

BÖLÜM 10: FONKSİYONLAR

Programlama dillerindeki temel düşünce, programların bloklar halinde yazılmasıdır. Profesyonel yazılımlar incelenirse, programların mantıksal bütünlük gösteren parçalar halinde yazıldığı görülecektir. Böylece, programın yazılması ve istenildiğinde değişiklikler yapılması kolaylaşacaktır. C programlama dilinde bu amaca yönelik olarak fonksiyonlar bir veya birden fazla işlemden oluşan alt programlar olarak da düşünülebilir.

C dilinde yazılan ana programın kendisi de aslında `main()` adında bir fonksiyondur. `main()` fonksiyonu yazılmaksızın herhangi bir başka fonksiyonun kullanılması mümkün değildir. Geleneksel olarak, programcılar tarafından komut olarak adlandırılmalarına rağmen, `scanf`, `printf`, `sin`, `log` ve bunlar gibi, şimdiye kadar kullanmış olduğumuz ve standart C derleyicileri tarafından desteklenmiş (header dosyalarında yer almış) bütün komutlar gerçekte birer fonksiyondur. Ancak bu bölümde standart C header dosyalarında yer alan hazır fonksiyonlardan değil, programcı tarafından yazılarak geliştirilen fonksiyonlardan söz edilecektir.

Aşağıda, programcı tarafından yazılan fonksiyonların genel şekli görülmektedir.

```
dönüş_tipi/void fonsiyon_adi([varsa] parametreler)
{
    [varsa] yerel tanımlamaları
    işlem1;
    işlem2;
    ...
    işlemN;
    [varsa] return değer;
}
```

dönüş_tipi: Eğer fonksiyon bir değer geri döndürecek ise değer tipi belirtilir ve bu değer `return` deyimi ile döndürülür. Döndürülen değer fonksiyon dönüş tipiyle (`int`, `float`, `char`, `v.b`) aynı olmalıdır. Değer döndürmeyecek ise `dönüş_tipi` yerine `void` yazılır ve `return` ifadesiyle herhangi bir değer döndürülmez.

fonsiyon_adi: Fonksiyona çağrıda bulunurken kullanılacak belirleyici ad olup değişken ismi tanımlarken uyulması gereken kurallar fonksiyon adı tanımlamaları için de geçerlidir.

parametreler: Varsa, fonksiyon içinde kullanılması gerekli değerleri ve tiplerini içerir. Her parametre değişken tanımlar gibi tanımlanır. Kullanılan parametre birden fazla ise birbirinden `,` ile ayrılır.

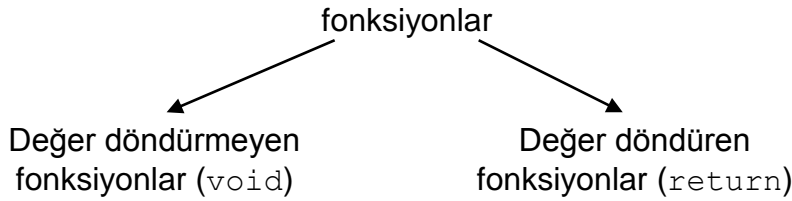
yerel tanımlamaları: Fonksiyona özgü değişken ve sabit gibi tanımlamalardır. Bu bölümde fonksiyonları yakından ilgilendiren değişken tanımlamalarına da yer vermemiz gerekmektedir.

Fonksiyon Prototipi: Tanımlanan bir fonksiyonun programın başında prototipinin yazılması gerekir. Prototip ön bildirimi ile fonksiyona çağrıda bulunulmadan önce derleyiciyi böyle bir fonksiyonun varlığından haberdar ederiz. Varsa dönüş değeri ve aldığı parametrelerin tipleri tanımlanır. Örnek:

`int kup(int x);` kup fonksiyonu `int` bir `x` değeri alır ve yine `int` bir değer döndürür.

Program yazımındaki satır sıralamasında, tanımlanan fonksiyon, bu fonksiyona çağrıda bulunulan satırlardan önce geliyorsa, fonksiyon prototipi ön bildirimine gerek kalmayabilir. Ama tavsiye edilen uygulama, tanımlanan her fonksiyonun program başında prototipinin yazılmasıdır. Bu, programcıya uygulamada esneklik sağlar.

Bir fonksiyona bir değer üretmek ve üretilen değeri çağrıda bulunan fonksiyona döndürülmesi amacıyla değil de bir işlevi yerine getirmesi amacıyla çağrıda bulunulmuş olabilir. Bu noktada şöyle bir ayırım yapabiliriz:



Şimdi bu iki tip fonksiyonu daha yakından inceleyelim.

