ALGORİTMA VE PROGRAMLAMA I

Yrd. Doç. Dr. Deniz KILINÇ
deniz.kilinc@cbu.edu.tr

YZM 1101

Celal Bayar Üniversitesi Hasan Ferdi Turgutlu
Teknoloji Fakültesi

Genel Bakış…

- Giriş
- Fonksiyonlara Genel Bakış
- C ve Fonksiyonlar
- Fonksiyon Tanımı

8. BÖLÜM

Fonksiyonlar

Giriş

- Gerçek hayattaki yazılım problemlerini çözen çoğu bilgisayar programları ve yazılımları, şu ana kadar öğrendiklerimizden çok daha geniş ve karmaşık bir yapıya sahiptir.
- Tecrübeler bu tür geniş programları yazmanın en iyi yolunun, küçük parçaları ya da her biri orijinal programdan daha kolay kullanılabilecek modülleri (daha önceden hazırlanmış program parçacıkları) birleştirmek olduğunu göstermiştir.
- Bu tekniğe, **böl ve ele geçir** (**divide & conquer**) denir

Fonksiyonlara Genel Bakış

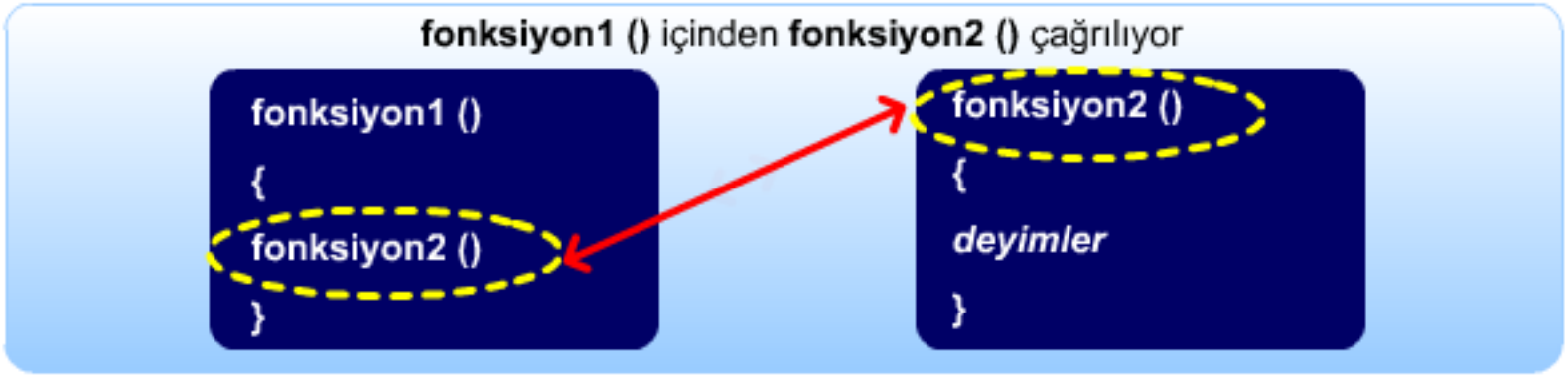
- Fonksiyonlar karmaşık yapılı programların karmaşıklığını azaltmak ve bu programları modüler bir yapıya kavuşturmak için kullanılırlar.
- Fonksiyonlar, programcıların tekrarlanan kodlar yazmalarını önlerler.
- Fonksiyon belirli bir adı olan program parçasıdır.
- Fonksiyonların çalışabilmesi için bir başka fonksiyondan adı ile çağrılması gerekmektedir.

C ve Fonksiyonlar

- C programları fonksiyonlardan oluşurlar.
- Şu ana dek kullandığımız main() de bir fonksiyondur. Bu fonksiyonun bir başka fonksiyon içinden çağrılmasına gerek yoktur.
- Her C programında bir main() fonksiyonun yer alması gerekmektedir.
- main() fonksiyonu, program çalıştırıldığında otomatik olarak çağrılan bir fonksiyondur.
- Bir main() fonksiyonu içinden bir başka fonksiyon çağrılabilir.

C ve Fonksiyonlar (devam…)

- Bir fonksiyon içinden bir başka fonksiyon çağrılabilir.
- Örneğin, **fonksiyon1()** isimli fonksiyondan **fonksiyon2()** isimli bir başka fonksiyon çağrılabilir.

