Fonksiyon Kavramı

Kendiniz veya bir arkadaşınız bilgisayarda çalışırken zaman zaman sürekli olarak tekrarlanan bir iş ile karşı karşıya gelmiştir. Genellikle bir çok kişinin işlerinin büyük bir kısmı rutin işlerin tekrarlanmasından ibarettir. Bir çok işyeri çalışanının, örneğin excel programını kullanarak günlük, haftalık, aylık dönemlerde aynı işleri aynı şekilde yapması çalışma hayatında sıklıkla görülen bir durumdur. Benzer durumlar sadece excel kullananlar için değil aynı zamanda programlama işi ile uğraşanlar için de geçerlidir. Aynı bir iş için aynı kodlar tekrar tekrar yazılıyor ya da daha önceden yazılmış olan kodlar kopyala-yapıştır işlemi ile alınıp küçük değişikler yapılarak yeniden kullanılıyordur.

Her hangi bir rutin dönmüş, aynı şekilde tekrar tekrar yapılıyorsa bunun en doğru yaklaşımı bu işi otomatik hale getirmektir. Mesela metin dosyalarını ya da excel tablolarını açıp içlerindeki belirli bilgileri kullanarak bazı hesaplamalar yapabilir veya bunları rapor haline getirebilirsiniz. Bu işlemleri her defasında tekrarlamak yerine bu işlemeleri yapan otomatik kodlar hazırlanabilir ve işlemlerin daha kısa sürede hatasız olarak tamamlanması sağlanabilir.

Programlamada en önemli nokta, eğer bir iş ikiden fazla tekrarlanıyorsa bu işlemi yapan kısa bir kod yazmaktır. Bu kısa kodlara fonksiyon ismi verilmektedir. Programlamadaki fonksiyon kavramı ile matematikteki fonksiyon kavramı birbirine benzerdir. Matematiksel olarak fonksiyon, bir A kümesinin her elemanını bir B kümesinin bir ve sadece bir elemanı ile eşleştiren ilişki olarak tanımlanır. Bir başka deyişle bir fonksiyon aldığı her bir girdi için sadece tek bir çıktı üreten bir ifadedir. Programlamada da aynı mantık söz konusudur. Bir fonksiyon kendisine girdi olarak verilen parametreleri kullanarak elde edilen çıktıyı iletir.

Fonksiyon Kavramı

Fonksiyonların kolaylıkla oluşturularak verimli bir şekilde kullanılabilmesi, C dilinin en önemli özelliklerinden biridir. Buradaki fonksiyon kelimesi matematiksel anlamıyla değil, çoğu programlama dillerinde kullanılan anlamıyla, "alt program", "prosedür", "subroutine" sözcüklerinin karşılığı olarak kullanılmaktadır. Bu şekilde tanımlanan bir fonksiyon gerekli oldukça çağrılır ve fonksiyonun çağrıldığı program içinde aynı kodun tekrarı önlenmiş olur. Tanımlanan fonksiyon sadece bir kez çağrılacak olsa da genel programdan ayrıldığı için ana program kodunun kısalmasını sağlar. Bu ise ana programa üzerinde daha kolay bir kontrol sağlanmasına imkan verir.

```
void main() {
    .
    . işlemler_1(a);
    .

    .
    . işlemler_1(b);
    .

    .
    . işlemler 2;
    .

}
```

```
void main() {
    .
    fonksiyon(a);
    fonksiyon(b);
    .
    . işlemler 2;
    .

}
```

```
int fonksiyon(x) {
    .
    . işlemler (x);
    .
}
```

Program yazımında, büyük bir programın tümünü tek bir main bloğu içine yazmak pek iyi bir yol değildir. Fonksiyonlar oluşturularak geniş kapsamlı bir program daha küçük parçalara ayrılabilir. Böylece ana programın esas işlevi ile ilgili olmayan detaylar ana programdan ayrılarak daha kolay kontrol sağlanabilir. Daha önceden yazılarak fonksiyon haline getirilmiş kodlar, bir başka programda aynı işleve ihtiyaç duyulduğunda, ilgili kodları yeniden yazmaya gerek kalmadan kullanılabilir.

