

Bölüm 5

Döngü (Loop) Deyimleri / Veri Belirleyicileri / Matematiksel Fonksiyonlar

İçindekiler

5.1	Formatlı Yazdırma	34
5.2	Döngü Deyimleri	34
5.2.1	Neden Döngü?	35
5.2.2	for Döngüsü	35
5.2.3	while Döngüsü	39
5.2.4	do while Döngüsü	41
5.2.5	continue Yapısı	42
5.2.6	break Yapısı	42
5.3	Veri Niteleyicileri	42
5.3.1	signed	43
5.3.2	unsigned	43
5.3.3	short ve long	43
5.4	Değişken Bildirim Yerleri ve Türleri	43
5.5	Matematiksel Fonksiyonlar	44

5.1 Formatlı Yazdırma

`printf` fonksiyonu içerisinde ekrana yazdırmak istediğimiz şeyleri belli bir format içerisinde yazmak mümkündür. Bu durumda format belirtecinin sağında ve solunda rakamlar yer alır. Sözelimi,

```
1 int x = 123456;
2 float y = 22.0/ 7.0;
3 printf("[%2d]\n",x);
4 printf("[%10d]\n",x);
5 printf("[% -10d]\n",x);
6 printf("[%+-10d]\n",x);
7 printf("[%9.3f]\n",y);
```

- 1. ve 2. satırda değişken tanımları yapıp, değer atama işlemleri yapıyor.
- 3. satırda 2'nin anlamı 2 hane boşluk bırakılır ve boşluğun en sağından itibaren rakamlar yazılır. 123456 sayısı 6 haneli olduğu için 2 haneye sığmaz, dolayısı ile tamamı yazılır. Çıktısı, [123456] şeklinde olacaktır.
- 4. satırda 10 hane boşluk bırakılır ve en sağdan itibaren rakamlar yazılır. 6 haneli rakam yazılacağından ilk 4 hane boş kalacaktır. Rakamlar sağa dayalı olarak yazılır. Bu durumda [123456] ifadesi ekrana yazılacaktır.
- 5. satırda -10'nun anlamı 4. satır ile aynı ancak - işaretinden dolayı sağa değil sola dayalı yazılacaktır. Dolayısı ile son dört hane boş kalacaktır. Bu durumda [123456] ifadesi ekrana yazılacaktır.
- 6. satırda 5. satırdaki işlem aynen tekrarlanır. Sadece + işareti eklenir. Bu durumda [+123456] ifadesi ekrana yazılacaktır. Artı işaretinden dolayı sondan 3 hane boş kalacaktır.
- 7. satırda 9.3'ün anlamı 9 hane boş bırak, noktadan sonra 3 hane yaz ve sağa dayalı olarak yaz. Bu durumda, [3.143] ifadesi ekrana yazılacaktır. İlk satırdan itibaren 4 hane boşluk kalacaktır.

5.2 Döngü Deyimleri

Belli bir koşul altında tekrar gerektiren durumlar için döngü yapıları kullanılır. `while`, `do ...while` ve `for` olmak üzere üç tip döngü deyimi vardır. Bu döngülerin yaptığı iş aynıdır. Sadece kullanımı farklıdır. `while` ve `for` döngü deyimleri `do ... while` döngü yapısından

farklıdır. `for` ve `while` döngülerinde koşul ilk önce sınanır, ardından tekrar işlemi koşul sağlandığı sürece devam eder. Ancak `do ... while` döngüsünde ilk önce döngüye girilir, ardından koşula bakılır. Eğer koşul sağlanırsa döngü işlemi gerçekleşir. Sağlanmaz ise döngü işlemi gerçekleşmez.

5.2.1 Neden Döngü?

Döngü kullanımına niçin gereksinim olduğunu basit bir örnek ile açıklayalım. Sözelimi, 1'den 100'e kadar olan sayıları toplayalım. Eğer döngü yapısını kullanılmazsa 1'den 100'e kadar olan sayılar için 100 adet değişken tanımlaması yapmak gerekirdi ve sayıları toplamak için ise bu sayıların ilgili değişkenlere atanarak toplama işlemi yapılırdı.

```
int x1=1, x2=2, ... , x100=100
sonuc = x1+x2+...x100
```

Bu işlem son derece mantıksız ve programlamadan uzaktır.

Bu amaçla döngü programlama açısından son derece önemli ve gerekli bir yapıdır.

Tekrar işleminin yapılabilmesi için bir değişkene ihtiyaç vardır. Bu değişkene *sayaç değişkeni* adı verilir.

