Program Kontrolü-Yineleme (İterasyon)

Bu bölümde şunları öğreneceğiz:
■ Sayaç kontrollü yinelemenin temelleri
■ İfadeleri art arda yürütmek için for ve do…while yineleme ifadeleri
■ switch deyimini kullanarak çoklu seçim.
■ Kontrol akışını değiştirmek için break ve continue deyimleri
■ Kontrol ifadelerinde karmaşık koşullar oluşturmak için mantıksal operatörler

C'nin for ve do…while yineleme ifadelerini tanıtacağız. Ayrıca şunları da tanıtalım:
• "switch" ifadesi: çoklu seçim,
• "break" ifadesi: belirli kontrol ifadelerinden hemen çıkmak için
• "continue" ifadesi: bir yineleme ifadesinin gövdesinin geri kalanını atlamak ve ardından döngünün bir sonraki yinelemesine devam etmek için.
• Ayrıca koşulları birleştirmek için kullanılan mantıksal operatörleri tartışacağız.

Yineleme (İterasyon) Temelleri

Çoğu program yineleme (veya döngü) içerir. Bir döngü; döngü devam koşulları doğru iken bilgisayarın tekrar tekrar yürüttüğü bir talimat grubudur. İki yineleme yöntemini tartışalım:
1. Sayaç kontrollü yineleme.
2. Gözcü kontrollü yineleme.
Bölüm 3'te, sayaç kontrollü yinelemenin, gerçekleştirilecek bir talimat grubu için yineleme sayısını saymak üzere bir kontrol değişkeni kullandığını gördük. Kontrol değişkeninin değeri, doğru sayıda yinelemenin tamamlandığını gösterdiğinde, döngü sona erer ve yürütme, yineleme ifadesinden sonraki ifadeyle devam eder. Yinelemelerin kesin sayısı önceden bilinmiyorsa ve döngü her gerçekleştirildiğinde döngü, veri elde eden ifadeler içeriyorsa, yinelemeyi kontrol etmek için gözcü değerleri kullandığımızı Bölüm 3'te gördünüz. Bir gözcü değer, "veri sonunu" belirtir. Gözcü, programa tüm normal veri öğeleri bittikten sonra girilir. Gözcü değerler normal veri öğelerinden farklı olmalıdır. (*Örneğin önceki bölümde gözcü değer olarak -1 kullanmıştık.*)

Sayaç Kontrollü Yineleme

Sayaç kontrollü yineleme şunları gerektirir:
• bir kontrol değişkeninin adı,
• kontrol değişkeninin başlangıç değeri,
• her yinelemede kontrol değişkeninin değiştirildiği artış (veya azalma) değeri,
• döngünün devam edip etmeyeceğini belirlemek için kontrol değişkeninin son değerini test eden döngü devam koşulu.

Şekil 4.1 de, 1'den 5'e kadar sayıları gösteren bir program düşünün. Tanım

```
int counter = 1; // başlatma
```

kontrol değişkenini adlandırır (counter), onu bir tamsayı olarak tanımlar, onun için bellek alanı ayırır ve başlangıç değerini 1 olarak ayarlar.

```c
// fig04_01.c
// Counter-controlled iteration.
#include <stdio.h>

int main(void) {
        int counter = 1; // initialization

        while (counter <= 5) { // iteration condition
                printf("%d ", counter);
                ++counter; // increment
        }
        puts("");
}
```

4.1 | Sayaç kontrollü yineleme.

++counter; //her döngü yinelemesinin sonunda sayacı 1 artırır.
counter <= 5 //kontrol değişkeninin değerinin 5'ten küçük veya eşit olup olmadığını test eder.
Bu while döngüsü, kontrol değişkeni 5'i aştığında sona erer (yani, sayaç 6 olur).

Kayan nokta değerleri yaklaşık değerler olabilir, bu nedenle kayan nokta değişkenleriyle sayma döngülerinin kontrol edilmesi, kesin olmayan sayaç değerlerine ve hatalı sonlandırma testlerine neden olabilir. Bu nedenle sayma döngülerini her zaman **tamsayı** değerlerle kontrol etmelisiniz.