Fonksiyon tanımı

Bir fonksiyon tanımlamak için ilk önce geri dönen parametreye göre değer türü belirtilir ve ardından fonksiyonun adı yazılır. Fonksiyonun görevini yerine getirecek kod bloğu, tanım satırından süslü parantezler "{ }" içinde yazılır. Fonksiyonlar, fonksiyon ismiyle birlikte aç-kapa parantezler ve varsa parametreler verilerek çağrılır.

```
dönüş tipi  fonksiyon ismi ( parametre listesi )
{
  .
  . fonksiyon işlemleri;
  .
  return değerler
}
```

Her fonksiyon tanımında parantezler içinde parametreler olması zorunlu değildir. **"fonksiyon_ismi(void)"** şeklinde kullanım fonksiyonun her hangi bir girdi parametresi olmadığı anlamına gelir. Fonksiyonların geri dönüş değeri ya da değerleri **"return"** kelimesiyle sağlanır ve **"return parametre"** şeklinde kullanılır. Geri dönen parametre kullanım amacına uygun her hangi bir değişken değerden oluşabilir.

Genel gösterimden de anlaşılacağı üzere fonksiyon tanım iki kısımdan oluşur:

- **fonksiyon başlığı ve**
- **fonksiyon gövdesi**

Fonksiyonun döndüreceği değerin tipi başlık kısmında fonksiyon isminin öncesine yerleştirilerek belirlenir. Böylece fonksiyonun döndüreceği değerin tipi önceden belirtilmiş olunur. C'de eğer her hangi bir tip belirleyici belirtilmez ise bu fonksiyonun **"int"** tipinde değerler döndüreceği anlamına gelir. Hiç bir değer döndürmeyen fonksiyonlar ise **"void"** tipi ile tanımlanır. Seçilen fonksiyon isminin fonksiyonun genel işlevine uygun bir şekilde açıkça belirtilmesi fonksiyonun kolay anlaşılmasını sağlayacaktır.

Bir fonksiyon örneği

Terminalden girilen pozitif bir tamsayıya kadar olan sayıların toplamını bulan bir programı göz önüne alalım.

```
1   #include<stdio.h>
2   int topla (int a);

3   int main(void)
4      {   int x, sonuc;
5          printf("Bir tamsayi giriniz:");
6          scanf("%d", &x);
7          sonuc = topla(x);
8          printf("Toplam=%d", sonuc);
9          return 0;
10     }
```

```
11  /* Fonksiyon tanımı */
12  int topla(int a)
13     {   int i,s;
14         s=0;
15         for(i=l; i<=a; ++i)
16             s=s+i;
17         return s;
18     }
```

Program incelediğimizde görüyoruz ki program **main()** ve **topla()** şeklinde iki fonksiyondan oluşmaktadır. Bilindiği üzere herhangi bir C programı her zaman **main()** fonksiyonu ile başlar ve bütün kontrol bu fonksiyondadır. Çünkü program her zaman otomatik olarak bu fonksiyondan başlatılır. **main()** fonksiyonu yürütülürken diğer fonksiyonları çağırır, onların yürütülmesini sağlar. Her bir fonksiyon çağrıldığı yere döndürülerek işlemlere devam edilir.

Program çalıştırıldığında öncelikle 4. ve 6. satırlar arasında tanımlanan komutlar yürütülür. Daha sonra 7. satıra gelindiğinde **topla()** fonksiyonu çağrılır. Bu aşamada kontrol 12.satırda tanımlanmış olan **topla()** fonksiyonuna geçer ve topla fonksiyonunun tüm komutları sırasıyla yürütülür. 18.satıra ulaşınca, kontrol tekrar **main ()** fonksiyonunun 7. satırına geri döner ve kalan komutlar 9.satır sonuna kadar yürütülür. Dolayısıyla bu programda main() çağıran fonksiyon, topla() çağırılan fonksiyon olarak adlandırılır.

