Bölüm 5 - Fonksiyonlar

İçerik

- Böl ve Zaptet
  - Programı küçük parçalardan veya bileşenlerden oluştur
    - Bu küçük parçalara modüller denir
  - Bir parça, tek bir bütün-programa göre daha kolay yönetilebilir

5.2 C de Program Modülleri

- Fonksiyonlar
  - C deki modüllerdir
  - Programlar kullanıcı ve kütüphane tanımlı programlardan oluşur
    - C standard kütüphanesi geniş çaplı fonksiyonlara sahiptir
- Fonksiyon Çağrımı
  - Fonksiyonlar çağrıldığında
    - Fonksiyon adı ve argümentleri (data) verilir
    - Fonksiyon ilgili işlemlerini yapar
    - Sonuçları geri gönderir
  - Fonksiyon çağrı benzetmesi:
    - Patron işçilere işi tamamlamalarını söyler
      - İşçiler bilgiyi alır, görevi yapar, sonuçları iade eder
      - Bilgi saklaması: patron detayları bilmez

5.3 Math Kütüphanesi Fonksiyonları

- Math kütüphanesi fonksiyonları
  - Bilinen matematik işlemleri yapar
  - **`#include <math.h>`**
- Fonksiyonları çağırma formatı
  - **`FonksiyonAdı(`** *argüment* **`);`**
    - Çoklu argümentler virgülle ayrılır
  - **`printf( "%.2f", sqrt( 900.0 ) );`**
    - **`sqrt`** fonksiyonunu çağırır ve argümentin karekökünü gönderir
    - Tüm matematik fonksiyonları **`double`** veri tipi gönderir
  - Argümentler; sabit, değişken veya ifade olabilir

5.4 Fonksiyonlar

- Fonksiyonlar
  - Bir programı modülleştirir
  - Fonksiyon içinde tanımlanan tüm değişkenler iç (yerel) değişkendir
    - Sadece fonksiyon tanımlandığında bilinir
  - Parametreler
    - Fonksiyonlar arası bilgi iletişimi sağlar
    - Yerel değişkenlerdir
- Fonksiyonların getirileri
  - Böl ve zaptet
    - Kullanışlı program geliştirmek
  - Yazılım tekrar-kullanılabilirliği
    - Yeni programlar için, hazır fonksiyonları inşa blokları şeklinde kullan
    - Soyutlama – iç detaylar saklıdır (kütüphane fonksiyonları)
  - Kod tekrarından kaçın

- Fonksiyon tanımlama formatı

Dönen_değer_tipi fonksiyon_adı(parametre-listesi)

```
{
  deklarasyonlar ve deyimler
}
```

  - Fonksiyon adı: herhangi bir geçerli belirleyici
  - Dönen değer tipi: Sonucun veri tipi (default **`int`**)
    - **`void`** – fonksiyonun hiç bir değer göndermediğini belirtir
  - Parametre listesi: parametreleri tanımlar-virgülle ayrılır
    - Parametrenin tipi **`int`** olmadıkça her bir parametrenin tipi açıkça yazılmalıdır

- Fonksiyon tanımlama formatı (Devam)

Dönen_değer_tipi fonksiyon_adı(parametre-listesi)

```
{
  deklarasyonlar ve deyimler
}
```

  - Deklarasyonlar ve deyimler: fonksiyon gövdesi (bloğu)
    - Değişkenler bloklar içinde tanımlanabilir (iç içe olabilir)
    - Fonksiyonlar başka fonksiyonlar içinde tanımlanmamalıdır
  - Dönüş kontrolü
    - Hiç bir şey dönmezse
      - **`return;`**
      - Veya, sağ paranteze erişinceye kadar
    - Döndürülen bir şey varsa
      - **`return`** *ifade*`;`