I. DEĞER DÖNDÜRMEYEN FONKSİYONLAR

Bu fonksiyonlar, alt program gibi kullanılarak ana fonksiyon (`main`) veya diğer fonksiyonlar tarafından çağrıda bulunulur. Fonksiyon içindeki işlemler icra edilir, fakat çağrıda bulunan fonksiyona herhangi bir değer döndürmezler. Aşağıdaki programı inceleyelim:

```
#include <stdio.h>
void yaz(); /* fonksiyon ön bildirimi */
void main()
{
    yaz();
}

void yaz()
{
    printf("C Programlama Notları\n");
}
```

Program çıktısı:

C Programlama Notları

Bu program çalıştırıldığında `yaz()` fonksiyon ismi kullanılarak fonksiyon çağrılır. `yaz()` fonksiyonu ana fonksiyon ile aynı ortamda yazılmış ve derlenmiştir. `yaz()` fonksiyonunun içinde de `stdio.h` header dosyasında tanımlanmış `printf` fonksiyonu kullanılarak ekrana C Programlama Notları yazdırılır. Bu işlem tamamlandıktan sonra ana programa herhangi bir değer aktarılması söz konusu değildir.

Tanımlanan fonksiyonlara, çağrıda bulunma esnasında, kullanılmak üzere parametreler yardımıyla değer aktarılabilceğini biliyoruz. Bu değerler fonksiyon

tarafından kullanılır ve verilen görev yerine getirilir. Ama fonksiyon tarafından üretilen değerin çağrıda bulunan fonksiyona aktarılması söz konusu olmayabilir. Çağrıda bulunan fonksiyon tarafından aktarılan değerler, fonksiyon tanımlaması sırasında (yani fonksiyon yazılırken), fonksiyon adının yanında, parantez içersinde parametre olarak listelenir. Aşağıdaki örnek programda `carp()` fonksiyonuna ana fonksiyondan değer aktarılmaktadır.

```
#include <stdio.h>
void carp(int x, int y); /* fonksiyon ön bildirimi */
void main()
{   carp(10,20);
    carp(5,6);
    carp(8,9);
}

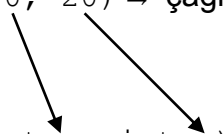
void carp(int x, int y) /* fonksiyon tanımlaması */
{   printf ("%2d * %2d = %3d\n", x, y, x*y);
}
```

Program çıktısı:

```
10 * 20 = 200
 5 *  6 =  30
 8 *  9 =  72
```

Yukarıdaki program çalıştırıldığında `carp()` fonksiyonunun ilk defa kullanıldığı satırda 10 ve 20 sayıları, fonksiyon tanımlaması sırasında parametre olarak verilen `int` tipi `x` ve `y` değişkenlerine gönderilir. `x` değişkenine 10, `y` değişkenine 20 değeri aktarılmış olur. Fonksiyondaki `printf` bu iki sayının çarpımını görüntüler. Aşağıda, ana programdan fonksiyona değer aktarımı görülmektedir.

`carp(10, 20) → çağrı satırı`



`carp(int x, int y) → fonksiyon tanımlaması`

Bu programda `carp()` fonksiyonu ikinci defa çağrıldığında ise 5 ve 6 değerleri fonksiyonun parametreleri olan `x` ve `y` değişkenlerine gönderilmekte ve fonksiyondaki `printf` bu sayıların çarpımını görüntülemektedir. `carp()` fonksiyonunun üçüncü çağrılışında ise `x` ve `y` değişkenlerine 8 ve 9 değerleri gönderilmiş ve bunların çarpımı fonksiyon tarafından görüntülenmiştir.

Aşağıdaki programda, klavyeden bir sayı girilmekte, `kare()` fonksiyonu bu sayının karesini alarak görüntülenmektedir.

```
#include <stdio.h>
void kare(int x); /* fonksiyon ön bildirimi */
void main() /* ana fonksiyon */
{   int sayi;
    printf("Bir sayı giriniz:");
    scanf ("%d", &sayi);
    kare(sayi);
}
```

```
void kare(int x) /* fonksiyon */
{   printf ("Sayı: %d Karesi: %d\n", x, x*x);
}
```

Program çıktısı:

```
Bir sayı giriniz: 5
Sayı: 5 Karesi: 25
```

Bu programda ana fonksiyonda klavyeden girilen sayı, `kare` fonksiyonunun adının yanında parantez içinde yazılı olan `int` tipi `x` değişkenine aktarılmakta ve fonksiyon bu sayının karesini hesaplayıp görüntülemektedir.

C programlama dilinde değişkenler tanımlandıkları konum itibarıyla iki gruba ayrılır. Bunlar;

Genel değişken (global variable): Programın hemen başında, main fonksiyonunun dışında tanımlanan değişkenlerdir. Tüm fonksiyonlarda (main dahil) geçerlidir ve programın her bölümünden değerine erişilebilir.