Fonksiyon prototipi

C'nin önemli bir özelliği fonksiyonların da değişkenler gibi main() fonksiyonundan önce tanımlanmalarıdır. Bu nedenle bir fonksiyon önce **Bildirim Fonksiyonu** olarak bildirilmesi gereklidir. Bunu için bildirim fonksiyonun ismi, döndüreceği değerin tipi, çağrı ile aktarılan verilerin sayısı ve tipi belirtilmelidir. Bu bildirime **fonksiyon prototipi** ya da kısaca **prototip** adı verilmektedir.

```
Döndürme tipi  Fonksiyon ismi ( parametre listesi );
```

C'de bir programda tanımlanan her bir fonksiyon için tek bir prototip bulunması gereklidir. Prototipler, genel olarak ana programın başlangıç kısmına yakın ve bildirim yapılan fonksiyonun ilk kullanımından önce yerleştirilmelidir. Böylece derleyiciye programdaki çağrıları kontrol etme imkanı sağlanmış olur. Prototipte fonksiyon parametreleri parantezler arasında virgülle ayrılmış veri tip listesi olarak verilir. Veri tipleri sıralanırken parametre isimleri kullanmak zorunlu değildir ancak konulması daha uygundur. Ayrıca fonksiyon başlığında yer alan parametre listesinin aynısı prototipte de kullanılabilir. Eğer fonksiyonun hiç bir parametresi yoksa prototipte de void veri tipi kullanılır.

```
döndürme tipi fonksiyon_ismi ( void );
```

Eğer fonksiyon hiçbir değer döndürmüyor ise prototipte de fonksiyon parametre tanımında olduğu gibi void tipi belirleyici kullanılmalıdır, tip verilmediğinde ise "int" türü değerler döndürüleceği kabul edilir.

Önemli not:

Bir fonksiyona aktarılan sabit değerlere ya da değişkenlere **argüman**, **gerçek argüman** ya da **gerçek parametre** adı verilir. Ancak fonksiyon prototipinde yani bildirimde parantezler içinde bildirilen değişkenlere ise **parametre**, **formal parametre** ya da **formal argüman** adı verilir. Daha basit bir ifade ile esas fonksiyonda yer alan değerlere ya da değişkenlere **argüman**, bildirimde verilen değişkenlere ise **parametre** adı verilmektedir.

Fonksiyon prototip örneği

Fonksiyon prototipi kullanımına başka bir örnek şu şekilde verilebilir.

```
#include<stdio.h>
int carp (int);              /* Fonksiyonun prototipi */

int main(void)
   {   int a, sonuc;
       printf("Bir sayı giriniz:");
       scanf("%d", &x);      /* Fonksiyona a değeri gönderilir */
       sonuç = carp(a);
       printf("%d", sonuç);  /* Geri donen sonuç görüntülenir */
       return 0;
   }

int carp(int X)
   {
       return X*X*X*X; /* Fonksiyon a değerini 4 kez çarpar ve değeri geri dondurulur */
   }
```

Burada dikkat edilmesi gereken önemli nokta prototip ile fonksiyonun parametrelerinde farklı isimler kullanılmış olmasıdır. Çoğunlukla bir prototipte ve fonksiyonda aynı parametre isimlerinin kullanılması tercih edilir. Fakat bu bir zorunluluk değildir. Ayrıca prototipte bir parametre isminin kullanılması da gerekli değildir.

Fonksiyonların sınıflandırılması

Fonksiyonlar kullanım amacına ve şekline göre farklı görevlerin yerine getirilmesi için büyük bir esneklik sağlar. Örneğin bir hesaplama sonucu gibi değerleri ana programa aktarılması için kullanılırken başka bir uygulamada da aynı işlemi farklı veriler üzerinde uygulamak için yeniden kullanılabilir. Bu açıdan fonksiyonlar genel şu şekilde gruplanabilir:

a) Dönüş değeri olan fonksiyonlar
b) Dönüş değeri olmayan fonksiyonlar
c) Parametre alan fonksiyonlar
d) Parametre almayan fonksiyonlar

a) Dönüş değeri olan fonksiyonlar

Belirli bir işlem veya hesaplama sonucunu ana programa geri döndüren fonksiyonlar olarak tanımlanır. Bu fonksiyonlar, bir işlemi gerçekleştirdikten sonra bir değer döndürmek için kullanılır ve dönüş türü, fonksiyonun başında belirtilir (örneğin, int, float, char). Geri döndürülen bu değer, **return** ifadesiyle sağlanır ve fonksiyonun çağrıldığı yere de değer iletilir.

```c
#include<stdio.h>
int topla (int a, int b);

int main(void) {
   int sonuç = topla(5, 3);
   printf("Taplam: %d\n", sonuc);
   // terminale "Toplam: 8" yazar
   return 0;
}
```

```c
// İki sayıyı toplayan fonksiyon
int topla(int a, int b) {
    return a + b;
   }
```