5.5 Fonksiyon Tanımları

```
/* Örnek 5.4
   Üç tamsayının maksimumunu bulma*/
#include <stdio.h>
int maximum( int x, int y, int z ); // fonksiyon prototipi
int main()
{
   int sayi1, sayi2, sayi3;
   printf( "Üç tamsayı gir: " );
   scanf( "%d%d%d", &sayi1, &sayi2, &sayi3);
   printf( "Maksimum =: %d\n", maximum( sayi1, sayi2, sayi3) );
   return 0;
}

// maxsimum fonksiyonunun tanımlanması
// x, y ve z parametreler
int maximum( int x, int y, int z )
{
   int max = x;      /* x i en büyük kabul ettik */
   if ( y > max ) { /* eğer y, x den büyük ise yer değiştir*/
      max = y;
   }

   if ( z > max ) { /* eğer z, max dan büyük ise yer değiştir*/
      max = z;
   }
   return max;
}
```

Fonksiyon çağrımı

Çıktı:

```
Üç tamsayı gir: 22 85 17
Maksimum = 85
```

5.6 Fonksiyon Prototipleri

- Fonksiyon prototipi
  - Fonksiyon adı
  - Parametreler – fonksiyonun girdileri
  - Dönen değer tipi – fonksiyonun gönderdiği veri tipi (default **`int`**)
  - Fonksiyonları geçerli kılmak için kullanılır
  - Sadece, eğer fonksiyon tanımı programda kullanıldıktan sonra geliyorsa prototipe gereksinim vardır

```
int maximum( int, int, int );
```

  - prototipindeki fonksiyon
    - 3 **`int (tamsayı)`** `alır`
    - `Bir` **`int (tamsayı)`** `gönderir`
- Yükseltme kuralları ve Değiştirme
  - Daha alt tipe değiştirmek hataya yol açabilir

5.7 Header Dosyaları

- Header dosyaları
  - Kütüphane fonksiyonları fonksiyon için prototipleri içerir
  - **<stdlib.h>** , **<math.h>** , vs
  - **#include <dosya adı>** ile yüklenir

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

- Kişisel header dosyaları
  - Fonksiyon dosyası oluştur
  - **dosyaadı.h** olarak kaydet
  - **#include "dosyaadı.h"** ile programa yükle
  - Başka programlarda da kullan

İki tip çağırma:

- Değer ile çağırma
  - Fonksiyona aktarılan argümentlerin kopyası
  - Fonksiyonda değerler değişse bile programda değişmez
  - Fonksiyon argüment değişimine gerek duymuyorsa kullan
    - Kazara değişimlerden kaçın
- Referans ile çağırma
  - Orjinal argümentleri aktarır
  - Fonksiyondaki değişim orjinali etkiler
  - Sadece güvenilir fonksiyonlarda kullan

- Şimdilik değer ile çağırma üzerinde duracağız

5.9 Rasgele Sayı Üreticisi

- **`rand`** fonksiyonu
  - **`<stdlib.h>`** -i yükle
  - **`i = rand();`** **`0`** ve **`RAND_MAX`** (en azından **`32767`**) arasında bir rasgele "random" tamsayı seçer
  - Önseçim
    - Bir "random" sayı dizisi kurar
    - Her çağrıldığında aynı diziyi verir
- Ölçekleme
  - **`1`** ve **`n`** arasında bir rasgele sayı

```
1 + ( rand() % n );
```

    - **`rand() % n;`** **`0`** ve **`n - 1`** arasında bir sayı seçer
    - **`1`** ve **`n`** arasında bir seçim için **`1`** ekle
    - **`1 + ( rand() % 6);`**
      - **`1`** ve **`6`** arasında sayı

5.9 Rasgele Sayı Üreticisi

- **`srand`** fonksiyonu
  - **`<stdlib.h>`**
  - Bir tamsayı çekirdeği alır ve yerine bir "rasgele" dizi atar

    `srand(` *seed* **`);`**
  - **`srand( time( NULL ) );  // <time.h> i yükle`**
    - **`time( NULL )`**
      - Program derleme anındaki zamanı gönderir
      - çekirdeği "rasgele alır"