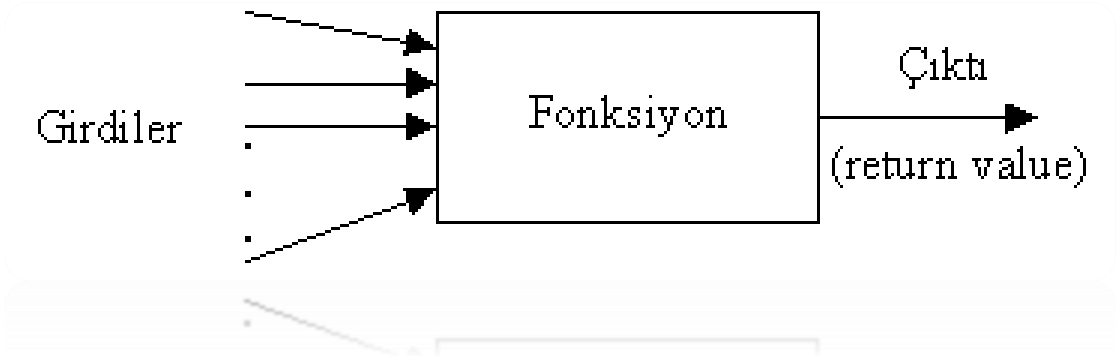


Fonksiyon Tanımı

- Fonksiyon, belirli sayıda verileri kullanarak bunları işleyen ve bir sonuç üreten komut grubudur.
- Her fonksiyonun bir **adı** ve fonksiyona gelen değerleri gösteren **parametreleri** (bağımsız değişkenleri) vardır.
- Bir fonksiyon bu parametreleri alıp çeşitli işlemlere tabi tutar ve bir değer hesaplar.

Fonksiyon Tanımı (devam…)

- Bu değer, çıktı veya geri dönüş değeri (**return** value) olarak adlandırılır.
- Bir fonksiyonun kaç girişi olursa olsun **sadece bir çıkışı** vardır.



Örnek 1: İki sayının toplamı

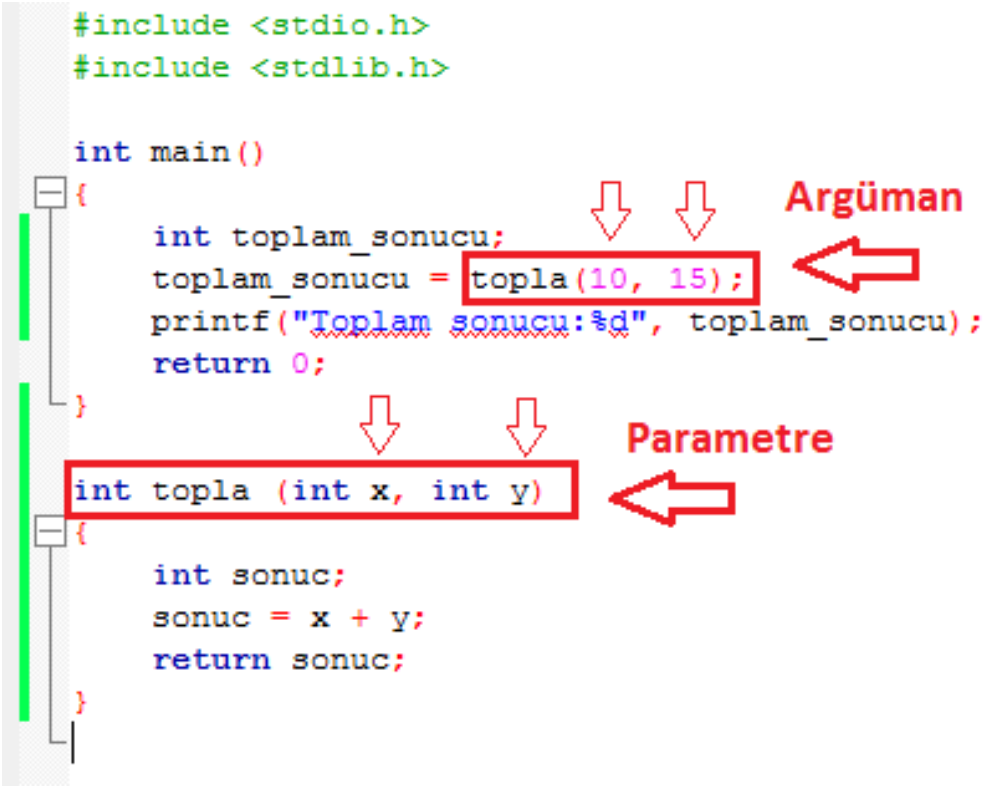
- Fonksiyon tipi: **int**
- Fonksiyon adı: **topla**
- Parametreler: **x** ve **y**
- Geri dönüş değeri: **x+y**

Örnek 1: İki sayının toplamı (devam...)


