



ITU ACM Student Chapter Course Program

Introduction to C

Week 3

Instructor

Ahmet Furkan Kavraz

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 gerçekleştirmemizi 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;
}
```

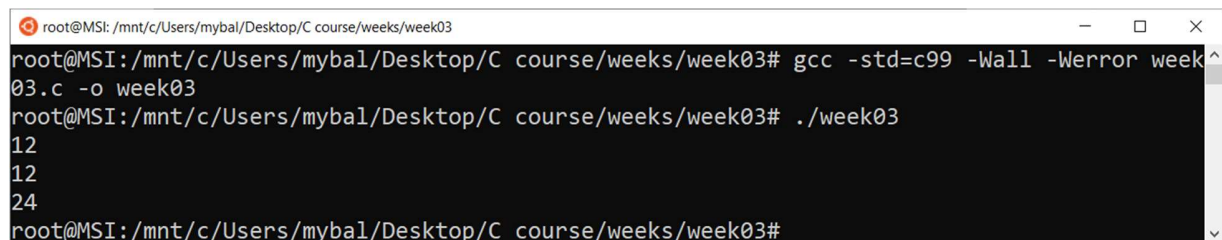
```
root@MSI: /mnt/c/Users/mybal/Desktop/C course/weeks/week03
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week03# gcc -std=c99 -Wall -Werror week
03.c -o week03
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week03# ./week03
12
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week03#
```

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 **a** ve **b** olarak gösterilen parametrelerin isimlerinin fonksiyon çağırılırken parametre olarak alınacak 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)
{
    ... ..
    ... ..
}
```

Two arrows originate from the arguments 'n1' and 'n2' in the function call 'addNumbers(n1, n2);' within the main function. One arrow points down to the parameter 'a' in the function definition 'int addNumbers(int a, int b)', and the other points down to the parameter 'b'.

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
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week03#
```

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
root@MSI: /mnt/c/Users/mybal/Desktop/C course/weeks/week03#
```

Dizinin herhangi bir elemanına **dizininAdi[elemaninSiraNumarasi]** yapısı kullanılarak ulaşılır. Dizinin elemanına ulaşarak bu değer değiştirilebilir ve 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
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week03#
```

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 gönderilebilmelidir.

```
#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#
```

İki Boyutlu Diziler

C dilinde iki ve daha fazla boyuta sahip diziler tanımlanabilir. Bir önceki derste işlenmiş diziler tek boyutlu dizilerdir. İki boyutlu diziler satır ve sütunlardan oluşmuş diziler olarak tanımlanabilir. Burada satırlar boyutlardan birini, sütunlar bir değerini temsil eder. Tek boyutlu dizilerde her bir elemanı temsilen tek bir sıra değeri vardır. İki boyutlu dizilerde ise her bir elemanı temsil etmek için iki sıra değeri (satır ve sütun) birlikte kullanılır.

| | Column 1 | Column 2 | Column 3 | Column 4 |
|-------|----------|----------|----------|----------|
| Row 1 | x[0][0] | x[0][1] | x[0][2] | x[0][3] |
| Row 2 | x[1][0] | x[1][1] | x[1][2] | x[1][3] |
| Row 3 | x[2][0] | x[2][1] | x[2][2] | x[2][3] |

İki boyutlu bir dizi şu şekilde tanımlanabilir:

```
double x[3][4];
```

İki boyutlu dizilere de ilk değer verilebilir. İki boyutlu bir dizinin elemanlarına süslü parantez içinde virgüllerle ayrılan kümeler kullanılarak ilk değer verilir.

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

int main(){
    //buradaki 3 satırı temsil eder, 4 ise sütunu temsil eder.
    double x[3][4] = {{1,2,3,4},{5,6,7,8},{9,10,11,12}};
    for(int i = 0;i < 3;i++){
        for(int j = 0;j < 4;j++){
            printf("%lf ",x[i][j]);
        }
        printf("\n");
    }
    return EXIT_SUCCESS;
}
```

```
root@MSI: /mnt/c/Users/mybal/Desktop/C course/weeks/week04
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week04# gcc -std=c99 -Wall -Werror week04.c -o week04
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week04# ./week04
1.000000 2.000000 3.000000 4.000000
5.000000 6.000000 7.000000 8.000000
9.000000 10.000000 11.000000 12.000000
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week04#
```