Dönüş değeri olan fonksiyonlar – başka bir örnek

Bir fonksiyonun geri dönüş değeri çoğunlukla sayısal ya da metin değişkenleri olabilmektedir. Bazı durumlarda geri dönüş değeri başka bir fonksiyon olabilir. Bu durumda çağrılan fonksiyonun geri dönüş değeri çağıran fonksiyonun geri dönüş değeri olmaktadır. Aşağıdaki örnek bu durumun basit bir şeklidir.

```c
#include <stdio.h>
#include <math.h>            // Kare kök fonksiyonu için gerekli kütüphane

// İki sayının aritmetik ortalamasını hesaplayan fonksiyon
float aritmetikOrtalama(float a, float b) {
    return (a + b) / 2.0;
}

// Aritmetik ortalamanın karekökünü hesaplayan fonksiyon
float ortalamaKarekok(float x, float y) {
    // Dönüş değeri başka bir fonksiyon olan satır
    return sqrt(aritmetikOrtalama(x, y));
}

int main() {
    float sayi1 = 9.0;
    float sayi2 = 16.0;

    float sonuc = ortalamaKarekok(sayi1, sayi2); // Fonksiyonu çağır ve sonucu al

    printf("%.2f ve %.2f sayılarının aritmetik ortalamasının karekökü: %.2f\n", sayi1, sayi2, sonuc);
    // Ekrana "9.00 ve 16.00 sayılarının aritmetik ortalamasının karekökü: 3.87" yazar

    return 0;
}
```

b) Dönüş değeri olmayan fonksiyonlar

Herhangi bir değer döndürmeyen ve **void** türü ile tanımlanan fonksiyonlardır. Bu fonksiyonlar, belirli bir işlemi gerçekleştirmek için kullanılır ve işlem sonucu olmadığında her hangi bir değer döndürmezler. Genellikle, ekran çıktısı verme, dosya okuma/yazma veya bir dizi veriyi işleme gibi yan etkisi olan işlemler için kullanılırlar. Bu fonksiyonlar, işlem tamamlandığında sadece tanımlanan görevleri yerine getirir ve **return** ifadesi kullanılmaz (ya da sadece fonksiyondan çıkmak için kullanılır).

```c
#include<stdio.h>
// Ekrana hata mesajı yazdıran fonksiyon
void uyariYaz() {
    printf("Bölüm Hatası: Payda sıfır olamaz!\n")
}

int main() {
   int a=5, b=2; double c;
   if(b != 0){
      c = a / b;
   }else{
      uyariYaz(); // Fonksiyonu çağırır ve mesajı yazar
   }
   return 0;
}
```

Fonksiyon Tanımı: void uyariYaz(), parametre almaz ve dönüş değeri yoktur.

Fonksiyon Kullanımı: uyariYaz() çağrısı ile ekrana mesaj yazdırılır.

```c
#include<stdio.h>
// Dönüş değeri olmayan bir fonksiyon
void selamVer() {
    printf("Merhaba, hoş geldiniz!\n")
}

int main() {
   // Fonksiyon çağrısı
   selamVer(); // Aynı fonksiyonu
   selamVer(); // tekrar tekrar çağırabiliriz
   return 0;
}
```

void selamVer(): Bu fonksiyonun dönüş tipi void, yani hiçbir değer döndürmez.

Fonksiyon Gövdesi: Fonksiyon, sadece ekrana "Merhaba, hoş geldiniz!" mesajını yazdırır.