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

int main()
{
    int toplam_sonucu;
    toplam_sonucu = topla(10, 15);
    printf("Toplam sonucu:%d", toplam_sonucu);
    return 0;
}

int topla (int x, int y)
{
    int sonuc;
    sonuc = x + y;
    return sonuc;
}
```

Parametre ve Argüman

- Fonksiyon çağrılırken gönderilen değerlere **Argüman** denir.
- Fonksiyon bildiriminde, fonksiyona girdi olarak, kullanılan değişkenlere **Parametre** denir.



Fonksiyon Bildirim Örnekleri

Örnek	Açıklama
`int islem();`	Tam sayı değer dönen ve parametre içermeyen bir fonksiyon
`int islem(void);`	Tam sayı değer dönen ve parametre içermeyen bir fonksiyon
`int islem(int x);`	Tam sayı değer dönen ve tam sayı türünde parametre girdisi olan bir fonksiyon
`void islem();`	Değer dönmeyen ve parametre girdisi olmayan bir fonksiyon
`void islem(int x);`	Değer dönmeyen ve tam sayı türünde parametre girdisi olan bir fonksiyon

Örnek 2: Fonksiyon çağırımı ve kod akışı


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

int main()
{
    mesaj();
    printf(" ve Pogramlama \n");
    return 0;
}

void mesaj()
{
    printf("Algoritma");
}
```

Hadi debug edelim...

Örnek 3: İç içe birden fazla fonksiyon çağırımı


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

int main()
{
    mesaj();
    printf(" Hos geldiniz...\n");
    return 0;
}

void mesaj()
{
    printf("Algoritma");
    mesaj2();
}

void mesaj2()
{
    printf(" ve Programlama");
}
```

```
Algoritma ve Programlama Hos geldiniz...
```

Fonksiyon Geri Dönüş Değerleri

- Fonksiyon geri dönüş değeri **return** anahtar sözcüğü kullanılarak gerçekleştirilir.
- **return** anahtar sözcüğünün iki önemli işlevi vardır:
  1. fonksiyonun geri dönüş değerini oluşturur
  2. fonksiyonu sonlandırır
- **return** deyiminden sonra bir değişken, işlem, sabit veya başka bir fonksiyon yazılabilir.

Fonksiyon Geri Dönüş Değerleri (devam..)

- Örnekler:
  - **return** (a+b/c); /* parantez kullanmak zorunlu değil */
  - **return** 10; /* değişken kullanmak mecbur değil */
  - **return** topla(a, b)/2.0; /* önce topla fonksiyonu çalışır */

- Programın çözüm mantığına göre bir fonksiyon içerisinde birden çok geri dönüş değeri kullanılabilir.
- Fakat, ilk karşılaşılan return deyiminden sonra fonksiyon sonlanır ve çağrılan yere bu değer gönderilir.

Fonksiyon Prototip Kullanımı

- Bir fonksiyon prototipi, derleyiciye fonksiyon tarafından döndürülen verinin tipini, fonksiyonun almayı beklediği parametre sayısını, parametrelerin tiplerini ve parametrelerin sırasını bildirir.
- Derleyici, fonksiyonların ilk hallerini (prototiplerini) fonksiyon çağrılarını onaylamakta kullanır.
- Fonksiyon prototipinin unutulması, fonksiyonun geri dönüş tipinin int olmadığı durumda ve fonksiyon tanımı fonksiyon çağrısından daha sonra bulunmuyorsa, yazım hatalarına (syntax error) sebep olur.

Örnek 4: Fonksiyon geri değer dönüşü ve prototip kullanımı


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

char Notu_Harfe_Donustur (int ogrenci_notu);

int main()
{
    printf("Not: 75, Harf: %c \n", Notu_Harfe_Donustur(75));
    printf("Not: 56, Harf: %c \n", Notu_Harfe_Donustur(56));
    return 0;
}

char Notu_Harfe_Donustur (int ogrenci_notu)
{
    if( ogrenci_notu>=0  && ogrenci_notu<50 ) return 'F';
    if( ogrenci_notu>=50 && ogrenci_notu<70 ) return 'D';
    if( ogrenci_notu>=70 && ogrenci_notu<80 ) return 'C';
    if( ogrenci_notu>=80 && ogrenci_notu<90 ) return 'B';
    if( ogrenci_notu>=90)                     return 'A';
}
```