Yerel değişken (local variable): Herhangi bir fonksiyon içinde tanımlanan değişkenlerdir. Sadece tanımlandığı fonksiyonda geçerlidir ve sadece bu fonksiyon içersinde değerine erişilebilir.

```
#include<...>
.... fonk(...);
int a; /*tüm fonksiyonlardan değerine erişilebilir,
        değeri değiştirilebilir. (global) */

void main()
{   int b; /* sadece main fonksiyonu içerisinden
        erişilebilir (local ) */
    ...
}

.... fonk(...)
{   int b; /* sadece fonk fonksiyonu içerisinden
        erişilebilir. main fonksiyonundaki
        b değişkenden bağımsızdır (local ) */
    int c; /* sadece fonk fonksiyonu içerisinden
        erişilebilir (local ) */
    ...
}
```

Fonksiyonun içinde tanımlanan bir değişkenin (lokal değişken) sadece o fonksiyon içinde geçerli olduğunu biliyoruz. Hatta aynı isimli değişken hem ana programda hem de fonksiyon içinde tanımlanabilir ve bu değişkenlerin isimleri aynı olmasına rağmen farklı değerler taşıyabilir. Aşağıdaki programda `x` değişkeni hem ana programda hem de fonksiyon içinde tanımlanmış olmasına rağmen iki ayrı değişken olarak kullanılmaktadır.

```
#include <stdio.h>
void fonk();

void main() /* ana program */
{
    int x;
    x=10;
    printf("ana programdaki x degeri: %d\n", x);
    fonk();
    printf("ana programdaki x degeri: %d\n",x);
}

void fonk() /* fonksiyon */
{
    int x;
    x=100;
    printf("fonksiyondaki x degeri:  %d\n", x);
    return;
}
```

Program çıktısı:

```
ana programdaki x degeri: 10
fonksiyondaki x degeri:  100
ana programdaki x degeri: 10
```

Bu programda önce ana programdaki `x` değişkenine 10 sayısı aktarılıyor ve görüntüleniyor. Daha sonra `fonk` fonksiyon çağırılmış. Fonksiyonda tanımlanmış olan `x` değişkenine ise 100 sayısı atanıyor ve bu da fonksiyon tarafından görüntüleniyor. Fonksiyon icra edildikten sonra tekrar ana programa dönülüp ana programdaki `x` değişkeni görüntüleniyor. Görüldüğü gibi, fonksiyondaki `x` değişkenine 100 değeri atandığı halde, ana programdaki `x` değişkeninin değeri değişmemiştir.

Ana programa herhangi bir değer aktarmayan fonksiyonlara, `void` fonksiyonlar da denir. Fonksiyon `void` tipi tanımlanarak, çağırıldığı noktaya herhangi bir değer göndermeyeceği vurgulanmış olur.

Fonksiyon sonlarındaki `return` ifadesi iki amaçla kullanılabilir. Bunlardan birincisi fonksiyonun çalışmasını sona erdirmek için isteğe bağlı olarak kullanılması, ikincisi ise fonksiyonda üretilen değeri ana programa aktarmaktır. Yukarıdaki kullanımda, `return` ifadesi fonksiyonun çalışmasını sona erdirmek amacıyla kullanılmıştır.

II. DEĞER DÖNDÜREN FONKSİYONLAR

Bu tip fonksiyonlarda, fonksiyon tarafından üretilen değer `return` ifadesi ile çağrıda bulunan fonksiyona döndürülmekte ve bu değer orada kullanılmaktadır.

Aşağıdaki programda ana programdan girilen iki sayı fonksiyona gönderilmekte, fonksiyon bu iki sayıyı çarparak ürettiği sonucu ana programdaki değişkene aktarmaktadır.

```
#include <stdio.h>
int carp(int x, int y); /* fonksiyon ön bildirimi */
void main() /* ana program */
{
    int a, b, c;
    printf ("Birinci sayiyi giriniz:");
    scanf("%d", &a);
    printf("Ikinci sayiyi giriniz:");
    scanf("%d", &b);
    c=carp(a, b);
    printf("Carpim: %d\n", c);
    /* veya printf("Carpim: %d\n", carp(a, b)); */
}

int carp(int x, int y) /* fonksiyon */
{
    int sonuc;
    sonuc= x * y;
    return sonuc; /* veya return x*y; */
}
```

Program çıktısı:

```
Birinci sayiyi giriniz: 6
Ikinci sayiyi giriniz: 7
Carpım: 42
```

Bu programda, ana programdan girilen 6 ve 7 sayıları fonksiyondaki integer x ve y değişkenlerine gönderilmektedir. Bu sayılar fonksiyonda çarpılarak elde edilen integer 42 değeri lokal c değişkeninde tutulmaktadır. `return c` ile ise lokal c değişkeninde tutulan değer ana programa aktarılmakta ve aktarılan bu değer tekrar `main` içerisinde tanımlanmış `sonuc` değişkenine aktararak `sonuc` değişkeninin değeri ana program tarafından görüntülenmektedir.