Fonksiyon Çağrısı: selamVer() fonksiyonu, main içinde iki kez çağrılır ve her çağrıldığında aynı mesajı ekrana yazar.

c) Parametre alan fonksiyonlar

Belirli bir işlemi gerçekleştirmek için dışarıdan veri alabilen fonksiyonlardır. Bu fonksiyonlar, parametre listesi olarak belirtilen değişkenler aracılığıyla çağrıldıkları yere bağlı olarak farklı girdileri işleyebilir. Her parametre, fonksiyona bir değer veya referans olarak iletilir ve fonksiyon bu verilere göre işlemler yapar. Parametrelerin türü, fonksiyonun başında tanımlanır ve işlevi belirler (örneğin, int, float, char). Bu fonksiyonlar, matematiksel hesaplamalar, veri işleme ve belirli kontroller gibi birçok işlem için kullanılır ve bu da kodun daha dinamik ve modüler olmasına yardımcı olur.

```c
#include<stdio.h>
// Dikdörtgenin alanını hesaplayan fonksiyon
float alanHesapla(float uzunluk, float genislik) {
    return uzunluk * genislik;
}

int main() {
   float uzunluk = 5.5;
   float genislik = 3.2;
   float alan = alanHesapla(uzunluk, genislik); // Fonksiyonu çağır ve alanı hesapla
   printf("Dikdörtgenin Alanı: %.2f\n", alan);  // Terminale "Dikdörtgenin Alanı: 17.60" yazar
   return 0;
}
```

Burada **alanHesapla** fonksiyonu, bir dikdörtgenin uzunluk ve genişlik değerlerini parametre olarak almaktadır. Verilen parametrelere göre fonksiyon, dikdörtgenin alanını hesaplar ve sonucu geri döndürür. **main** fonksiyonunda ise dikdörtgenin uzunluk ve genişlik parametreleri **alanHesapla** fonksiyonuna aktarılarak dikdörtgenin alanı hesaplanır ve sonuç ekrana yazdırılır.

Parametre alan fonksiyonlar – başka örnek

Bu örnekte, bir sıcaklık değeri alınır ve belirli aralıklara göre bir uyarı mesajı verilir. Örneğin, sıcaklık çok yüksekse **"Çok sıcak!"**, düşükse **"Çok soğuk!"** gibi uyarılar gösterilir.

```c
#include<stdio.h>
// Sıcaklık değeri üzerinden uyarı veren fonksiyon
void sicaklikUyari(float sicaklik) {
    if (sicaklik > 30.0) {
        printf("Uyarı: Çok sıcak!\n");
    } else if (sicaklik < 5.0) {
        printf("Uyarı: Çok soğuk!\n");
    } else {
        printf("Hava sıcaklığı normal.\n");
    }
}

int main() {
   float sicaklik;

   printf("Sıcaklık değerini girin (C): ");
   scanf("%f", &sicaklik);

   sicaklikUyari(sicaklik); // Fonksiyonu çağır ve uyarıyı al
   return 0;
}
```

Burada **sicaklikUyari** verilen sıcaklık değerine göre bir uyarı mesajı yazdırır. Eğer sıcaklık 30°C'nin üzerindeyse "Çok sıcak!", 5°C'nin altındaysa "Çok soğuk!" mesajı verir. Aksi takdirde, sıcaklık normal kabul edilir. main fonksiyonunda kullanıcıdan bir sıcaklık değeri alınır ve **sicaklikUyari** fonksiyonu çağrılarak uygun uyarı mesajı ekrana yazdırılır.

Parametre alan fonksiyonlar – başka bir örnek

Hemen hemen herkesin bildiği Kelvin, Celsius ve Fahrenheit sıcaklık ölçüleri arasındaki dönüşüm formülleri düşünelim. Bu sıcaklık dereceleri belirli bağıntılarla birbirine dönüştürülebilmektedir. Bu bağıntılar;

```
celsius = kelvin - 273
fahrenheit = 1.8 * celcius + 32
```

olarak verilmektedir. Şimdi Kelvin cinsinden verilen bir sıcaklığı Fahrenheit'e çevirecek bir fonksiyonu oluşturalım. Yukarıdaki bağıntıları dikkate aldığımızda görüleceği üzere Kelvin'den Fahrenheit'e direkt bir çevirim söz konusu değildir. İlk adımda Kelvin Celsius'a ardından Celsius Fahrenheit'e çevrilme işlemlerinin yapılması gerekmektedir.

```c
#include<stdio.h>
// alt fonksiyon
int kelvin_celsius(int kelvin_sicaklik){
    return (kelvin_sicaklik - 273);
}

// ana fonksiyon
float kelvin_fahrenheit (int sicaklik) {
    return (1.8 * kelvin_celsius(sicaklik) + 32);
}
```

```c
int main() {
    int kelvin;
    float fahrenheit;
    printf("Kelvin sıcaklığı giriniz : ");
    scanf("%d", &kelvin);

    fahrenheit = kelvin_Fahrenheit(kelvin);
    printf("Fahrenheit sıcaklığı: %.1f", fahrenheit);
    return 0;
}
```