5.2.2 for Döngüsü

Kendisinden sonra gelen yapıları tekrarlamak için kullanılır. Eğer ilgili koşul sağlanırsa işlemler belirtilen sayıda tekrarlanır. Koşul sınaması ilk önce yapılır, eğer koşul sağlanırsa döngüye girilir. Aksi durumda döngü işlemi gerçekleşmez.

```
for(sayac-degiskeninin-baslangic-degeri;kosul-sinaması;attırım)
deyim;
```

for döngüsünün yapısını inceleyelim.

- Parantezler içerisinde 3 bölüm bulunur ve her bölüm ; simgesi ile ayrılır!!! ; dışında genellikle , simgesi ile ayırma işlemi sık yapılan hatalardan biridir, dikkat!
- C dilinde eğer bir parantez açılıyorsa, o parantez kesinlikle kapanmalıdır. Dolayısı ile 3 bölümün verilmesinden sonra parantez kesinlikle kapanmalıdır.
- for satırı parantez ile sonlandırıldıktan sonra kesinlikle ; simgesi kullanılmaz.
- Döngü içinde birinci bölümde,döngüyü kontrol eden yani döngünün tekrarını sağlayan sayaç değişkenine başlangıç değeri verilir Eğer başlangıç değeri verilmez ise bellekte o sırada var olan bir sayı değer olarak alınır. Dolayısı ile yanlış sonuçlar elde edilir.

- İkinci bölümde ise döngünün dönebilmesi için sağlaması gereken koşul verilir. Bu koşul için kullanılacak olan değişken sayaç değişkenidir.
- En son bölümde ise döngünün dönmesini yani tekrarını sağlayan sayaç değişkeninin arttırım işlemi yapılmalıdır. Eğer arttırım işlemi düzgün yapılmaz ise döngü sonsuza girer. Sonsuz defa tekrar eder.

Sözelimi;

```
1 int i;  
2 for(i=0;i<3;i++)  
3 printf("i=%d\n",i);
```

Bu for döngüsü şu şekilde çalışacaktır.

1. satırda i sayaç değişkeni tanımlanıyor.

2.satırda ise for döngüsü içinde i sayaç değişkenine başlangıç değeri veriliyor. Yani i=0 atama işlemi gerçekleştiriliyor. Başlangıç değeri olarak istenilen sayı verilebilir.

Koşul i değişkeninin değerinin 3'den küçük olmasıdır. Yani i değişkeni 3'den küçük oluncaya kadar tekrarlama işlemi yapılacak, 3'den büyük olduğunda ise tekrarlama işlemi duracaktır.

i++ işlemi ile döngünün devam edebilmesi için i değişkeninin değeri arttırılıyor. Ancak arttırma işlemi hemen yapılmıyor. İlk adımda yapılmıyor. Döngüye **bir kez** girildikten sonra yapıyor.

Buna göre program şu şekilde çalışacak.

1. TUR

```
i=0          0<3          DOĞRU!  
i=0          EKRANA YAZILIR!
```

2. TUR

```
i++'dan dolayı      i=1 olur.  
i<3                 1<3 DOĞRU!  
i=1                 EKRANA YAZILIR!
```

3. TUR

```
i++'dan dolayı      i=2 olur.  
i<3                 2<3 DOĞRU!  
i=2                 EKRANA YAZILIR!
```

4. TUR

```
i++'dan dolayı      i=3 olur.  
i<3                 3<3 YANLIS!!
```

Döngü sonlanır, dolayısı ile ekrana birşey yazılmaz.

Dolayısı ile bu döngü 3 tur döner ve program çalıştırıldığında

```
i=0  
i=1  
i=2
```

yazılır.

ÖNEMLİ NOT: `i++` yerine `i=i+1`'da yazılabilir. Aynı anlama gelir.

`for` döngüsü içinde birden fazla deyim de kullanılabilir. Ancak bunun için `{}` kullanılmalıdır.

```
for (...)  
{  
deyim1;  
deyim2;  
...  
}
```

Bu işlem benzer şekilde karşılaştırma deyimleri ve diğer döngü işlemleri için de yapılmaz!!!

`for` döngüsünün farklı kullanım biçimleri aşağıda verilmektedir.

- Sonsuz döngü, tekrarlama işleminin sonsuz sayıda yapılmasıdır. Bu durumda sonsuz döngüyü sonlandırmak için `break` kullanmak gerekir. Sonsuz döngü C dilinde `for(;;)` deyimini ile kurulur.
- Sayaç değişkeninin başlangıç değeri döngüden önce de verilebilir.

```
x=30;  
for(;x<50;x+=2)  
    printf("%d\n",x);
```

Bu durumda `for` döngüsü içinde başlangıç değerinin verilmesine gerek yoktur. Bu program 30'dan 50'ye kadar (50 dahil değil) olan sayıları ikişer atlamalı olarak yazar.