"for" Yineleme İfadesi

for yineleme ifadesi (Şekil 4.2'nin 8-10. satırları), sayaç kontrollü yinelemenin tüm ayrıntılarını ele alır. Okunabilirlik için, **for** ifadesinin başlığını (satır 8) bir satıra sığdırmaya çalışılır. **for** deyimi şu şekilde çalışır:

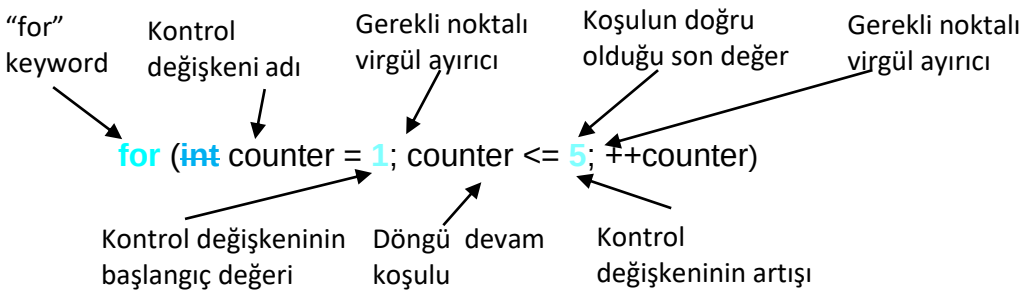
• Yürütülmeye başladığında **for** ifadesi, sayaç kontrol değişkenini tanımlar ve onu 1 olarak başlatır.
• Ardından, döngü devam koşulu counter <= 5 test eder. Sayacın başlangıç değeri 1'dir, yani koşul doğrudur ve for ifadesi, sayacın değerini, yani 1'i görüntülemek için printf ifadesini (satır 9) yürütür.
Daha sonra, for ifadesi, ++counter ifadesini kullanarak sayaç kontrol değişkenini artırır, ardından döngü devam koşulunu yeniden test eder. Kontrol değişkeni artık 2'ye eşittir, dolayısıyla koşul hala doğrudur ve for ifadesi, printf ifadesini yeniden yürütür.
• Bu işlem, kontrol değişkeni sayacı 6 olana kadar devam eder. Bu noktada, döngü devam koşulu yanlıştır ve yineleme sona erer. Program **for**'dan sonraki ilk deyimle (satır 12) çalışmaya devam eder.

NOT: Bu programda "int counter" for döngüsünün dışında tanımlanmalıdır. For döngüsünün içinde çalışmayabilir.

```c
// fig04_02.c
// Counter-controlled iteration with the for statement.
#include <stdio.h>

int main(void) {
// initialization, iteration condition, and increment
// are all included in the for statement header.
        int counter
        for (counter = 1; counter <= 25; ++counter) {
                printf("%d   ", counter);
        }
        puts(""); // outputs a newline
}
```

Aşağıdaki diyagram, Şekil 4.2'nin sayaç kontrollü yineleme için gereken öğelerin her birini belirten for ifadesine daha yakından bakmaktadır. for'un gövdesinde birden fazla deyim varsa, ayraçlar gerekir. Diğer kontrol ifadelerinde olduğu gibi, yalnızca bir ifadeye sahip olsa bile, her zaman bir for ifadesinin gövdesini parantez içine alın.



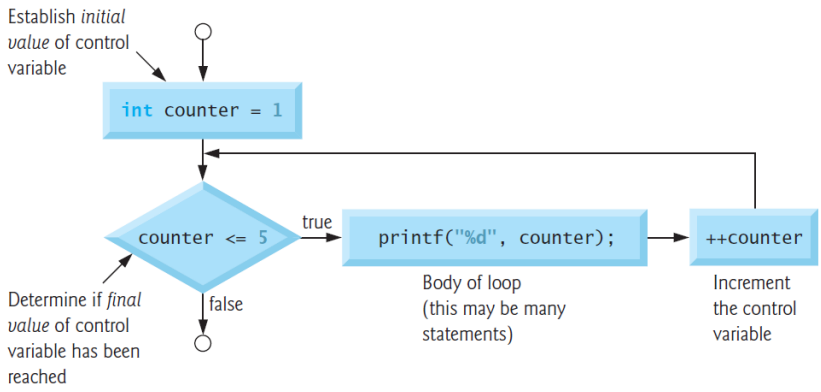
Kontrol değişkenini for ifadesinde ön artırmalı tanımladığınızda, Şekil 4.2'nin 8. satırında olduğu gibi:

```
int counter = 1
for (counter = 1; counter <= 5; ++counter) {
```

Kontrol değişkeni yalnızca döngü sona erene kadar mevcuttur. Bu nedenle, for ifadesinin kapanış sağ parantezinden (}) sonra kontrol değişkenine erişmeye çalışmak bir derleme hatası verir.