Kod çalıştırılıp kelvin derecesi olarak **303** değeri girildiğinde, "86.0" fahrenheit derecesi değerini elde ederiz.

d) Parametre almayan fonksiyonlar

Dışarıdan herhangi bir veri almadan belirli bir işlemi gerçekleştiren fonksiyonlardır. Bu fonksiyonlar, parantez içinde boş bir parametre listesi ile tanımlanır ve genellikle sabit bir işlemi yerine getirmek için kullanılırlar. Parametre almadıkları için her çağrıldıklarında aynı işlemi gerçekleştirirler ve dışarıdan gelen verilere bağlı değildirler. Genellikle programın belirli bir sabit görevini yerine getirmek, sabit bir mesaj yazdırmak veya durum kontrolü yapmak gibi işlemler için kullanılır.

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

// Mevsime göre mesaj veren fonksiyon
void mevsimMesaji() {
    int ay;
    printf("Hangi aydayız (1-12) : ");
    scanf("%d", &ay);

    // Mevsime göre farklı mesajlar
    if (ay >= 2 && ay <= 4) {
        printf("Mevsim Bahar, artık doğa canlanıyor!\n");
    } else if (ay >= 5 && ay <= 7) {
        printf("Mevsim Yaz, sıcak günler başlıyor!\n");
    } else if (ay >= 8 && ay <= 10) {
        printf("Mevsim Sonbahar, rüzgarlar esiyor.\n");
    } else {
        printf("Mevsim Kış, soğuk günleri başladı.\n");
    }
}
```

```c
int main() {
    // Mevsim mesajını yazdıran fonksiyon çağrısı
    mevsimMesaji();  // Parametresiz fonksiyon çağrısı
    return 0;
}
```

main fonksiyonunda, **mevsimMesaji** fonksiyonu çağrılır ve girilen ay numarasına göre geçerli mevsim için bir mesaj ekrana yazdırılır.

Özyinelemeli (Rekürsif) Fonksiyonlar

Özyinelemeli temelde kendi kendini çağıran fonksiyonlardır. Yani, bir fonksiyon, belirli bir koşul sağlanana kadar kendisini tekrar tekrar çağırır. Özyineleme, özellikle karmaşık problemlerin daha basit alt problemlere bölünerek çözülmesini sağlar. Her özyinelemeli fonksiyonunda bir **temel durum** (base case) ve bir **yenilemeli durum** (recursive case) olması gerekir. Temel durum, fonksiyonun kendi kendini çağırmayı sonlandıracağı koşuldur, bu sayede sonsuz döngülerden kaçınılır.

Özyineleme genellikle, sıralama algoritmaları (örneğin, hızlı sıralama veya birleşim sıralaması), arama algoritmaları (örneğin, ikili arama), ve matematiksel hesaplamalar (örneğin, faktöriyel hesaplama) gibi problemlerin çözümünde yaygın olarak kullanılır.

```c
#include<stdio.h>

// özyinelemeli faktöriyel fonksiyonu
int faktoriyel(int n) {
    if (n == 0) {  // Temel durum
        return 1;
    } else {
        return n * faktoriyel(n - 1);  // yenilemeli durum
    }
}
int main() {
   int sayi;
   printf("Bir sayı girin: ");
   scanf("%d", &sayi);

   printf("Faktöriyel: %d\n", faktoriyel(sayi));
   return 0;
}
```

Özyinelemeli fonksiyonlar – başka örnek

Fibonacci dizisi, her sayının önceki iki sayının toplamı olduğu bir dizidir. Fibonacci dizisinin ilk iki elemanı 0 ve 1'dir. Özyinelemeli bir fonksiyon kullanarak bir Fibonacci dizisinin n adet elemanını yazdıralım.