Tek boyutlu bir dizinin elemanlarını bastırabilmek için tek bir **for** döngüsü yeterliydi. Fakat iki boyutlu bir dizi için bu yeterli olmayacaktır. Çünkü bahsedildiği gibi iki boyutlu diziler satır ve sütunlardan oluşurlar. Yukarıdaki örnekte görüldüğü üzere iki boyutlu bir diziye bastırabilmek için iç içe iki **for** döngüsü kullanılmıştır. Bu iç içe döngüde **i** değişkeni satırları temsil eder, **j** değişkeni ise **i**'nin temsil ettiği satırdaki sütunları temsil eder. Örneğin **i** değişkeninin değeri 0 iken, döngü 0. satır için çalışmaya başlar. Bu sırada **j** değişkeni de 0'dan 3'e kadar olan sütunları tek tek gezer. Bu işlem sona erdiğinde **i** değişkeninin değeri bir artar ve **j** değişkeni tekrar 0'dan 3'e kadar bu sefer 1. satır için çalışır.

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

int main() {
    double x[3][4] = {{1,2,3,4},{5,6,7,8},{9,10,11,12}};
    x[2][3] = 100;
    for(int i = 0; i < 3; i++) {
        for(int j = 0; j < 4; j++) {
            printf("%lf ", x[i][j]);
        }
        printf("\n");
    }
    return EXIT_SUCCESS;
}
```

```
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week04# gcc -std=c99 -Wall -Werror week04.c -o week04
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week04# ./week04
1.000000 2.000000 3.000000 4.000000
5.000000 6.000000 7.000000 8.000000
9.000000 10.000000 11.000000 100.000000
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week04#
```

Örnekte de görüldüğü üzere her dizide olduğu gibi iki boyutlu dizilerde de değiştirilecek elemanın sıra değeri belirtilerek bu eleman değiştirilebilir.

İki Boyutlu Dizilerin Fonksiyonlarla Kullanımı

Tek boyutlu dizilerde olduğu gibi iki boyutlu diziler de fonksiyonlarla beraber kullanılabilir. Burada en çok dikkat edilmesi gereken nokta iki boyutlu dizinin sütun büyüklüğüdür. Sütun büyüklüğü sabit bir sayı olarak belirlenmelidir ve dizinin belirtildiği her yerde sütun değeri de kendisine ait köşeli parantez içerisinde belirtilmelidir. Bunu yaparken sütun değerini macro olarak belirlemek tavsiye edilen bir yöntemdir. Sütun değeri macro olarak belirlendikten sonra artık değiştirilemez bir değerdir ve tüm fonksiyonlar tarafından kullanılabilir. Bu durumda iki boyutlu diziyi fonksiyona gönderirken beraberinde sadece satır değeri gönderilir.

```
#include <stdio.h>
#include <stdlib.h>
#define SUTUN 5

void deger_ara(int satir,int dizi[][SUTUN],int aranan_deger){
    for(int i = 0;i < satir;i++){
        for(int j = 0;j < SUTUN;j++){
            if(dizi[i][j] == aranan_deger){
                printf("Dizide aradiginiz deger (%d) bulunmaktadır\n",aranan_deger);
                return;
            }
        }
    }
    printf("Dizide aradiginiz deger (%d) bulunmamaktadır\n",aranan_deger);
}

int main(){
```

```

int satir = 5;
int dizi[][SUTUN] = {{1,2,3,4,5},
                      {6,7,8,9,10},
                      {11,12,13,14,15},
                      {16,17,18,19,20},
                      {21,22,23,24,25}};

int aranan_deger;
printf("Dizide aramak istediginiz degeri giriniz: ");
scanf("%d",&aranan_deger);
deger_ara(satir,dizi,aranan_deger);
return EXIT_SUCCESS;
}

```

