



ITU ACM Student Chapter Course Program

Introduction to C

Week 3

Instructor

Mihriban Nur Koçak

Prepared by

Mehmet Yiğit Balık & Mihriban Nur Koçak & Emir Oğuz

Fonksiyonlar

Fonksiyon, kod bloklarından oluşan spesifik bir görevi gerçekleştirmeyi amaçlayan temel bir programlama birimidir. Fonksiyonlar kompleks problemleri küçük parçalara bölerek programın anlaşılmasını ve tekrar tekrar kullanılmasını kolaylaştırır.

İki tip fonksiyon vardır:

- Standart Fonksiyonlar (Standard Functions)
- Kullanıcı Tanımlı Fonksiyonlar (User Defined Functions)

Standart Fonksiyonlar

Standart fonksiyonlar, belirli işlevleri olan, kütüphaneler içerisinde hazır olarak bulunan ve kullanıcının direkt kullanabildiği fonksiyonlardır. Örneğin önceki derslerde gösterilmiş olan **<stdlib.h>** kütüphanesine ait **abs(int value)** fonksiyonu bir standart fonksiyondur. Bu fonksiyon bir tam sayının mutlak değeri alınmak istendiğinde kolayca bu işlemi gerçekleştirmeyi sağlar. Burada **abs(int value)** fonksiyonunda parantez içerisinde görülen "int value" değeri fonksiyonda işleme girmesi istenilen değerin veri tipini temsil eder yani bu fonksiyonun **parametresidir**.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    printf("%d\n", abs(-12));
    return EXIT_SUCCESS;
}
```

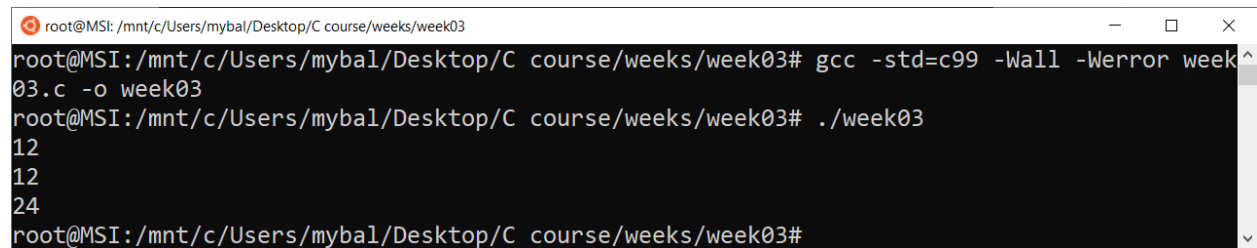
```
root@MSI: /mnt/c/Users/mybal/Desktop/C course/weeks/week03
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week03# gcc -std=c99 -Wall -Werror week03.c -o week03
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week03# ./week03
12
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week03#
```

Kullanıcı Tanımlı Fonksiyonlar

Kullanıcı tanımlı fonksiyonlar, kullanıcının kendisinin belirli bir işlevi gerçekleştirmek üzere yazmış olduğu fonksiyonlardır. Kullanıcı bu fonksiyonun tüm yapısını (döndüreceği veri tipi, adı, parametreleri, döndüreceği değer) kendisi belirler ve işlemi gerçekleştirecek kod bloğunu da kendisi yazar.

```
#include <stdio.h>
#include <stdlib.h>
int toplama_fonksiyonu (int a, int b){
    int sonuc = a + b;
    return sonuc;
}

int main(){
    int sayi_1;
    int sayi_2;
    scanf("%d %d",&sayi_1,&sayi_2);
    printf("%d\n",toplama_fonksiyonu(sayi_1,sayi_2));
    return EXIT_SUCCESS;
}
```



```
root@MSI: /mnt/c/Users/mybal/Desktop/C course/weeks/week03
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week03# gcc -std=c99 -Wall -Werror week03.c -o week03
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week03# ./week03
12
12
24
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week03#
```

Fonksiyonlar belirli bir yapıdan oluşur

- Fonksiyonun döndüreceği değerın veri tipi
- Fonksiyonun adı
- Fonksiyonun parametreleri (argümanları)
- Fonksiyonun işlevini gerçekleştiren kod bloğu
- Fonksiyonun döndüreceği değer

Örneğin burada yazılmış olan fonksiyonun adı kullanıcı tarafından **toplama_fonksiyonu** olarak belirlenmiştir. Fonksiyonun adının önünde yazan veri tipi yani **int** de fonksiyonun döndüreceği değerin veri tipini ifade eder. Fonksiyon adının ardından gelen parantez içerisinde belirtilmiş **int a** ve **int b** değerleri ise fonksiyonun parametresidir. Fonksiyonun içerisinde yapılan işlemler fonksiyonun işlevini gerçekleştirir. En sonunda fonksiyonda **return** ile döndürülen **sonuc** değeri ise fonksiyonunun çıktısıdır.