prototip

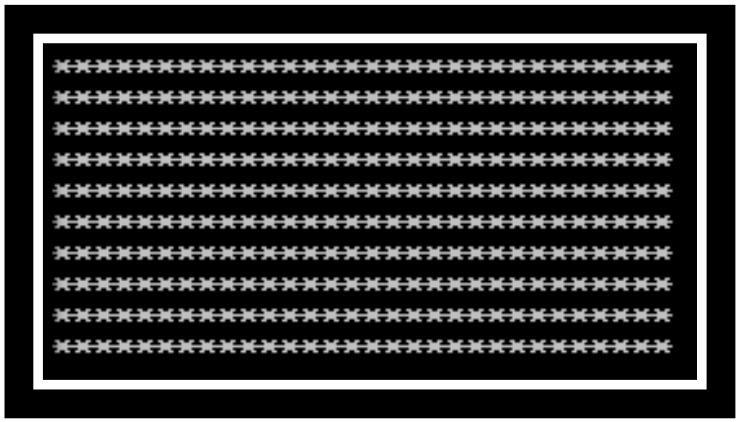
```
Not: 75, Harf: C
Not: 56, Harf: D
```

void Fonksiyonlar

- Bir fonksiyonun her zaman geri dönüş değerinin olması gerekmez.
- Bu durumda **return** deyimi kullanılmayabilir. Eğer bu anahtar kelime yoksa, fonksiyon ana bloğu bitince kendiliğinden sonlanır.
- Böyle fonksiyonların tipi **void** (boş, hükümsüz) olarak belirtilmelidir.
- Bu tip fonksiyonlar başka bir yerde kullanılırken, herhangi bir değişkene atanması söz konusu değildir, çünkü geri dönüş değeri yoktur. Ancak, void fonksiyonlara parametre aktarımı yapmak mümkündür.

Örnek 5: void Fonksiyon Örneği


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

int main()
{
    kutu_ciz(10, 30);
    return 0;
}

void kutu_ciz( int satir, int sutun )
{
    int j;
    for ( ; satir > 0; satir--){

        for (j = sutun; j > 0; j--)
          printf("*");

        printf("\n");