```

root@MSI: /mnt/c/Users/mybal/Desktop/C course/weeks/week04
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week04# gcc -std=c99 -Wall -Werror 04.c -o week04
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week04# ./week04
Dizide aramak istediginiz degeri giriniz: 12
Dizide aradiginiz deger (12) bulunmaktadır
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week04# ./week04
Dizide aramak istediginiz degeri giriniz: 100
Dizide aradiginiz deger (100) bulunmamaktadır
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week04# ./week04
Dizide aramak istediginiz degeri giriniz: 32
Dizide aradiginiz deger (32) bulunmamaktadır
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week04# ./week04
Dizide aramak istediginiz degeri giriniz: 20
Dizide aradiginiz deger (20) bulunmaktadır
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week04#

```

Sık Kullanılan Dizi Algoritmaları

Dizilerle işlem yaparken sık kullanılan bazı algoritmalar vardır:

- Dizinin en küçük değeri bulma
- Dizinin en büyük değeri bulma
- Dizinin elemanlarını büyükten küçüğe veya küçükten büyüğe sıralama (bubble sorting)

Bir dizideki en büyük değeri bulmak için önce dizinin ilk elemanının değeri **max_deger** olarak belirlenir. Sonrasında dizinin her bir elemanı güncel **max_deger** ile

karşılaştırılarak eğer karşılaştırılan eleman güncel **max_deger**'den büyükse, **max_deger** karşılaştırılan elemanın değerini alarak güncellenir. Bu işlem dizinin son elemanı da **max_deger** ile karşılaştırıldıktan sonra sona erer. Sonuç olarak **max_deger** dizinin en büyük değerini almış olur.

Bir dizideki en küçük değeri bulmak için önce dizinin ilk elemanının değeri **min_deger** olarak belirlenir. Sonrasında dizinin her bir elemanı güncel **min_deger** ile karşılaştırılarak eğer karşılaştırılan eleman güncel **min_deger**'den küçükse, **min_deger** karşılaştırılan elemanın değerini alarak güncellenir. Bu işlem dizinin son elemanı da **min_deger** ile karşılaştırıldıktan sonra sona erer. Sonuç olarak **min_deger** dizinin en küçük değerini almış olur.

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

int maksimum_deger(int dizi[], int boyut){
    //ilk adımda dizinin ilk elemanı max deger olarak varsayılır
    int max_deger = dizi[0];
    //for dongusu kullanılarak en büyük bulunur
    for(int i = 0; i < boyut; i++){
        if(max_deger < dizi[i]){
            max_deger = dizi[i];
        }
    }
    return max_deger;
}

int minumum_deger(int dizi[], int boyut){
    int min_deger = dizi[0];
    for(int i = 0; i < boyut; i++){
        if(dizi[i] < min_deger){
            min_deger = dizi[i];
        }
    }
    return min_deger;
}

int main(){
    int size = 10;
    int arr[size];
    for(int i = 0; i < size; i++){
        scanf("%d", &arr[i]);
    }
    int max_deger = maksimum_deger(arr, size);
    int min_deger = minumum_deger(arr, size);
}
```



```
printf("Dizinin en buyuk elemani: %d\n",max_deger);  
printf("Dizinin en kucuk elemani: %d\n",min_deger);  
return EXIT_SUCCESS;  
}
```

```
root@MSI: /mnt/c/Users/mybal/Desktop/C course/weeks/week04  
root@MSI: /mnt/c/Users/mybal/Desktop/C course/weeks/week04# gcc -std=c99 -Wall -Werror week04.c -o week04  
root@MSI: /mnt/c/Users/mybal/Desktop/C course/weeks/week04# ./week04  
12  
100  
-2  
3  
1  
3  
2  
4  
5  
43  
Dizinin en buyuk elemani: 100  
Dizinin en kucuk elemani: -2  
root@MSI: /mnt/c/Users/mybal/Desktop/C course/weeks/week04#
```