5.9 Rasgele Sayı Üreticisi

```
/* Örnek 5.9
Zar atma  */
#include <stdlib.h>
#include <stdio.h>
int main()
{
   int i;
   unsigned seed;
   printf( " seed i gir: " );
   scanf( "%u", &seed );
   srand( seed );
   for ( i = 1; i <= 10; i++ ) {
      printf( "%10d", 1 + ( rand() % 6 ) );
      if ( i % 5 == 0 )
         printf( "\n" );
   }
   return 0;
}
```

Çıktı:

```
Seed i gir: 67
         6         1         4         6         2
         1         6         1         6         4
```

```
Seed i gir: 465
         2         4         6         1         6
         1         1         3         6         2
```

```
Seed i gir: 4
         6         1         3         5         5
         1         4         1         3         2
```

```
Seed i gir: 67
         6         1         4         6         2
         1         6         1         6         4
```

5.10 Örnek: Şans Oyunu

- Barbut Simulatörü
- Kurallar
  - İki zar at
    - İlk atışta toplam 7 veya 11 atan kazanır
    - İlk atışta toplam 2, 3, veya 12 kaybeder
    - İlk atışta toplam 4, 5, 6, 8, 9, 10 - oyuncunun "puanları"
      - Sonraki atışlarda toplam ilk atışa aynı ise kazanır.
      - Toplam 7 ise kaybeder.
      - Diğer toplamlarda atış devam eder.

5.10 Örnek: Şans Oyunu

```
/* Örnek 5.10
   Barbut */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int zarat( void );
int main()
{
   int oyundurumu, top, puanim;

   srand( time( NULL ) );

   top = zarat();              /* ilk zar atışı */

   switch ( top ) {
      case 7: case 11:           /* ilk atışta kazandı */
         oyundurumu = 1;

         break;
   case 2: case 3: case 12:   /* ilk atışta kaybetti */
         oyundurumu = 2;
         break;

   default:                   /* puanı hatırla */

         oyundurumu = 0;

         puanim = top;
         printf( "Puanım= %d\n", puanim );

         break;
   }
```

5.10 Örnek: Şans Oyunu

```
    while ( oyundurumu == 0 ) {     /* atışa devam */
       sum = zarat();
    if ( top == puanim )         /* kazandı */
          oyundurumu = 1;
       else
          if ( top == 7 )           /* kaybetti 7 */
             oyundurumu = 2;
    }
    if ( oyundurumu == 1 )
       printf( "Kazandı" );
else
   printf( "Kaybetti\n" );
return 0;
}
int zarat( void ) /* fonksiyon tanımlaması */
{
    int zar1, zar2, toplam;

    zar1 = 1 + ( rand() % 6 );
    zar2 = 1 + ( rand() % 6 );
    toplam= zar1 + zar2;
    printf( "Atılan zarlar %d + %d = %d\n", zar1, zar2, toplam );
    return toplam;
```

5.10 Örnek: Şans Oyunu

Çıktı:

```
Atılan zarlar 6 + 5 = 11
Kazandı
```

```
Atılan zarlar 6 + 6 = 12
Kaybetti
```

```
Atılan zarlar 4 + 6 = 10
Puanım 10
Atılan zarlar 2 + 4 = 6
Atılan zarlar 6 + 5 = 11
Atılan zarlar 3 + 3 = 6
Atılan zarlar 6 + 4 = 10
Kazandı
```

```
Atılan zarlar 1 + 3 = 4
Puanım 4
Atılan zarlar 1 + 4 = 5
Atılan zarlar 5 + 4 = 9
Atılan zarlar 4 + 6 = 10
Atılan zarlar 6 + 3 = 9
Atılan zarlar 1 + 2 = 3
Atılan zarlar 5 + 2 = 7
Kaybetti
```