- `for(x=1;x<=100;)`
 `printf("%f\n",sqrt(x++));`

Yukarıdaki program parçasında sayaç değişkenine arttırım verilmemiştir. Ancak döngü sayacı `x` döngü içerisinde kullanılırken arttırılmıştır. Bu program, 0 ile 101 arasındaki tamsayıların karekökünü alır ve ekrana yazar.

- `for` döngüsü içinde birden fazla sayaç değişkeni ve onlara ait koşul kullanılabilir.

```
for(x=0, y=0;x+y<15;x++)
{
..
y++;
...
}
```

Bu programda döngü sayacından biri arttırım kısmında, diğeri ise döngü içerisinde attırılmıştır. Bu iki döngü sayacı toplanıp 15 ile karşılaştırılır. Toplamın 15'den küçük olması durumunda döngü devam eder. **Birden fazla sayaç değişkeni kullanılması durumunda bu değişkenler birbirlerinden , karakteri ile ayrılmak zorundadır.**

- İç içe Döngüler (Nested Loops)

```
for(...)
  for(...)
    for(...)
      deyim;
```

Birden fazla döngü iç içe kullanılabilir. Bu durumda en içteki döngü bir dıştakinin herbir tekrarı kadar tekrarlanır.

```
for(i=1;i<=2;i++)
  for(j=1;j<3;j++)
    for(k=1;k<3;k++)
      printf("i=%d\n, j=%d, k=%d\n",i,j,k);
```

Yukarıdaki program şu şekilde çalışır. Sayaç değişkenlerinin aldıkları değerler gösterilmektedir.

K DONGUSU

i=1, j=1, k=1

i=1, j=1, k=1 EKRANA YAZILIR. k++'dan

i=1, j=1, k=2

i=1, j=1, k=2 EKRANA YAZILIR. k++'dan

i=1, j=1, k=3, 3<3 koşul sağlanmaz, döngü sonlanır.

J DONGUSU

j++'dan

i=1, j=2, k=1

```
i=1, j=2, k=1 EKRANA YAZILIR. k++'dan
i=1, j=2, k=2
i=1, j=2, k=2 EKRANA YAZILIR. k++'dan
i=1, j=2, k=3, 3<3 koşul sağlanmaz, döngü sonlanır.
j++'dan
i=1, j=3 olur, 3<3 koşul sağlanmaz döngü sonlanır.
```

I DONGUSU

```
i++'dan
i=2, j=1, k=1
i=2, j=1, k=1 EKRANA YAZILIR. k++'dan
i=2, j=1, k=2
i=2, j=1, k=2 EKRANA YAZILIR. k++'dan
i=2, j=1, k=3, 3<3 koşul sağlanmaz, döngü sonlanır.
j++'dan
i=2, j=2, k=1
i=2, j=2, k=1 EKRANA YAZILIR. k++'dan
i=2, j=2, k=2
i=2, j=2, k=2 EKRANA YAZILIR. k++'dan
i=2, j=2, k=3, 3<3 koşul sağlanmaz, döngü sonlanır.
j++'dan
i=2, j=3 olur, 3<3 koşul sağlanmaz döngü sonlanır.
i++'dan
i=3 olur, 3<=2 değildir, döngü sonlanır.
```

Bu durumda k döngüsü 2X2X3 defa, j döngüsü 2X3 defa, i döngüsü ise 2 defa dönecektir.

5.2.3 while Döngüsü

Tekrarlam deyimidir. Bir veya birden fazla deyim `while` kullanarak da tekrar edilebilir. Genel yazım biçimi

```
while(koşul)
{
...
deyim;
deyim;
...
};
```

- Döngünün koşulu hemen `while`'dan sonra verilir. Koşul olumlu olduğu sürece tekrar işlemi devam eder.
- `while` döngüsünde birden fazla koşul da kullanılabilir. Bu durumda mantıksal operatörler ile bu koşullar birleştirilmelidir.
- Döngü içerisinde sadece bir deyim kullanılacaksa `{}` parantezlerin kullanımına gerek yoktur. Birden fazla deyim kullanımı durumunda `{}` karakterleri kullanılır.
- **Koşulda kullanılan değişkene, yani sayaç değişkenine kesinlikle başlangıç değeri verilmelidir. Herhangi bir değişkene başlangıç değeri verilmemesi durumunda, o değişkenin içeriği rastale değer alır. Bu değer koşulu olumsuz kılırsa, hiç döngü içerisine giremez.**
- **Koşul içerisinde kullanılan değişkenin yani sayaç değişkeninin attırımı döngü içerisinde yapılmalıdır. Yapılması unutulursa döngü sonsuz çevirime girer.**

```
int k;  
k=1;  
while(k<2)  
    printf("k=%d",k);
```