Not: Bazı derleyicilerde int tanımlayıcısını for yapısının parantezleri içinde kullanılırsa hata verir. Bu nedenle int tanımlayıcısının for döngüsünden önce tanımlanması gerekir. Bu durumda oluşabilecek hatalar önlenmiş olur. Örneğin kitabınızdaki bazı örneklerde int tanımlayıcısı for döngüsünün parantezleri içerisinde yer almaktadır. Ama bu örnekleri çalıştırırken hata alabilirsiniz. Eğer hata alırsanız bu durumda int tanımlayıcısını döngünün önüne dışına taşıyın.

Şekil 4.2'deki for ifadesi için akış şeması aşağıdadır: Bu akış şeması, başlatmanın bir kez gerçekleştiğini ve artışın for ifadesinden sonra gerçekleştiğini açıkça ortaya koymaktadır.



Aşağıdaki örnekler, bir for ifadesinde kontrol değişkenini değiştirmenin yollarını gösterir.
(NOT:** Kitapta verilen Buradaki **for** ifadelerindeki **int** tanımı programda for döngüsünün dışına alınmalı.**)
1. Kontrol değişkenini 1'den 100'e 1'lik artışlarla değiştirin.

```
int i
for (i = 1; i <= 100; ++i)
```

2. Kontrol değişkenini -1'lik artışlarla (yani 1'lik azalmalar) 100'den 1'e değiştirin.

```
int i
for (i = 100; i >= 1; --i)
```

3. Kontrol değişkenini 7'den 77'ye 7'lik artışlarla değiştirin.

```
int i
for (i = 7; i <= 77; i += 7)
```

4. Kontrol değişkenini -2'lik artışlarla 20'den 2'ye değiştirin.

```
int i
for (i = 20; i >= 2; i -= 2)
```

5. Kontrol değişkenini 2, 5, 8, 11, 14 ve 17 değerleri üzerinden değiştirin.

```
int j
for (j = 2; j <= 17; j += 3)
```

6. Kontrol değişkenini aşağıdaki değer dizisine göre değiştirin: 44, 33, 22, 11, 0.

```
int j
for (j = 44; j >= 0; j -= 11)
```

Şekil 4.3, 2 ile 100 arasındaki çift tamsayıları toplamak için for ifadesini kullanır. Her döngü yinelemesi (8-10 satırları), toplama kontrol değişkeni numarasının mevcut değerini ekler.

NOT: Bu programda "int number;" for döngüsünün dışında tanımlanmalıdır. For döngüsünün içinde (kitapta verildiği gibi) çalışmayabilir.

```
1 // fig04_03.c
2 // Summation with for.
3 #include <stdio.h>
4
5 int main(void) {
6 int sum = 0; // initialize sum
7       int number;
8       for (number = 2; number <= 100; number += 2) {
9               sum += number; // add number to sum
10      }
11
12      printf("Sum is %d\n", sum);
13 }
```

Sonraki örnek, for ifadesini kullanarak bileşik faizi hesaplar. Aşağıdaki problemi göz önünde bulunduralım:
Bir kişi, %5 faiz getiren bir tasarruf hesabına 1000,00$ yatırır. Tüm faizin hesapta mevduatta kaldığını varsayarak, 10 yıl boyunca her yılın sonunda hesaptaki para miktarını hesaplayın ve yazdırın. Bu miktarları belirlemek için aşağıdaki formülü kullanın:

$a = p(1 + r)^n$
burada,
p, yatırılan orijinal miktardır (yani anapara, burada 1000,00 $),
r, yıllık faiz oranıdır (örneğin, %5 için .05),
n, burada 10 olan yıl sayısıdır ve
a, n'inci yılın sonunda yatırılan tutardır.

Çözüm (Şekil 4.4), paranın mevduatta kaldığı 10 yılın her biri için aynı hesaplamayı yapmak üzere sayaç kontrollü bir döngü kullanır. for deyimini 10 kez yürütür, bir kontrol değişkenini 1'den 10'a 1'lik artışlarla değiştirir. C bir üs alma işleci içermez, bu nedenle bunun için Standart kütüphane fonksiyonu pow'u (satır 17) kullanırız. pow(x, y) çağrısı, x'in y'inci kuvvetini hesaplar. Fonksiyon, double veri türünden iki argüman alır. Hesaplamayı tamamladığında, pow bir double değeri döndürür, daha sonra bunu asıl değerle çarparız (satır 17).