5.11 Depolama Sınıfları

- Depolama sınıfı belirticisi
  - Depolama zamanı – bir nesne bellekte ne kadar tutulacak
  - Odak – objenin programda referens edildiği yer
  - Bağlantı – Bir belirleyicinin bilindiği dosyaları belirtir (Bölüm 14)
- Otomatik depolama
  - Obje bulunduğu blok içinde yaratılır ve yokedilir
  - **`auto`**: yerel değişkenler için default (halihazırda)dır

```
auto double x, y;
```

  - **`register`**: yüksek hızlı kayıtlar koymayı dener
    - sadece otomatik değişkenler için kullanılabilir

```
register int sayac = 1;
```

5.11 Depolama Sınıfları

- Static (durağan) depolama
  - Değişkenler tüm program boyunca geçerlidir
  - Default değeri sıfırdır
  - **`static`**: fonksiyonlarda tanımlanan yerel değişkenler.
    - Fonksiyon çağrıldıktan sonra değerini korur
    - Sadece kullanıldığı fonksiyonda tanınır
  - **`extern`**: Global değişkenler ve fonksiyonlar için default-tur
    - Herhangi bir fonksiyonda tanınır

5.12 Tanınma Kuralları

- Dosya tanınması
  - Fonksiyon dışında tanımlanan, tüm fonksiyonlarca tanınan belirleyici
  - Global değişkenler, fonksiyon tanımları, fonksiyon prototipleri için kullanılır
- Fonksiyon tanınması
  - Sadece bir fonksiyonun içinde referans verilebilir
  - Sadece etiketler için kullanılır (**`start:`**, **`case:`** , vs.)

5.12 Odaklama Kuralları

- Blok tanınma
  - Bir blok içinde tanımlanan belirleyici
    - Blok tanınması tanımlandığı noktada başlar, sağ parantezde biter
  - Değişkenler, fonksiyon parametreleri (fonksiyonların yerel değişkenleri) için kullanılır
  - Eğer iç blokta aynı adlı bir değişken var ise, dış blok iç bloktan gizlenir
- Fonksiyon prototipi tanınması
  - Parametre listesindeki belirleyiciler için kullanılır

```c
/* Örnek 5.12
   tanınma örneği */
#include <stdio.h>

void a( void );   /* fonksiyon prtotipi */
void b( void );   /* fonksiyon prtotipi */
void c( void );   /* fonksiyon prtotipi */

int x = 1;        /* dış değişken */

int main()
{
   int x = 5;   /* main-de iç değişken */

   printf("main-in dış odağındaki iç x= %d\n", x );

   {            /* yeni odak başlat */
      int x = 7;

      printf( " main-in iç odağındaki iç x= %d\n", x );
   }            /* yeni odağı bitir */

   printf( " main-in dış odağındaki iç x= %d\n", x );

   a();         /* a bir otomatik iç x e sahip*/
   b();         /* b bir static iç x e sahip */
   c();         /* c global(dış) x kullanıyor*/
   a();         /* a otomatik iç  x-i yeniden belirliyor */
   b();         /* static iç x eski değerini koruyor */
   c();         /* global x de değerini koruyor */
```

```
31
32      printf( "main deki iç x= %d\n", x );
33      return 0;
34  }
35
36  void a( void )
37  {
38      int x = 25;  /* her a çağrılışında belirle */
39
40      printf( "\a-ya girildikten sonra; a daki iç x =%d \n", x );
41      ++x;
42      printf( "a dan çıkılmadan önce; a daki iç x= %d \n", x );
43  }
44
45  void b( void )
46  {
47       static int x = 50;  /* sadece static değer belirleme */
48                           /* b ilk çağrıldığında */
49       printf( "\nb ye girildiğinde iç  static x= %d  b\n", x );
50       ++x;
51       printf( "b den çıkılırken iç static x = %d \n", x );
52  }
53
54  void c( void )
55  {
56      printf( "\nc ye girildiğinde global x = %d c\n", x );
57      x *= 10;
58      printf( "c den çıkılırken global x = %d \n", x );
59  }
```