Aşağıdaki programda ise, ana programda dairenin yarıçapı girilmekte, bu değer fonksiyona gönderilmekte, fonksiyon da dairenin alanını hesapladıktan sonra bu değeri ana programa aktarmaktadır.

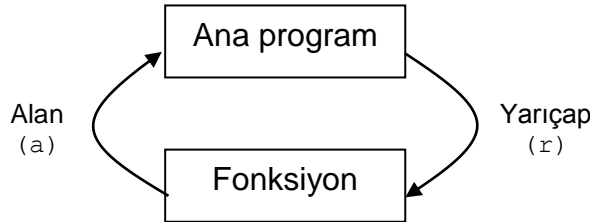
```
#include <stdio.h>
#define PI 3.141593
float daire(float y); /* fonksiyon ön bildirimi */
void main() /* ana program */
{
    float r, s;
    printf ("Dairenin yaricapini giriniz: ");
    scanf ("%f", &r);
    s = daire(r) ;
    printf("Dairenin alani: %6.2f dir.\n",s);
}

/* dairenin alanını hesaplayan fonksiyon */
float daire(float y)
{
    float a;
    a =PI * y * y;
    return(a);
}
```

Program çıktısı:

Dairenin yarıcapini giriniz: 5
Dairenin alanı: 78.54 dir.

Bu programdaki fonksiyon ile ana program arasındaki değer alışverişi aşağıdaki şekilde görülmektedir.



Aşağıdaki programda ise aynı satırda girilen 3 adet tam sayının en büyüğü bulunarak ekrana yazdırılmaktadır. Bu programda en büyük sayının bulunması işlemi `eb` fonksiyonunda yapılmaktadır. Bulunan değer ana fonksiyona döndürülerek ekrana yazdırılmaktadır. Fonksiyon ön bildiriminde `eb` fonksiyonunun üç adet integer değer alacağı bildirilmiş olup, değişken isimleri bildirilmemiştir. Bu tür tanımlamalar sadece fonksiyon prototiplerinde yapılabilir

```
#include <stdio.h>
int eb(int, int, int ); /* fonksiyon ön bildirimi */
void main()
{
    int a, b, c;
    printf("3 adet tam sayı giriniz:");
    scanf("%d%d%d", &a, &b, &c);
    printf("Girilen sayıların en büyüğü: %d\n", eb(a, b, c));
}

int eb(int x, int y, int z )
{
    int ebs = x;
    if ( y > ebs )
        ebs = y;
    if ( z > ebs )
        ebs = z;
    return ebs;
}
```

Aşağıdaki örnekte ise kullanıcı tarafından girilen pozitif bir tamsayının faktöriyeli `fakt` fonksiyonu içerisinde hesaplanıp `main` fonksiyona döndürülmektedir.

```
#include <stdio.h>
int fakt(int); /* fonksiyon ön bildirimi */

void main()
{
    int n;
    printf("Pozitif bir tam sayı giriniz:");
    scanf("%d", &n);
    printf("Sayının faktöriyeli: %d\n", fakt(n));
}
```

```
int fakt(int i)
{
    int j, f = 1;
    for(j = i; j >= 1; j--)
        f = f * j;
    return f;
}
```

Şimdi inceleyeceğimiz fonksiyonda ise `static` tipi bir değişken kullanılmıştır.

```
#include <stdio.h>
int fonk();
void main()
{
    int i;
    for(i=1; i<=5; i++)
        printf ("%d ", fonk());
}

int fonk()
{
    static int a; /* başlangıç değeri 0 kabul edilir */
    a = a + 10;
    return(a);
}
```

Program çıktısı:

10 20 30 40 50

Yukarıdaki programda `fonk` fonksiyonu ana program tarafından defalarca çağırılmakta, her çağırılışında `a` değişkeni daha önceki değerini korumaktadır.

III. RECURSIVE (ÖZYİNELİ) FONKSİYONLAR

Doğrudan veya dolaylı olarak kendi kendini çağıran fonksiyonlardır. Bazı uygulamalarda birçok algoritmanın recursive fonksiyonlarla yazılması daha sade ve kolaydır. Döngü deyimleri kullanılarak yazılan uygulamalar recursive fonksiyonlarla da gerçekleştirilebilir.