NOT: Bu programda "int year;" for döngüsünün dışında tanımlanmalıdır. For döngüsünün içinde (kitapta verildiği gibi) çalışmayabilir.

```
1 // fig04_04.c
2 // Calculating compound interest.
3 #include <stdio.h>
4 #include <math.h>
5
6 int main(void) {
7         double principal = 1000.0; // starting principal
8         double rate = 0.05; // annual interest rate
9
10        // output table column heads
11        printf("%4s%21s\n", "Yil", "Yatırılan miktar");
12        // calculate amount on deposit for each of ten years
13        int year;
14        for (year = 1; year <= 10; ++year) {
15
16                // calculate new amount for specified year
17                 double amount = principal * pow(1.0 + rate, year);
18
19                // output one table row
20                        printf("%4d%21.2f\n", year, amount);
21        }
22 }
```

"pow" ve C'nin diğer matematik fonksiyonlarını kullanmak için <math.h> (satır 4) eklemelisiniz. Eğer başlığı dahil etmezseniz, linker pow fonksiyonunu bulamayacağı için bu program hata verir. "pow" işlevi fonksiyonu iki çift bağımsız değişken gerektirir, ancak year değişkeni bir tamsayıdır. math.h dosyası, derleyiciye "pow" çağrılmadan önce year değerini geçici bir ikili gösterime dönüştürmesini söyleyen bilgileri içerir. Bu bilgi, pow'un fonksiyon prototipinde bulunur. Diğer birçok matematik kütüphanesi fonksiyonlarını da özetlediğimiz Bölüm 5'te fonksiyon prototipleri açıklanacaktır.
C standardı, her kayan nokta türünün minimum boyutlarını belirtir ve double türünün en az float kadar kesinlik sağladığını ve long double türünün en az double kadar kesinlik sağladığını belirtir. C'nin temel sayısal türlerinin ve tipik aralıklarının bir listesi için bkz.
https://en.cppreference.com/w/c/language/arithmetic_types

“switch” Çoklu Seçim İfadesi

Bölüm 3'te **if** tek seçim, **if…else** çift seçim ve **if…. elseif…..else** çoklu seçim ifadelerini tartıştık. Bazen bir algoritma, bir değişkeni veya ifadeyi varsayabileceği her bir tamsayı değeri için ayrı ayrı test eden ve ardından farklı eylemler gerçekleştiren bir dizi karar içerir. Buna çoklu seçim denir. C, bu tür karar vermeyi işlemek için **switch** çoklu seçim deyimini kullanır. **switch** deyimi, bir dizi durum etiketinden, isteğe bağlı bir varsayılan durum ve her durum için yürütülecek ifadelerden oluşur.

Şekil 4.5, bir sınavda kazanılan her farklı harf notu öğrencisinin sayısını saymak için **switch** ifadesini kullanır. Programda, kullanıcı öğrencilerin harf notlarını girer. **while** başlığında (satır 17),

```
while ((grade = getchar()) != EOF)
```

önce parantez içindeki atama (grade = getchar()) yürütülür. **getchar** fonksiyonu(<stdio.h>'den) klavyeden bir karakter okur ve bu karakteri tamsayı değişkeninde saklar.

```
1  // fig04_05.c
2  // Counting letter grades with switch.
3  #include <stdio.h>
4
5  int main(void) {
6     int aCount = 0;
7     int bCount = 0;
8     int cCount = 0;
9     int dCount = 0;
10    int fCount = 0;
11
12    puts("Enter the letter grades.");
13    puts("Enter the EOF character to end input.");
14    int grade = 0; // one grade
15
16    // loop until user types end-of-file key sequence
17    while ((grade = getchar()) != EOF) {
18
19       // determine which grade was input
20       switch (grade) { // switch nested in while
21          case 'A': // grade was uppercase A
22          case 'a': // or lowercase a
23             ++aCount;
24             break; // necessary to exit switch
25          case 'B': // grade was uppercase B
26          case 'b': // or lowercase b
27             ++bCount;
28             break;
29          case 'C': // grade was uppercase C
30          case 'c': // or lowercase c
31             ++cCount;
32             break;
```

```
33            case 'D': // grade was uppercase D
34            case 'd': // or lowercase d
35               ++dCount;
36               break;
37            case 'F': // grade was uppercase F
38            case 'f': // or lowercase f
39               ++fCount;
40               break;
41            case '\n': // ignore newlines,
42            case '\t': // tabs,
43            case ' ': // and spaces in input
44               break;
45            default: // catch all other characters
46               printf("%s", "Incorrect letter grade entered.");
47               puts(" Enter a new grade.");
48               break; // optional; will exit switch anyway
49         } // end switch
50      } // end while
51
52      // output summary of results
53      puts("\nTotals for each letter grade are:");
54      printf("A: %d\n", aCount);
55      printf("B: %d\n", bCount);
56      printf("C: %d\n", cCount);
57      printf("D: %d\n", dCount);
58      printf("F: %d\n", fCount);
59   }
```

4.5 | Switch ile harf notlarını sayma.