5.12 Odaklama Kuralları

Çıktı:

```
main-in dış odağındaki iç x= 5
main-in iç odağındaki iç x= 7
main-in dış odağındaki iç x= 5

a-ya girildikten sonra; a daki iç x = 25
a dan çıkılmadan önce; a daki iç x= 26

b ye girildiğinde iç  static x= 50
b den çıkılırken iç static x = 51

c ye girildiğinde global x =1
c den çıkılırken global x =10

a-ya girildikten sonra; a daki iç x = 25
a dan çıkılmadan önce; a daki iç x= 26

b ye girildiğinde iç  static x= 51
b den çıkılırken iç static x = 52

c ye girildiğinde global x =10
c den çıkılırken global x =100
main deki iç x= 5
```

5.13 Rekürans

- Rekürans fonksiyonları
  - Kendini çağıran fonksiyonlar
  - Sadece bir taban yapı çözülür
  - Problem alt parçalara bölünür
    - Ne yapılabilir
    - Ne yapılamaz
      - Yapılamayanların neresi orjinal probleme benzer
      - Yapılamayacakları çözmek için fonksiyon kendinin bir kopyasını yükler(rekürans adımları)
  - Sonunda taban yapısı çözülür
    - Öncekileri al ve tümünü çözmek için olanlara ekle

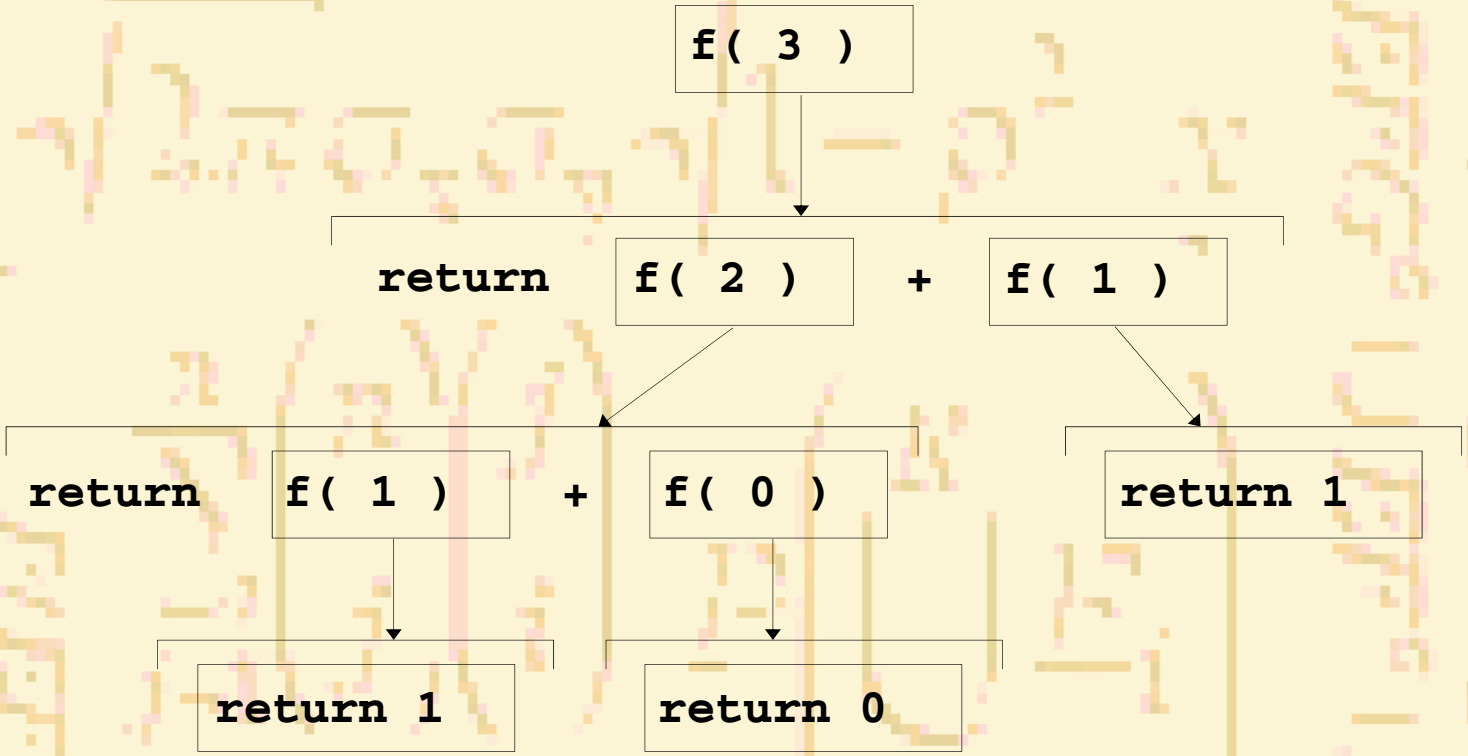
- Örnek: faktoriyel
  - `5! = 5 * 4 * 3 * 2 * 1`
  - Dikkat edilirse
    - `5! = 5 * 4!`
    - `4! = 4 * 3!` ...
  - Faktoriyeli ardarda hesaplar
  - Taban yapıyı çöz (`1! = 0! = 1`) ve reküransda kullan
    - `2! = 2 * 1! = 2 * 1 = 2;`
    - `3! = 3 * 2! = 3 * 2 = 6;`