```c
#include<stdio.h>
// Özyinelemeli fonksiyon ile Fibonacci dizisi hesaplama
int fibonacci(int n) {
    if (n == 0) {  // Durma koşulu: Fibonacci(0) = 0
        return 0;
    } else if (n == 1) {  // Durma koşulu: Fibonacci(1) = 1
        return 1;
    } else {
        return fibonacci(n - 1) + fibonacci(n - 2);  // Özyineleme
    }
}

// Fibonacci dizisinin tüm elemanlarını yazdıran fonksiyon
void fibonacciDizisi(int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", fibonacci(i));  // Fibonacci(i)'yi yazdır
    }
    printf("\n");
}

int main() {
    int n;
    printf("Fibonacci dizisinin eleman sayısını girin: ");
    scanf("%d", &n);  // Kullanıcıdan Fibonacci dizisinin eleman sayısını al

    // Fibonacci dizisinin n elemanını yazdıran fonksiyonu çağır
    fibonacciDizisi(n);
    return 0;
}
```

```
Fibonacci dizisinin eleman sayisini girin: 10
0 1 1 2 3 5 8 13 21 34
```

Fonksiyonlarda değişken kapsamı

C dilinde, özellikle de fonksiyonlar kullanılırken, **değişkenlerin geçerliliği** yani değerlerinin hangi alanlarda kullanılabilir olduğu önemli bir kavramdır ve bu alanlara değişken kapsamı adı verilir. Değişken kapsamı, bir değişkenin tanımlandığı yerin programın farklı bölümlerinde nasıl etkili olduğunu belirler. Bu kapsam, değişkenin hangi bölümlerde erişilebilir ve değiştirilebilir olduğunu netleştirir. C dilinde üç ana değişken kapsamı türü bulunur:

- **yerel değişkenler (local variables),**
- **global değişkenler (global variables) ve**
- **statik değişkenler (static variables).**

Bu kapsam türleri, değişkenlerin kullanım süresini ve erişilebilirliğini etkiler. Değişken kapsamının doğru yönetilmesi, özellikle büyük ve karmaşık projelerde bellek kullanımını optimize eder ve değişkenlerin beklenmedik şekilde değiştirilmesini engeller.

yerel değişken

```
int fonk1(int a) {
    int d, a;
    .
    . işlemler;
    .
}

void main() {
    int x;
    .
    . işlemler;
    .
}
```

global değişken

```
int z, b;

int ekle(int a) {
    int a;
    .
    b = a + 2
}

void main() {
    .
    z = b + 1;
    .
}
```

sabit değişken

```
int veriAyar(int x) {
    static int k = 0;
    .
    k += x + 1
}

void main() {
    int m = 3;
    .
    veriAyar(m); // k=4
    veriAyar(m); // k=8
}
```

Yerel (lokal) değişkenler

Bir fonksiyonun veya kod bloğunun içinde tanımlanan ve sadece o fonksiyon veya blok içinde geçerli olan değişkenlerdir. Bu değişkenler, fonksiyon veya blok çalıştığında bellekte oluşturulur ve fonksiyon veya blok sona erdiğinde bellekten silinir. Yerel değişkenler, diğer fonksiyonlarla çakışmayı önlemek için sadece tanımlandıkları yerden erişilebilir, bu da programın daha güvenli ve modüler olmasını sağlar.

Kullanım amacı, bir işlemi gerçekleştirmek için gerekli olan geçici verilerin saklanmasıdır ve bu veriler, diğer fonksiyonlardan izole edilerek bağımsız bir çalışma ortamı sağlar. Yerel değişkenlerin kullanımı, özellikle büyük programlarda değişken isimlerinin çakışmasını engelleyerek kodun okunabilirliğini artırır ve hata riskini azaltır.

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

void sub();

void main() {
   system("cls");
   int x = 10;
   printf("main'in basinda x = %d\n", x);
   sub();
   printf("main'in sonunda x = %d\n", x);
}
```

```c
void sub() {
    int x = 5;
    printf("sub'in basinda x = %d\n", x);
    {
        printf("sub içi bloğun basinda x = %d\n", x);
        int x = 1;
        printf("sub içi bloğun sonunda x = %d\n", x);
    }
    printf("sub'in sonunda x = %d\n", x);
}
```

```
main'in basinda x = 10
sub'in basinda x = 5
sub içi bloğun basinda x = 5
sub içi bloğun sonunda x = 1
sub'in sonunda x = 5
main'in sonunda x = 10
```

Aynı isimli değişkenlerin değerleri yerel olarak korunmakla beraber kodlama açısından aynı isimlerin kullanılması pek tavsiye edilmez. Bir müddet sonra değişkenin hangi blokta neyi temsil ettiği karışacaktır.