```
int toplama_fonksiyonu (int a, int b){  
    int sonuc = a + b;  
    return sonuc;  
}
```

Görüldüğü üzere kullanıcı tarafından yazılmış **toplama_fonksiyonu** 'na , **main** fonksiyonu içerisinde tanımlanmış **sayi_1** ve **sayi_2** değerleri parametre olarak verilmiş ve istenilen çıktı alınmıştır. Burada dikkat edilmesi gereken nokta fonksiyon tanımında parametreler **a** ve **b** olarak isimlendirilmiş olsa da fonksiyon çağırılırken parametre olarak verilecek değişkenlerin isimlerinin **a** ve **b** olmak zorunda olmamasıdır. Sonuç olarak **printf** tarafından bastırılan değer **int** veri tipinde **sayi_1** ve **sayi_2** değişkenlerinin değerlerinin toplamıdır.

```
int main(){  
    int sayi_1;  
    int sayi_2;  
    scanf("%d %d",&sayi_1,&sayi_2);  
    printf("%d\n",toplama_fonksiyonu(sayi_1,sayi_2));  
    return EXIT_SUCCESS;  
}
```

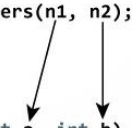
How to pass arguments to a function?

```
#include <stdio.h>

int addNumbers(int a, int b);

int main()
{
    ... ..
    sum = addNumbers(n1, n2);
    ... ..
}

int addNumbers(int a, int b)
{
    ... ..
    ... ..
}
```

A diagram with two arrows. One arrow starts from 'n1' in the function call 'addNumbers(n1, n2)' inside the main function and points down to the parameter 'a' in the function definition 'int addNumbers(int a, int b)'. The other arrow starts from 'n2' in the function call and points down to the parameter 'b' in the function definition.

Not: C kodları derleyici tarafından yukarıdan aşağıya doğru okunur. C dilinin bir kuralı olarak fonksiyonlar **main** kısmının üstüne yazılmalıdır. Eğer üstüne değil de alta yazılırlarsa **main** kısmının üstüne bu fonksiyonun bir prototipi konulmalıdır.

Bir başka örnek olarak istenilen bir değerin, fonksiyon yardımıyla karesinin alınması verilebilir. Burada bir önceki örnekten farklı olarak fonksiyonun çıktı değerini direkt yazdırmak yerine bu çıktı, main fonksiyonu içerisinde tanımlanmış bir değere atanmıştır.

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

int kare_alma_fonksiyonu (int a){
    int sonuc = a * a;
    return sonuc;
}

int main(){
    int sayi;
    printf("Karesi alınacak sayiyi giriniz: ");
    scanf("%d",&sayi);
    int sayinin_karesi = kare_alma_fonksiyonu(sayi);
    printf("sayinin karesi: %d\n",sayinin_karesi);
    return EXIT_SUCCESS;
}
```

```
root@MSI: /mnt/c/Users/mybal/Desktop/C course/weeks/week03
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week03# gcc -std=c99 -Wall -Werror week03.c -o week03
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week03# ./week03
Karesi alınacak sayiyi giriniz: 12
sayinin karesi: 144
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week03#
```

Bir fonksiyonun döndürebileceği bir çok veri tipi vardır. Bunlar temel veri tipleri ve **void** veri tipidir. **void** fonksiyonun döndürdüğü bir verinin olmamasıdır. Yani **void** veri tipinde tanımlanmış bir fonksiyon herhangi bir değer döndürmez (**return**).

```
#include <stdio.h>
#include <stdlib.h>
void Asal_sayi_kontrolu (int a){
    int bolenler = 0;
    for(int i = 2; i < a; i++){
        if(a % i == 0){
            bolenler++;
        }
    }
    if(bolenler == 0){
        printf("Asaldir \n");
    }
    else{
        printf("Asal degildir\n");
    }
}
int main(){
    int sayi;
    printf("Asal sayi kontrolu icin bir sayi giriniz: ");
    scanf("%d",&sayi);
    Asal_sayi_kontrolu(sayi);
    return EXIT_SUCCESS;
}
```

```
root@MSI: /mnt/c/Users/mybal/Desktop/C course/weeks/week03
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week03# ./week03
Asal sayi kontrolu icin bir sayi giriniz: 12
Asal degildir
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week03# ./week03
Asal sayi kontrolu icin bir sayi giriniz: 7
Asaldir
```

Lokal ve Global Değişkenler, Macro Sabitleri

- **Lokal değişkenler:** Fonksiyonların içinde tanımlanmış ve sadece tanımlandığı fonksiyonlar içerisinde kullanılabilen yerel değişkenlerdir.
- **Global değişkenler:** Fonksiyonların dışında tanımlanmış ve tüm fonksiyonlar tarafından kullanılabilen evrensel değişkenlerdir.
- **Parametre olarak tanımlanan değişkenler:** Fonksiyon parametresi olarak tanımlanan, fonksiyon çağırılırken kendisine bir değer atanan ve sadece tanımlandığı fonksiyon içerisinde kullanılabilen değişkenlerdir.
- **Macro sabitleri:** **#define** anahtar kelimesiyle kullanılan, büyük harflerle adlandırılan ve değiştirilemez bir değere sahip sabitlerdir(**const**). Makrolar globaldir ve tüm fonksiyonlar tarafından kullanılabilir.