5.14 Rekürans Örneği: Fibonacci Serisi

- Fibonacci serisi: 0, 1, 1, 2, 3, 5, 8...
  - Her sayı önceki ikisinin toplamı
  - Ardışık olarak çözülebilir:
    - **fib(n) = fib(n - 1) + fib(n - 2)**
  - **fibonacci** fonksiyonu için kod

```
long fibonacci( long n )
{
   if (n == 0 || n == 1)  // taban durum
      return n;
   else
      return fibonacci( n - 1) +
         fibonacci( n - 2 );
}
```

- **`fibonacci`** fonksiyonu için ardışık çağrım



5.14 Rekürans Örneği: Fibonacci Serisi

```
1  /* Örnek 5.15
2     fibonacci rekürans fonksiyonu */
3  #include <stdio.h>
4
5  long fibonacci( long );
6
7  int main()
8  {
9     long sonuc, sayi;
10
11    printf( "Bir tamsayı gir: " );
12    scanf( "%ld", &sayi );
13    sonuc = fibonacci( sayi );
14    printf( "Fibonacci( %ld ) = %ld\n", sayi, sonuc );
15    return 0;
16 }
17
18 /* fibonacci ardışık çağrım*/
19 long fibonacci( long n )
20 {
21    if ( n == 0 || n == 1 )
22       return n;
23    else
24       return fibonacci( n - 1 ) + fibonacci( n - 2 );
25 }
```

5.14 Rekürans Örneği: Fibonacci Serisi

Çıktı:

```
Bir tamsayı gir: 2
Fibonacci(2) = 1

Bir tamsayı gir : 3
Fibonacci(3) = 2

Bir tamsayı gir : 4
Fibonacci(4) = 3

Bir tamsayı gir : 5
Fibonacci(5) = 5

Bir tamsayı gir : 6
Fibonacci(6) = 8

Bir tamsayı gir : 10
Fibonacci(10) = 55

Bir tamsayı gir : 20
Fibonacci(20) = 6765

Bir tamsayı gir : 30
Fibonacci(30) = 832040

Bir tamsayı gir : 35
Fibonacci(35) = 9227465
```

5.15 Rekürans ve Iterasyon

- Tekrar
  - İterasyon: açık döngü
  - Rekürans: ardışık fonksiyon çağrımı
- Sonlanması
  - İterasyon: döngü koşulu sağlanmaz
  - Rekürans: taban durumu bulunur
- Her ikisi de sonsuz döngüye girebilir
- Denge
  - Performans (iterasyon) ile iyi yazılım mühendisliği(rekürans) arasındaki seçim