     }
}
```



Fonksiyon Kullanım Hataları

1. Fonksiyon tanımlamalarında geri dönüş değerini unutmak.
2. Geri dönüş tipi void olarak bildirilmiş bir fonksiyonun bir değer geri döndürmesi bir yazım hatasıdır.
3. Aynı tipte fonksiyon parametrelerini ***double*** *x*, ***double*** *y* yerine ***double*** *x,y* olarak bildirmek. double x, y biçiminde parametre bildirmek, ***y parametresinin tipinin int olmasına*** sebep olur. Çünkü belirtilmeyen parametre tipi otomatik olarak int tipinde varsayılır

Fonksiyon Kullanım Hataları (devam…)

5. Parametre listesini yazdığımız parantezlerin dışına noktalı virgül koymak yazım hatasıdır.
6. Bir fonksiyon parametresini daha sonradan fonksiyon içinde yerel bir değişken olarak kullanmak bir yazım hatasıdır.
7. Bir fonksiyon içinde başka bir fonksiyon tanımlamak yazım hatasıdır.
8. Fonksiyon prototipinin sonuna noktalı virgül koymamak bir yazım hatasıdır.

Fonksiyonların Yinelemesi - Kendi Kendilerini Çağırması (Recursive Function)

- Bazı problem tipleri için fonksiyonların kendi kendilerini çağırması kullanışlı olabilir.
- Bir yineleme fonksiyonu (recursive function), kendi kendini doğrudan ya da bir başka fonksiyon içinden çağıran fonksiyondur.
- Yineleme fonksiyonu, bir problemi çözmek için çağrılır.
- Bu fonksiyon, yalnızca en basit durumu ya da temel durum olarak adlandırılan durumu nasıl çözeceğini bilmektedir.
- Eğer fonksiyon temel bir durumla çağrılırsa, fonksiyon bir sonuç geri döndürür.

Fonksiyonların Yinelemesi - Kendi Kendilerini Çağırması (Recursive Function) devam…

- Yinelemeyi mümkün kılmak için sonraki parça orijinal probleme benzemelidir, fakat orijinal problemin daha basit ya da daha küçük bir versiyonu olmalıdır.
- Yineleme için verilebilecek en güzel örnek popüler bir matematik konusu olan Negatif olmayan bir n tamsayısının faktöriyelinin hesaplanmasıdır.
- **Örnek:**

  5!=5*4*3*2*1

  5!=5*(4*3*2*1)

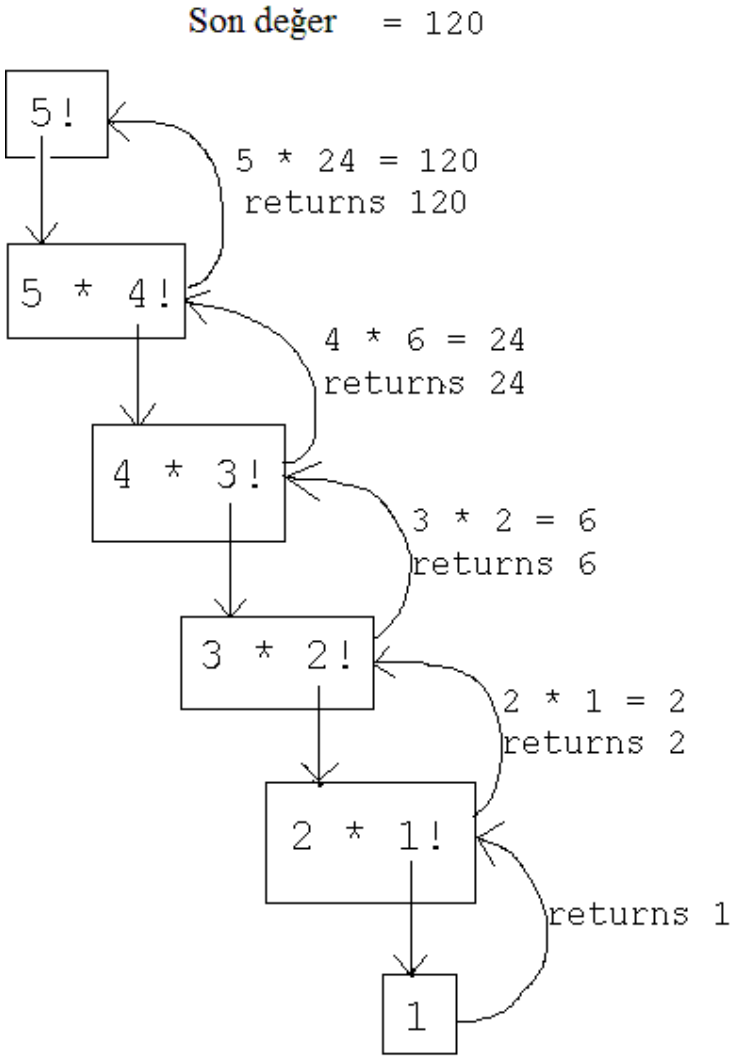
  5!=5*(4!)

Örnek 6: Yinelemeli Fonksiyon Kullanarak Yapılan Faktöriyel Hesabı


```c
#include <stdlib.h>

int faktoryel_hesapla(int sayi);

int main()
{
    int sayi;
    printf("Faktoryel hesabi icin sayi giriniz:");
    scanf("%d", &sayi);

    printf("Faktoryel sonucu: %d \n", faktoryel_hesapla(sayi));
    return 0;
}

int faktoryel_hesapla(int sayi)
{
    if (sayi <= 1)
        return 1;
    else
        return (sayi * faktoryel_hesapla(sayi-1)); //yineleme kismi
}
```