Aşağıdaki örnekte kullanıcı tarafından girilen pozitif bir tamsayının faktöriyeli recursive olarak hesaplanmakta ve görüntülenmektedir.

```
#include <stdio.h>
int fakt(int); /* fonksiyon ön bildirimi */
void main()
{
    int n;
    printf("Pozitif bir tam sayı giriniz:"); scanf("%d", &n);
    printf("Sayının faktoriyeli: %d\n", fakt(n));
}

int fakt(int i)
{
    if (i > 1)
        return i*fakt(i-1);
    else
        return 1;
}
```


Örneğin $n = 4$ için fonksiyonun kendisini çağırdığı satır $n! = n \times (n-1)!$ ilkesi uyarınca aşağıdaki gibidir;

```
      4 * fakt(3) --> 3 * fakt(2) --> 2 * fakt(1)
24 = 4 *      6      <-- 3 *      2      <-- 2 *      1
```

Recursive fonksiyonların daha iyi anlaşılması için diğer bir örnek:

```
#include <stdio.h>
void fonk(int); /* fonksiyon ön bildirimi */

void main()
{   int n;
    printf("Pozitif bir tam sayi giriniz: ");
    scanf("%d", &n);
    fonk(n);
}

void fonk(int i)
{   printf("Gelen sayi: %d\n", i);
    if (i > 1)
        fonk(i - 1);
}
```

Örnek program çıktısı:

```
Pozitif bir tam sayi giriniz: 4
Gelen sayi: 4
Gelen sayi: 3
Gelen sayi: 2
Gelen sayi: 1
```

```
printf fonksiyonunun if koşulunda sonraya atılması durumunda;
void fonk(int i)
{   if (i > 1)
        fonk(i - 1);
    printf("Gelen sayi: %d\n", i);
}
```

Örnek program çıktısı:

```
Pozitif bir tam sayi giriniz: 4
Gelen sayi: 1
Gelen sayi: 2
Gelen sayi: 3
Gelen sayi: 4
```

IV. DİZİLERİ FONKSİYONLARA GÖNDERME

Şu ana kadar gördüğümüz bilgi çerçevesinde, fonksiyonlara gönderilen parametre değerlerinin fonksiyon içerisinde değiştirilmesi mümkün değildir. Fonksiyona değişkenin sadece değeri gönderilir. Fonksiyon içerisindeki işlemlerden gönderilen parametrenin kendisi etkilenmez.

Dizilerde durum böyle değildir. Normal kullanımda dizi fonksiyona gönderildiğinde elemanlar değiştirilebilir (referans). Dizin sadece herhangi bir elemanı gönderildiğinde ise değeri değiştirilemez (değer). Diziyi fonksiyona gönderirken sadece adını parametre olarak yazmak yeterlidir.

Aşağıdaki örnekte dizi eleman değerlerinin fonksiyon içerisinde değiştirilmesi gösterilmektedir.

```
#include <stdio.h>
void kareleri(int []);
void main()
{   int a[10];
    int i;
    for (i=0; i<10; i++)
        a[i] = i + 1;
    printf("Dizinin eleman degerleri:\n");
    for (i=0; i<10; i++)
        printf("%d ",a[i]);
    kareleri(a);
    printf("\nKare alma islemi sonrasi degerler:\n");
    for (i=0; i<=9; i++)
        printf("%d ",a[i]);
}

void kareleri(int a[])
{   int i;
    for (i=0; i<=9; i++)
        a[i] = a[i] * a[i];
}
```

Dizi elemanlarının teker teker gönderilmesi durumunda aynı sonucu almak için elemanları aşağıdaki örnekte olduğu gibi adresleriyle göndermemiz gerekir.

```
#include <stdio.h>
void kareleri(int *);

void main()
{   int a[10];
    int i;
    for (i=0; i<10; i++)
        a[i] = i + 1;
    printf("Dizinin eleman degerleri:\n");
    for (i=0; i<10; i++)
        printf("%d ",a[i]);
    for (i=0; i<10; i++)
        kareleri(&a[i]);
    printf("\nKare alma islemi sonrasi degerler:\n");
    for (i=0; i<=9; i++)
        printf("%d ",a[i]);
}
```

```
void kareleri(int *b)
{
    *b = *b * *b;
}
```

veya

```
#include <stdio.h>
int kareleri(int);
void main()
{
    int a[10];
    int i;
    for (i=0; i<10; i++)
        a[i] = i + 1;
    printf("Dizinin eleman degerleri:\n");
    for (i=0; i<10; i++)
        printf("%d ", a[i]);
    for (i=0; i<10; i++)
        a[i] = kareleri(a[i]);
    printf("\nKare alma islemi sonrasi degerler:\n");
    for (i=0; i<=9; i++)
        printf("%d ", a[i]);
}

int kareleri(int b)
{
    return b * b;
}
```

V. FONKSİYONLARI REFERANS YOLUYLA ÇAĞIRMA

Şu ana kadar yazdığımız fonksiyonlarda gönderilen parametrelerin (diziler hariç) değerlerinin değiştirilmesi mümkün değildi. Fonksiyon çağırıldığı zaman parametrelerin bir kopyası çıkartılıp fonksiyona gönderiliyordu. Bir fonksiyonun birden fazla değer döndürebilmesi için pointerlere (işaretçilere) ihtiyaç vardır. Aşağıdaki örneği inceleyelim;

```
#include <stdio.h>
int kare(int);
void main()
{
    int i = 5;
    printf("Oncesi :%d\n", i);
    printf("Karesi :%d\n", kare(i));
    printf("Sonrasi:%d\n", i);
}

int kare(int k)
{
    k++;
    int kare;
    kare = k * k;
    return kare;
}
```

