içinde tanımlanan yerel değişkenlerdir.
 - Fonksiyon sonlandıktan sonra değişken değeri saklanır
 - Sadece tanımlandıkları fonksiyonda geçerlidirler

Saklama Sınıfları

- **Dosya kapsama alanı**

- Bir fonksiyon dışında tanımlanan bir tanımlayıcı dosya kapsama alanına sahiptir.
- Böyle bir tanımlayıcı tanımlandıkları noktadan itibaren dosya sonuna kadar tüm fonksiyonlar içinde geçerlidir
- Global değişkenler, fonksiyon tanımlamaları hep dosya kapsama alanına sahiptir.

Saklama Sınıfları

- **Blok Alanı**

- Blok içinde tanımlanan tanımlayıcılar
- Blok alanı tanımlandığı noktadan başlar sağ kırlangıç paranteze kadar devam eder.
- Değişkenler, yerel değişkenler ve fonksiyonlar için kullanılır.
- Dış bloklar eğer iç blokta aynı isimle değişken varsa iç bloktan gizlenirler.

Saklama Sınıfları

```
1  /* Fig. 5.12: fig05_12.c
2     A scoping example */
3  #include <stdio.h>
4
5  void useLocal( void ); /* function prototype */
6  void useStaticLocal( void ); /* function prototype */
7  void useGlobal( void ); /* function prototype */
8
9  int x = 1; /* global variable */
10
11 /* function main begins program execution */
12 int main( void )
13 {
14     int x = 5; /* local variable to main */
15
16     printf("local x in outer scope of main is %d\n", x );
17
18     { /* start new scope */
19         int x = 7; /* local variable to new scope */
20
21         printf( "local x in inner scope of main is %d\n", x );
22     } /* end new scope */
23 }
```

Saklama Sınıfları

```
24 printf( "local x in outer scope of main is %d\n", x );
25
26 useLocal(); /* useLocal has automatic local x */
27 useStaticLocal(); /* useStaticLocal has static local x */
28 useGlobal(); /* useGlobal uses global x */
29 useLocal(); /* useLocal reinitializes automatic local x */
30 useStaticLocal(); /* static local x retains its prior value */
31 useGlobal(); /* global x also retains its value */
32
33 printf( "\nlocal x in main is %d\n", x );
34 return 0; /* indicates successful termination */
35 } /* end main */
36
37 /* useLocal reinitializes local variable x during each call */
38 void useLocal( void )
39 {
40     int x = 25; /* initialized each time useLocal is called */
41
42     printf( "\nlocal x in useLocal is %d after entering useLocal\n", x );
43     x++;
44     printf( "local x in useLocal is %d before exiting useLocal\n", x );
45 } /* end function useLocal */
46
```

Fig. 5.12 | Scoping example. (Part 2 of 4.)

Saklama Sınıfları

```
47  /* useStaticLocal initializes static local variable x only the first time
48     the function is called; value of x is saved between calls to this
49     function */
50  void useStaticLocal( void )
51  {
52     /* initialized only first time useStaticLocal is called */
53     static int x = 50;
54
55     printf( "\nlocal static x is %d on entering useStaticLocal\n", x );
56     x++;
57     printf( "local static x is %d on exiting useStaticLocal\n", x );
58 } /* end function useStaticLocal */
59
60 /* function useGlobal modifies global variable x during each call */
61 void useGlobal( void )
62 {
63     printf( "\nglobal x is %d on entering useGlobal\n", x );
64     x *= 10;
65     printf( "global x is %d on exiting useGlobal\n", x );
66 } /* end function useGlobal */
```

Saklama Sınıfları

```
local x in outer scope of main is 5  
local x in inner scope of main is 7  
local x in outer scope of main is 5
```

```
local x in useLocal is 25 after entering useLocal  
local x in useLocal is 26 before exiting useLocal
```

```
local static x is 50 on entering useStaticLocal  
local static x is 51 on exiting useStaticLocal
```

```
global x is 1 on entering useGlobal  
global x is 10 on exiting useGlobal
```

```
local x in useLocal is 25 after entering useLocal  
local x in useLocal is 26 before exiting useLocal
```

```
local static x is 51 on entering useStaticLocal  
local static x is 52 on exiting useStaticLocal
```

```
global x is 10 on entering useGlobal  
global x is 100 on exiting useGlobal
```

```
local x in main is 5
```

QUIZ

Klavyeden Integer tipinde tek boyutlu bir dizi alan ve dizide kaç tane 4'ün katı olan sayı olduğunu döndüren fonksiyonu ve ana programı yazın.

Kaynaklar

- ▶ Doç. Dr. Fahri Vatansever, “Algoritma Geliştirme ve Programlamaya Giriş”, Seçkin Yayıncılık, 12. Baskı, 2015.
- ▶ J. G. Brookshear, “Computer Science: An Overview 10th Ed.”, Addison Wisley, 2009.
- ▶ Kaan Aslan, “A’dan Z’ye C Klavuzu 8. Basım”, Pusula Yayıncılık, 2002.
- ▶ Paul J. Deitel, “C How to Program”, Harvey Deitel.
- ▶ Bayram AKGÜL, C Programlama Ders notları