Örnekte bir diziye elemanlarının değerinin büyüklüğüne göre sıralamak için kullanılan algoritmanın adı **Bubble Sort** algoritmasıdır. Bubble sort, en basit sıralama algoritmalarından biridir. Karşılaştırma temelli olan bu algorithmada, listedeki her bir eleman yanındaki eleman ile karşılaştırılır. Eğer ilk elemanın değeri, ikinci elemanın değerinden büyükse, iki eleman yer değiştirir. Daha sonra ikinci ve üçüncü elemanların değerleri karşılaştırılır. İkinci elemanın değeri üçüncü elemanın değerinden büyükse bu iki eleman yer değiştirir, küçükse yer değiştirmezler ve bu işlem, tüm liste sıralanana kadar bu şekilde devam eder. Birbirleriyle karşılaştırılan değerlerin yer değiştirmesi gereken durumlarda bu iki değeri de kaybetmemek adına, değerlerden birini saklayacak geçici bir değişken tanımlanır. Bu değişkene iki değerden biri atanır. Örneğin **deger1** ve **deger2** isimli karşılaştırma sonucu yer değiştirmesi gereken iki değişken olsun. Bu durumda yeni bir **gecici_degisken** tanımlansın. Önce **gecici_degisken**'e değer olarak **deger1** atansın sonrasında **deger1**'e **deger2** atansın ve en sonunda **deger2**'ye **gecici_degisken** atansın. Tüm bu işlemler yapıldığında **deger1** ve **deger2**'nin değerlerinin değişmiş olduğu görülür.

```

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

void kucukten_buyuge(int arr[], int n)
{
    int i, j;
    for (i = 0; i < n - 1; i++)
    {
        for (j = 0; j < n - i - 1; j++)
        {
            if (arr[j] > arr[j + 1])
            {
                int gecici_degisken = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = gecici_degisken;
            }
        }
    }
}

void buyukten_kucuge(int arr[], int n)
{
    int i, j;
    for (i = 0; i < n - 1; i++)
    {
        for (j = 0; j < n - i - 1; j++)
        {
            if (arr[j] < arr[j + 1])
            {
                int gecici_degisken = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = gecici_degisken;
            }
        }
    }
}

void printArray(int arr[], int size)

```

```

{
    int i;
    for (i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

int main()
{
    int buyukluk = 10;
    int arr[buyukluk];
    for (int i = 0; i < buyukluk; i++)
    {
        printf("Lutfen %d. elemani giriniz: ", i);
        scanf("%d", &arr[i]);
    }
    kucukten_buyuge(arr, buyukluk);
    printf("Kucukten buyuge siralanmis: \n");
    printArray(arr, buyukluk);

    buyukten_kucuge(arr, buyukluk);
    printf("Buyukten kucuge siralanmis: \n");
    printArray(arr, buyukluk);

    return EXIT_SUCCESS;
}

```

```

root@MSI: /mnt/c/Users/mybal/Desktop/C course/weeks/week04
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week04# gcc -std=c99 -Wall -Werror week04.c -o week04
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week04# ./week04
Lutfen 0. elemani giriniz: -90
Lutfen 1. elemani giriniz: 100
Lutfen 2. elemani giriniz: 50
Lutfen 3. elemani giriniz: 2
Lutfen 4. elemani giriniz: 1
Lutfen 5. elemani giriniz: 0
Lutfen 6. elemani giriniz: -2
Lutfen 7. elemani giriniz: -1
Lutfen 8. elemani giriniz: 32
Lutfen 9. elemani giriniz: -87
Kucukten buyuge siralanmis:
-90 -87 -2 -1 0 1 2 32 50 100
Buyukten kucuge siralanmis:
100 50 32 2 1 0 -1 -2 -87 -90
root@MSI:/mnt/c/Users/mybal/Desktop/C course/weeks/week04#

```