Program çıktısı:

Oncesi :5
Karesi :36
Sonrası:5

Gönderilen parametrenin kopyası fonksiyona gönderildiği için fonksiyon içerisinde yapılan değişiklikler fonksiyonun çağırıldığı yeri etkilemez. Eğer gönderilen parametredeki değişikliklerin fonksiyonun çağırıldığı yerde de geçerli olmasını istiyorsak fonksiyona aşağıdaki gibi parametreyi adresiyle göndermemiz gereklidir. Parametreyi adresiyle gönderdiğimiz için de, aşağıdaki örnekte olduğu gibi ilgili fonksiyonda bu adresi karşılayan bir pointer değişkene ihtiyaç vardır.

```
#include <stdio.h>
int arttir(int *);
void main()
{
    int i = 5;
    printf("Oncesi  %d\n", i);
    printf("Karesi  %d\n", arttir(&i));
    printf("Sonrasi %d\n", i);
}

int arttir(int *k)
{
    (*k)++;
    int kare;
    kare = *k * *k;
    return kare;
}
```

Program çıktısı:

Oncesi :5
Karesi :36
Sonrası:6

Aşağıdaki örnekte, Bubble Sort'ta kullanılan yer değişikliği (swap) algoritmasının fonksiyon yordamıyla uygulanması ve parametrelerin referans yolu ile gönderilmesi verilmiştir. Parametrelerin referans yoluyla değil de, değer yoluyla gönderilmeleri durumunda karşılaşılan sorunu siz tespit ediniz.

```
#include <stdio.h>
#define N 5

void degistir (int *, int *);

void main()
{
    int a[N], i, j;
    printf("%d tane sayi giriniz:\n", N);
    for(i=0;i<N;i++)
    {
        printf("%d. sayi: ", i+1);
        scanf("%d", &a[i]);
    }
}
```

```
    for(i=0;i<N-1;i++)
        for(j= i+1; j<N; j++)
            if(a[j] < a[i])
                degistir(&a[j], &a[i]);
    printf("\nSayilarin siralanisi:\n");
    for(i=0;i<N;i++)
        printf("%d. Sayı: %d\n",i+1, a[i]);
}

void degistir(int *a, int *b)
{
    int gec;
    gec = *a;
    *a = *b;
    *b = gec;
}
```

Dizilerde işaretçi olduğu için a değişkeni bir dizi (veya işaretçi) ise a[i] ile *(a+i) ifadeleri aynı anlamı taşır.

Fonksiyona gönderilen dizinin fonksiyon içerisinde işaretçi olarak kullanımı örneği;

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
float ort (int *);
void rasgele(int s[]);
#define N 10

void main()
{
    int i, s[N];
    rasgele(s);
    for (i=0; i<N; i++)
        printf("%4d\n",s[i]);
    printf("Ortalama:%6.2f\n", ort(&s[0]));
}

float ort (int *a)
{
    int i;
    float t = 0;
    for (i=0; i<N; i++)
        t = t + *(a+i);
    return t/N;
}

void rasgele(int s[])
{
    int i;
    srand(time(NULL));
    for (i=0; i<N; i++)
        s[i] = rand() % 100 + 1;
}
```

VI. HARİCİ FONKSİYONLAR

Şimdiye kadar incelemiş olduğumuz fonksiyonlar, çağrıldıkları ana program ile aynı dosyada yer almıştır. Fonksiyonları, ayrı bir dosya olarak yazdıktan sonra diskete veya hard disk'e kaydedip daha sonra çeşitli programlarda kullanmak mümkündür. Bu tip fonksiyonlara HARİCİ FONKSİYON adı verilir.

Aşağıdaki fonksiyon, önce ayrı bir dosya şeklinde yazılmış ve `yaz.c` adı altında kaydedilmiştir. Fonksiyonun ismi ise `bilgi`'dir. Yani başka bir programda kullanılırken `bilgi()` yazılarak çağrılacaktır.

```
#include <stdio.h>
void bilgi()
{
    printf("C Programlama.\n");
}
```

Aşağıdaki programda ise `bilgi()` fonksiyonu kullanılmıştır.