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

#define MAKRO 12 // bu Makro sabitidir
int global = 12; // bu Global degiskendir
void Asal_sayi_kontrolu (int a){ //bu parametre olarak tanımlanmıştır
    /*Code*/
    int lokal = 12; // bu sadece ait olduğu fonksiyon icinde
    /*Code*/          // kullanılabilen Lokal degiskendir
}
int main() {
    /*Code*/
    return EXIT_SUCCESS;
}
```

Dizilere (Array) Giriş

Diziler belirli bir veri tipine ait birden fazla değeri depolayabilen yapılardır. Her bir dizinin bünyesinde depoladığı değişmez bir veri tipi vardır. Yani bir dizi birden fazla veri tipini depolayamaz. Diziler en fazla belirlenen sayıda veriyi depolayabilirler. Bu kapasite değeri ya dizi tanımlanırken belirlenir ya da diziye belirli bir küme atanarak otomatik olarak belirlenir.

```
veriTipi diziAdi[diziKapasitesi];
```

```
veriTipi diziAdi[] = {veriKumesi};
```

Dizilerin içerisinde depoladıkları verilerinin belirli bir sırası vardır. Bu sıra 0'dan başlayarak verinin depolama kapasitesinin bir eksiğine kadar numaralandırılarak devam eder. Örneğin 5 elemanlı bir dizinin ilk elemanının sıra numarası 0, son elemanının sıra numarası ise 4'tür. Dizi elemanlarına bu sıra numaraları kullanılarak ulaşılır. Diziler verileri belleğin bir kısmında ardışık olarak depolar.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int array_1[] = {1,2,3,4,5};
    int array_2[5];
    for(int i = 0; i < 5; i++){
        printf("array_2 %d. elemani giriniz: ", i+1);
        scanf("%d", &array_2[i]);
    }
    for (int i = 0; i < 5; i++)
    {
        printf("array_1 %d. elemani: %d\n", i+1, array_1[i]);
        printf("array_2 %d. elemani: %d\n", i+1, array_2[i]);
    }
    return EXIT_SUCCESS;
}
```



```
root@MSI: /mnt/c/Users/mybal/Desktop/C course/weeks/week03
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week03# gcc -std=c99 -Wall -Werror week03.c -o week03
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week03# ./week03
array_2 1. elemani giriniz: 6
array_2 2. elemani giriniz: 7
array_2 3. elemani giriniz: 8
array_2 4. elemani giriniz: 9
array_2 5. elemani giriniz: 0
array_1 1. elemani: 1
array_2 1. elemani: 6
array_1 2. elemani: 2
array_2 2. elemani: 7
array_1 3. elemani: 3
array_2 3. elemani: 8
array_1 4. elemani: 4
array_2 4. elemani: 9
array_1 5. elemani: 5
array_2 5. elemani: 0
```

Dizinin herhangi bir elemanına **dizininAdi[elemanınSiraNumarasi]** yapısı kullanılarak ulaşılır. Dizinin elemanına ulaşarak bu değer değiştirilebilir veya bu değer bir değişkene atanabilir.

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

int main(){
    int array_1[] = {1,2,3,4,5};
    array_1[4] = 100;
    array_1[2] = 200;
    for (int i = 0; i < 5; i++)
    {
        printf("array_1 %d. elemani: %d\n",i+1,array_1[i]);
    }
    return EXIT_SUCCESS;
}
```

```
root@MSI: /mnt/c/Users/mybal/Desktop/C course/weeks/week03
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week03# gcc -std=c99 -Wall -Werror week03.c -o week03
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week03# ./week03
array_1 1. elemani: 1
array_1 2. elemani: 2
array_1 3. elemani: 200
array_1 4. elemani: 4
array_1 5. elemani: 100
```

Dizilerin Fonksiyonlarla Kullanımı

Diziler aynı zamanda veri tiplerinde olduğu gibi fonksiyonlarla beraber kullanılabilir. Yani fonksiyonun parametresi bir dizi olabilir. Bu işlem yapılırken dizinin büyüklüğü de fonksiyona parametre olarak gönderilmelidir.

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

double ortalama(int arr[], int size){
    int toplam = 0;
    for(int i = 0; i < size; i++){
        toplam += arr[i];
    }
    double ortalama_sonuc = toplam / size;
    return ortalama_sonuc;
}

int main(){
    int array_1[] = {1,2,3,4,5};

    double array_ortalama_degeri = ortalama(array_1, 5);
    printf("array ortalamasi: %lf\n", array_ortalama_degeri);
    return EXIT_SUCCESS;
}
```

```
root@MSI: /mnt/c/Users/mybal/Desktop/C course/weeks/week03
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week03# gcc -std=c99 -Wall -Werror week03.c -o week03
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week03# ./week03
array ortalamasi: 3.000000
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week03#
```