Global değişkenler

Bir fonksiyonun dışında, genellikle bir programın en başında tanımlanan ve tüm program boyunca geçerli olan değişkenlerdir. Bu değişkenler, program başladığında bellekte oluşturulur ve program sona erene kadar bellekte kalır. Global değişkenlere programın herhangi bir yerinden erişilebilir ve tüm fonksiyonlar tarafından paylaşılabilir. Bu özellik, farklı fonksiyonlar arasında veri paylaşımını kolaylaştırır ve aynı veriye birden fazla yerden erişme ihtiyacını karşılar. Ancak, global değişkenlerin kullanımı dikkat gerektirir, çünkü değişkenin değeri herhangi bir fonksiyondan değiştirilebilir, bu da beklenmedik hatalara yol açabilir.

Global değişkenler, genellikle programın genel durumu veya birçok fonksiyon tarafından kullanılan ortak verileri saklamak için kullanılır, bu da veriyi merkezi bir yerde tutarak yönetimi kolaylaştırır.

```
#include<stdio.h>
int globalDegisken = 20;
void ornekFonksiyon() {
    globalDegisken += 5;
    printf("Global Degisken (Fonksiyon içinde): %d\n", globalDegisken);
}
void main() {
 printf("Global Degisken (Ana fonksiyon içinde): %d\n", globalDegisken);
 ornekFonksiyon();
 return 0;
}
```

Test amacıyla **ornekFonksiyon** içinde değer arttırımı kaldırılıp **globalDegisken** değerindeki değişim kontrol edilebilir.

Statik değişkenler

Bir fonksiyonun veya dosyanın içinde tanımlanabilir ve değeri korunan bir özelliğe sahiptir: Bu değişkenler sadece bir kez bellekte oluşturulur ve program boyunca değerlerini korur. Fonksiyon içinde tanımlanan bir statik değişken, fonksiyon her çağrıldığında yeniden oluşturulmaz, önceki çağrılarda aldığı değeri saklar. Bu nedenle, fonksiyonlar arasında değerini kaybetmeyen verilerin tutulması gerektiğinde kullanılır. Statik değişkenler, global olarak tanımlandıklarında ise sadece tanımlandıkları dosya içinde erişilebilirler, böylece değişkenin erişim kapsamı sınırlandırılmış olur.

Statik değişkenlerin kullanımı, özellikle fonksiyonların durumunu koruma veya sayacı takip etme gibi işlemler için oldukça faydalıdır ve değişkenlerin bellekte daha verimli yönetilmesini sağlar.

```c
#include<stdio.h>

void sayacFonksiyonu() {
    static int sayac = 0;  // Statik yerel değişken
    sayac++;
    printf("Fonksiyon %d. kez çağrıldı.\n", sayac);
}

int main() {
   sayacFonksiyonu();
   sayacFonksiyonu();
   sayacFonksiyonu();
 return 0;
}
```

```
Fonksiyon 1. kez çağrıldı.
Fonksiyon 2. kez çağrıldı.
Fonksiyon 3. kez çağrıldı.
```

Çalışma soruları

1. Ders içinde verilmiş olarak Kelvin'den Fahrenheit'a sıcaklık çevirme işlemini Fahrenheit'tan Kelvin'e dönüştürme şeklinde yeniden yazınız. Kullanıcıdan Fahrenheit olarak alınan sıcaklığı Kelvin'e çeviren kodu derste verilen benzer şekilde ayarlayınız.

2. Kullanıcıdan aldığı boy ve kilo değerlerini kullanarak kişinin Vücut Kitle Indeksi (VKI) değerini bir fonksiyon ile hesaplayıp, bu VKI değerine göre kişinin "zayıf, normal, kilolu ve obez" olduğu bilgisini veren kodu yazınız (VKI bağıntısı internet ortamında bu bağıntı bulunacaktır.)