```
#include "D:\yaz.c"
void main()
{
    bilgi();
}
```

Program çıktısı:
C Programlama.

`bilgi()` fonksiyonu `yaz.c` dosyasında bulunduğu için, programın başlangıcında `#include` direktifi kullanılarak `yaz.c` dosyası programa dahil edilmiş, daha sonra da `bilgi()` yazılarak bu fonksiyon çalıştırılmıştır. Ana programın derlenmesi sırasında fonksiyon da otomatik olarak derlenerek programa dahil edilir.

Aşağıda ise toplama işlemi yapan `topla()` fonksiyonu ve bu fonksiyonu kullanan program görülmektedir.

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

Önce `topla()` fonksiyonu yazılır ve `toplama.c` adı altında hard disk'in D sürücüsüne kaydedilir. Daha sonra da aşağıdaki program yazılır ve derlenir.

```
#include <stdio.h>
#include <D:\toplama.c>
void main ()
{
    int a,b,c;
    printf("Birinci sayiyi giriniz:");
    scanf ("%d",&a) ;
    printf("Ikinci sayiyi giriniz:");
```

```
scanf("%d",&b) ;  
c=topla(a,b) ;  
printf("Sayilarin toplami: %d\n",c);  
}
```

Program çıktısı:

```
Birinci sayıyı giriniz: 5  
İkinci sayıyı giriniz: 10  
Bu sayıların toplamı: 15
```

Bu programda önce `#include` direktifi kullanılarak D'de bulunan `toplam.c` dosyası programa dahil edilir. Bu dosyada `topla()` fonksiyonu bulunmaktadır. Daha sonra ana program içinde `topla()` fonksiyonu kullanılarak toplama işlemi gerçekleştirilmiştir.

VII. `main()` FONKSİYONA DEĞER AKTARILMASI

Ana fonksiyon olan `main()` fonksiyonu şimdiye kadar herhangi bir değer ataması yapılmadan kullanılmıştır. `argc` ve `argv` argümanları kullanarak, komut satırından `main()` fonksiyonuna değer aktarılması mümkündür. `argc`, komut satırından girilen kelimelerin sayısı olup bu sayıya programın ismi de dahildir. `argv` ise komut satırından girilen kelimelerin başlangıç adresini tutan pointerdir.

Aşağıdaki programın `ornek.c` adı altında yazıldıktan sonra derlendiğini ve böylece `ornek.exe` dosyasının elde edildiğini kabul edelim. EXE uzantılı bir dosya olduğuna göre normal olarak komut satırında `ornek` yazılıp ENTER tuşuna basılırsa çalışması gerekecektir. Ancak bu programın özelliği şudur: Komut satırında program adının yanına başka bir kelime de yazılıp ENTER tuşuna basılırsa, bu kelime `main()` fonksiyona gönderilmiş olur.

```
#include <stdio.h>  
  
void main(int argc, char *argv[])  
{  
    if (argc < 2)  
        printf("Adinizi yazmayi unuttunuz.\n");  
    else  
        printf("Merhaba %s", argv[1]);  
}
```

Program çıktısı:

```
D:\Debug> ornek Kadir (ENTER TUŞUNA BASILACAK)  
Merhaba, Kadir
```

Aşağıdaki programda ise, komut satırında program adından sonra yazılan DOS komutları peş peşe çalışır.

```
#include <stdio.h>  
#include <stdlib.h>  
  
void main(int argc, char *argv[])  
{  
    int i;
```

```
        for(i=1; i<argc;i++)
            system(argv[i]);
    }
```

Bu program çalıştırılırken program adının yanına sırayla `DIR`, `VER` ve `CHKDSK` yazdıktan sonra `ENTER` tuşuna bastığımızı düşünelim. Bu durumda bu üç komut peş peşe çalışacaktır.

VIII. GENEL ÖRNEKLER

Örnek 1: Aşağıdaki programda, fonksiyonlar kullanılarak bir işyerinde çalışan işçilerin ücretleri hesaplanmakta ve bunlarla ilgili yazma ve okuma işlemleri yapılmaktadır. Programda değişiklikler yapılarak çeşitli konulara uyarlanabilir.

```
#include <stdio.h>
#include <stdlib.h>
#define N 5
int menu(int);
void giris();
void okuma();
char ad[N][80];
char tel[N][20];
int saat[N];
long maas[N];

void main()
{    int sec;
    do
    {    sec=menu(sec);
        switch(sec)
        {    case 0: break;
            case 1: giris(); break;
            case 2: okuma(); break;
            default: printf("Yeniden deneyiniz");
        }
    }
    while(sec!=0);
}

int menu(int i)
{    char str[80];
    printf("\n");
    printf("0 - Cikis\n");
    printf("1 - Bilgi Girisi\n");
    printf("2 - Bilgi Okuma\n");
    printf ("\nSeciminizi giriniz:");
    gets(str);
    i = atoi(str);
    return i;
}
```