Bu programda sonsuz döngüye girilir. Çünkü hiçbirzaman `k` sayaç değişkeni 2 değerini almaz, başka bir deyişle 2'den büyük olamaz. `k` değişkeni herzaman 1 olur. Çünkü `k` sayaç değişkenine attırım işlemi yapılmamıştır. Bu durumda `k` değişkeni `k++` işlemi ile döngü içerisinde arttırılmalıdır.

`while` döngüsüne girmeden önce koşulun sağlanıp sağlanmadığına bakılır. Eğer koşul sağlanır ise döngüye girilir, işlemler yapılır. Ardından sayaç değişkeninin değeri istenilen sayıda arttırılır ve tekrar koşula bakılır. Bu işlemler koşul sağlanmadığı zaman durur ve döngü sonlanır.

```
char kr;  
.  
.  
while((kr=getch()) != ' ');  
.  
.
```

Bu programda `kr` değişkenine kullanıcının klavyeden girdiği karakter atanır ve bu karakter boşluk karakteri ile karşılaştırılır. Kullanıcının girdiği karakter boşluk karakteri olana kadar kullanıcıdan karakter girilmesi istenir. Kullanıcı boşluk karakterini girdiğinde koşul sağlanmaz ve döngü sonlanır.

5.2.4 do while Döngüsü

Bu deyim `while`'dan farkı, koşula döngü sonunda bakılır. Dolayısı ile en az bir kere döngüye girilir.

Koşul olumlu olduğu sürece çevrim yinelenir. Birden fazla koşul verilmesi durumda, mantıksal operatörler ile bu koşullar birleştirilmelidir. Buna göre `do while` yapısı çevrim yinelenir. Birden fazla koşul verilmesi durumda, mantıksal operatörler ile bu koşullar birleştirilmelidir. Buna göre `do while` yapısı

```
do
{
..
deyim;
..
}while(kosul);
```

şeklindedir.

- ; karakteri `while`'dan hemen sonra verilmelidir.
- Sayaç değişkeni başlangıç değeri ve sayaç değişkeni arttırım işlemleri `while` döngüsünde olduğu gibidir.
- Bu döngü tipi daha çok standart girdiden veri istenmesi durumunda kullanılır. Çünkü bu durumda en az bir kere döngüye girilmesi gerekir ki, kullanıcının girdiği değer ile koşul karşılaştırılması yapılabilir. Sözelimi,

```
do
{
printf("bir sayı giriniz, çıkmak için 0'a basınız\n");
scanf("%d",&sayi);
}while(sayi!=0);
```

Burada sayaç değişkeni `sayi` isimli değişkendir. Burada sayaç değişkenine bir değer atamak zorunda değiliz. Sayaç değişkeni değerini döngü içerisinde alacaktır. Bu program kullanıcıdan bir değer girmesini ister ve kullanıcı 0 değerini girene kadar da istemeye devam eder. Kullanıcının ne kadar sayıda değer girmek istediğini bilmediğimiz için bu yapıyı kullandık. Sayaç değişkenine arttırım verecek olsak belli bir sayıdan söz etmiş olacaktık. Bu da programımızı esnek olmamasına neden olacaktı.

5.2.5 continue Yapısı

Bir döngü içerisinde `continue` deyimi ile karşılaşırsa, ondan sonra gelen deyimler veya fonksiyonlar atlanır ve döngü bir sonraki çevirime girer. Sözelimi,

```
for(i=1;i<10;i++)
{
    if(i==5)
        continue;
    printf("i=%d",i);
}
```

`i` sayaç değişkeni 5'e eşit olduğu zaman `continue` deyimi çalışacak ve `i=5` için olan döngü atlanacaktır. Bu durumda sırası ile ekrana 1 2 3 4 6 7 8 9 yazılacaktır.

5.2.6 break Yapısı

Bir döngü içerisinde `break` deyimi ile karşılaşırsa, döngü koşula bakılmaksızın sonlanır ve programın akışı döngüden sonra ilk deyim veya fonksiyona atlar. Özel durumlarda döngüden çıkmak için kullanılır. Sözelimi,

```
do
{
    kr = getchar();
    if(k=='s')
        break;
}while(1);
```

Bu programda `while(1)` sonsuz döngüye işaret eder. Bu döngü sonsuz bir döngüdür. Burada kullanıcıdan bir karakter girilmesi istenir. Kullanıcı `s` karakterini girdiği zaman `if` koşulu sağlanır ve `break` deyimi çalışır ve sonsuz döngü sonlanır.