```
Faktoryel hesabi icin sayi giriniz:5
Faktoryel sonucu: 120
```

Örnek 6: Yinelemeli Fonksiyon Kullanarak Yapılan Faktöriyel Hesabı (devam…)



Yineleme mi Tekrar mı?

- Tekrar ve yinelemenin ikisi de döngü içerir.
- Tekrar özellikle döngü yapısını kullanırken, yineleme döngüyü fonksiyon çağrılarının tekrarında kullanır.
- Tekrar ve yinelemenin ikisi de bir sonlandırma testi içerirler.
- Yineleme temel bir durumla karşılaşıldığında, tekrar ise döngü devam koşulu yanlış hale geldiğinde sona erer.
- Yineleme bir çok negatif özelliğe sahiptir.
- Yineleme, mekanizmayı sürekli çağırarak fonksiyon çağrılarının artmasına sebep olur. Bu, işlemci zamanı ve hafızada fazladan yük demektir.

Yineleme mi Tekrar mı? (devam…)

- Her yineleme çağrısı, fonksiyonun başka bir kopyasının oluşmasına sebep olur, bu da hafızayı fazladan işgal etmek demektir.
- Tekrar, fonksiyonların sürekli olarak çağrılması ve fazladan hafıza kullanılması engellenir.
- Yinelemeli olarak çözülen her problem tekrarlı bir biçimde çözülebilir. Yineleme yaklaşımı genelde problemi daha iyi yansıttığı ve daha kolay anlaşılan ve hataları kolay ayıklanan programlar yazılmasını sağlattığı için, tekrar yaklaşımına göre tercih edilebilir.
- Yinelemeli çözümleri seçmenin başka bir sebebi de tekrarlı çözümün kolaylıkla bulunamayışıdır.

Dizilerin Fonksiyonlarda Kullanılması

- Diziler de sıradan değişkenler gibi bir fonksiyona parametre olarak aktarılabilirler.
- Fakat, aktarma kuralı biraz farklıdır.

Her zaman dizinin yanında

boyutunun da bilinmesi gereklidir !!!

Örnek 7: Dizinin Fonksiyonda Kullanılması


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

void dizi_yaz(float x[], int n);

int main(){
   float dizi[5]= { 8.471, 3.683, 9.107, 4.739, 3.918 };
   dizi_yaz(dizi, 5);
   return 0;
}

void dizi_yaz(float x[], int n)
{
   int i;
   for(i=0; i<n; i++)
      printf("%.2f \n", x[i]);
   printf("\n");
}
```

```
8.47
3.68
9.11
4.74
3.92
```

Tavsiyeler

- ✓ Birden fazla fonksiyon kullanılan programlarda, **main** fonksiyonu programın esas görevini yerine getiren fonksiyonların çağırıcısı olarak kullanılmalıdır.
- ✓ Her fonksiyon, iyi olarak tanımlanmış tek bir işi yapacak şekilde sınırlandırılmalıdır ve fonksiyon ismi, fonksiyonun görevini etkili bir biçimde açıklamalıdır. Bu, özetlemeyi ve yazılımın yeniden kullanılabilirliğini sağlatır.
- ✓ Eğer fonksiyonun görevini açıklayacak etkili bir isim bulamıyorsanız muhtemelen yazdığınız fonksiyon birden fazla görevi yerine getirmeye çalışmaktadır. Bu tarzda fonksiyonları daha küçük fonksiyonlara bölmek en iyi yoldur.

Tavsiyeler (devam…)

- ✓ Bir fonksiyon genellikle bir sayfadan daha uzun olmamalıdır. Küçük fonksiyonlar yazılımın yeniden kullanılabilmesini sağlatır.
- ✓ Programlar, küçük fonksiyonların bir araya getirilmesiyle yazılmalıdır. Bu, programların daha kolay yazılması, değiştirilmesi ve hatalarının giderilmesini sağlar.
- ✓ Çok fazla sayıda parametreye ihtiyaç duyan fonksiyonlar birden fazla görevi yerine getiriyor olabilir. Böyle fonksiyonları ayrı görevleri gerçekleştiren daha küçük fonksiyonlara bölmek gerekir. Fonksiyonun başlığı mümkünse bir satıra sığmalıdır.

KAYNAKLAR

- N. Ercil Çağıltay ve ark., C DERSİ PROGRAMLAMAYA GİRİŞ, Ada Matbaacılık, ANKARA; 2009.
- Milli Eğitim Bakanlığı "Programlamaya Giriş ve Algoritmalar Ders Notları", 2007
- http://tr.wikipedia.org/wiki/Code::Blocks
- http://www.codeblocks.org
- http://www.AlgoritmaveProgramlama.com
- http://www1.gantep.edu.tr/~bingul/c

İYİ ÇALIŞMALAR…

Yrd. Doç. Dr. Deniz KILINÇ
deniz.kilinc@cbu.edu.tr