```
void giris()
{
    int i;
    char temp[80];
    for(i=0; i<N; i++)
    {
        printf("\n") ;
        printf("Iscinin adini giriniz :");
        gets(ad[i]);
        printf("Telefon numarasini giriniz :");
        gets(tel[i]);
        printf("Calisilan saat sayisini giriniz :");
        gets(temp);
        saat[i]=atoi(temp);
        printf("Saat ucretini giriniz :");
        gets(temp);
        maas[i]=atoi(temp);
    }
}

void okuma()
{
    int i;
    for(i=0; i<N; i++)
    {
        printf("\n") ;
        printf("Adı : %s Telefon no : %s\n", ad[i], tel[i]);
        printf ("Haftalik odeme: %d\n",maas[i] * saat[i]);
    }
}
```

Bu programdaki `atoi()` fonksiyonu, `char` tipi dizi değişkene aktarılmış olan bir sayıyı integer tipine çevirmektedir.

Örnek 2: Girilen harfi büyük harfe dönüştüren program.

```
#include <stdio.h>
char buyuk(char);

void main ()
{
    char ch;
    printf("Bir harf giriniz: ");
    scanf ("%c", &ch);          /* veya ch = getchar(); */
    printf("%c\n", buyuk(ch));
}

char buyuk(char kar)
{
    if (kar >= 'a' && kar <= 'z')
        return kar - 32;
    else
        return kar;
}
```

Örnek 3: iki sayı ve aralarındaki işlem.

```
#include <stdio.h>
float hesapla(float, float, char);
```

```

void main ()
{
    float a,b;
    char op;
    printf("Operatoru, 1. sayiyi ve 2. sayiyi giriniz:");
    scanf ("%c%f%f",&op, &a, &b) ;
    printf("%6.2f %c %6.2f = %6.2f\n",a,op,b,hesapla(a,b,op));
}

float hesapla(float a, float b, char op)
{
    float s;
    switch(op)
    {
        case '+': s = a + b; break;
        case '-': s = a - b; break;
        case '*': s = a * b; break;
        case '/': s = a / b; break;
        case '%': s = (int)a % (int)b; break;
        default: s = 0;
    }
    return s;
}

```

Örnek 4: *N* adet öğrencinin (öğrenci sayısı belli) öğrenci numarasını ve sınav sonuçlarını okuyup, okunan bu bilgileri giriş sırasına göre görüntüleyen ve sınav sonuç ortalamasını bulan uygulama aşağıda verilmiştir.

```

#include <stdio.h>
void giris(int no[], int not[], int N);
float ortalama(int not[], int N);
void goruntule(int no[], int not[], int N);

void main()
{
    int N;
    int OgrNo[BUFSIZ], BNotu[BUFSIZ];
    float ort;
    printf("Oğrenci sayisini giriniz: ");
    scanf("%d", &N);
    giris(OgrNo, BNotu, N);
    goruntule(OgrNo, BNotu, N);
    ort = ortalama(BNotu, N);
    printf("Not ortalamasi: %5.2f\n", ort);
}

void giris(int no[], int not[], int N)
{
    int i;
    for (i=0; i<N; i++)
    {
        printf("Oğrenci no gir: ");
        scanf("%d", &no[i]);
        printf("%d numaralı öğrencinin notunu gir : ", no[i]);
        scanf("%d", &not[i]);
    }
}

```

```

void goruntule(int no[], int not[], int N)
{
    printf("Ogrenci#\tNotu\n");
    for (int i=0; i<N; i++)
        printf("%8d\t%3d\n", no[i], not[i]);
}

```

```

float ortalama(int not[], int N)
{
    float t;
    int i;
    t = 0;
    for (i=0; i<N; i++)
        t = t + not[i];
    float ort = t / N;
    return ort;
}

```

Yukarıdaki uygulamaya ek olarak;

- 60'dan küçük sınav sonuçlarını ve bu sonuçlara ait öğrenci numaralarını görüntüleyen fonksiyonu,
- Ortalamanın üstünde not alan öğrenci numaraları ve bu öğrencilere ait notları görüntüleyen fonksiyon,
- 60'dan küçük sınav sonuç değerlerinin sayısını veren fonksiyonu (`return`),
- En yüksek sınav sonucunu veren fonksiyonu (`return`),
- En düşük sınav sonucunu veren fonksiyonu (`return`),
- Sınav sonuçlarına göre küçükten büyüğe sıralı olarak öğrenci numaralarını ve sınav sonuçlarını görüntüleyen fonksiyon,
- Öğrenci numaralarına göre küçükten büyüğe sıralı olarak öğrenci numaralarını ve sınav sonuçlarını görüntüleyen fonksiyon ve

- Sınav sonuçlarına ait standart sapmayı veren fonksiyon $\left(\sqrt{\frac{\sum_{i=1}^n (\bar{x} - x_i)^2}{n}} \right)$

yazınız (`return`).

\bar{x} : Not ortalaması
 x_i : i. Öğrencinin notu