`break` ve `continue` deyimleri `while`, `do while`, `for`, `switch` deyimlerinden çıkmak için kullanılır. İç içe döngü içerisinde kullanıldığı zaman en içteki döngüden çıkarılır.

5.3 Veri Niteleyicileri

Özel niteliyiciler (`short`, `unsigned`, `long` gibi) değişken tiplerinin önüne gelerek değişik veri tipleri meydana getirirler.

5.3.1 signed

Bütün değişken tipleri öntanımlı olarak **signed** tanımlıdır. Bu şekilde tanımlı sayılar artı veya eksi işaretli olabilir.

5.3.2 unsigned

Özel niteleyicilerden biridir. Bu niteleyici, tam sayı tiplerinin herhangi birinin veya karakter tipindeki değişkenlerin önüne gelebilir. Bunun anlamı yalnızca artı (pozitif) sayılar için çalışacak demektir. Eğer yazılan bir programda kullanılacak değişkenler yalnızca artı değerler alacaksa, değişkenler işaretsiz (unsigned) olarak bildirilebilirler.

5.3.3 short ve long

Normal tamsayı 16 bit ise uzun tamsayı 32 bit uzunluğundadır. Kısa tamsayı 16 biti geçmeyecek uzunluktadır. Bu uzunluklar sabit değerler olmayıp, derleyicilerde donanıma bağlı olarak değişebilir. `sizeof` deyimi kullanılarak bunların bellekte ne kadar yer işgal ettikleri öğrenilebilir.

`long` tipinde olan değişkenleri de 16 bitten daha fazla veri saklamaya gereksinim olduğunda kullanılabilir.

5.4 Değişken Bildirim Yerleri ve Türleri

C programında değişkenler önceden bildirilmelidir. Değişkenin nerede bildirildiği oldukça önemlidir. Eğer bir değişken, fonksiyonların birçoğunda kullanılacaksa genel (global), yalnızca bir fonksiyon içerisinde kullanılacak ise yerel (local) olarak bildirilmelidirler. Bir değişken genel olarak tanımlanırsa programın tümünde tarafından tanınır, yerel tanımlanırsa sadece tanımladığı yapı içerisinde tanınırlar. Sözelimi,

```
main ()
{
    int a,b // a ve b yerel değişken
    a =1, b=2;
}
int fonk(int k)
{
    int d; // d yerel degisken
    return d;
}
```

a ve b değişkenleri `main` fonksiyonu içerisinde tanımlandıkları için yerel değişkenlerdir. Aynı şekilde d değişkeni de `fonk` isimli fonksiyon içerisinde tanımlandığı için yerel değişkenlerdir. a ve b değişkeninin `fonk` fonksiyonu içerisinde bilinmediği gibi, d değişkeni de `main` fonksiyonu içerisinde bilinmez.

```
int a,b // a ve b genel değişken
main ()
{
    a =1,b=2;
}
int fonk(int k)
{
    int d; // d yerel degisken
    return d;
}
```

Bu tanımlama da ise a ve değişkenleri genel (global) değişkenlerdir. Çünkü programın en başında `main` fonksiyonundan önce tanımlanmıştır. Bu durumda bu iki değişken de tüm program boyunca bilinir, `fonk` isimli fonksiyonda da bu değişkenler tanınır. Yani bu değişkenlerin değerleri bu fonksiyon içerisinde yazdırılabilir, işleme sokulabilir. Yerel tanımlama yapılmış olsaydı bu mümkün olmayacaktı.

5.5 Matematiksel Fonksiyonlar

C dilinde bazı matematiksel fonksiyonlar önceden oluşturulmuştur. Trigonometrik, logaritmik fonksiyonlar gibi fonksiyonlar önceden tanımlanmıştır. Ancak bu fonksiyonları kullandıktan sonra ilgili matematik kütüphanesini programa eklemek gerekmektedir. Linux da derleme esnasında ise `-lm` parametresi ile bu kütüphaneye bağlantı oluşturularak derleme yapılması gerekmektedir.

```
#include<stdio.h>
#include<math.h>
int main ()
{
    float x;
    x = 1.45536;
    printf("sin (%f) =%f\n",x,sin(x));
    return 0;
}
```

Sözelimi, bu programın adı `deneme.c` olsun. Bu program `gcc deneme.c -lm` şeklinde derlenmelidir. Bu programda x değerinin sinüs değeri hesaplanır.