EOF (dosya sonu) girmek için tuş kombinasyonları sisteme bağlıdır. Linux/UNIX/macOS sistemlerinde, EOF göstergesi bir satıra tek başına yazılarak girilir.

Ctrl + d

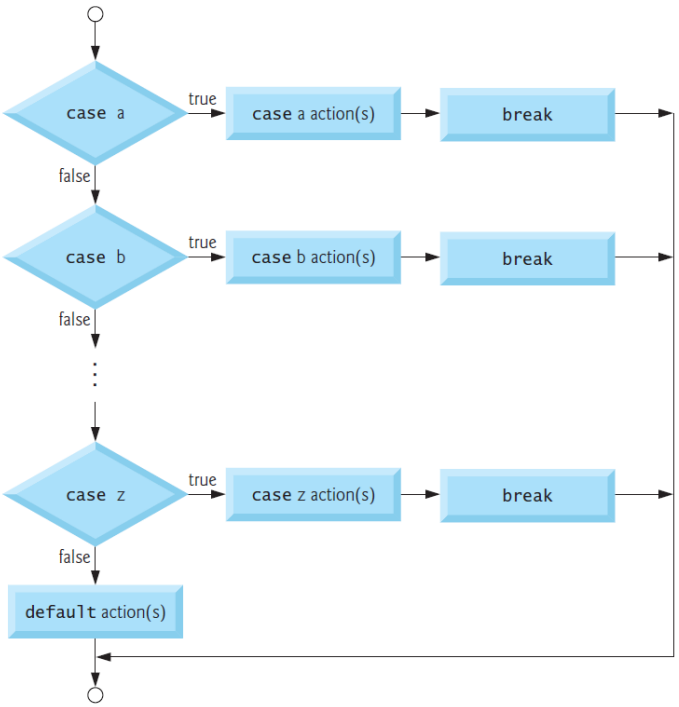
Bu gösterim, aynı anda hem Ctrl tuşuna hem de d tuşuna basmak anlamına gelir. Microsoft Windows gibi diğer sistemlerde, EOF göstergesi yazılarak girilebilir.

Ctrl + z

Ayrıca Windows'ta Enter tuşuna basmanız gerekir. Kullanıcı notları klavyeden girer. Enter tuşuna basıldığında getchar fonksiyonu tarafından karakterler teker teker okunur. Girilen karakter eşit değilse EOF'a, switch deyimi (20–49 satırları) yürütülür.

switch'i parantez içindeki değişken "grade" takip eder. Buna kontrol ifadesi denir. **switch**, bu ifadenin değerini "case" etiketlerinin her biri ile karşılaştırır. Her "case"in bir veya daha fazla eylemi olabilir, ancak belirli bir durumda birden fazla eylemin etrafında parantez gerekli değildir. Kullanıcının not olarak C harfini girdiğini varsayalım. switch C'yi her bir durumla karşılaştırdığında, bir eşleşme meydana gelirse ('C' durumu), bu durum için ifadeler yürütülür. C harfi için, **switch** cCount'u 1 artırır (satır 31), ardından break ifadesi (satır 32) switch'den hemen çıkar ve program kontrolünün **switch** ifadesinden sonraki ilk ifadeyle devam etmesine neden olur. Burada bir **break** deyimi kullanıyoruz, çünkü aksi takdirde bir **switch** deyimindeki **case**'ler birlikte çalışır. **break** ifadeleri olmadan, her eşleşme gerçekleştiğinde, kalan tüm **case** ifadeleri yürütülür. Bir **switch** deyiminde bir **break** deyimine ihtiyaç duyulurken bir **break** deyiminin unutulması bir mantık hatasıdır.
Eşleşme olmazsa, **default case** yürütülür. Bu programda bir hata mesajı görüntüler. Her zaman bir **default case** eklemelisiniz; aksi takdirde, switch ifadesinde açıkça test edilmeyen değerler yok sayılır. **default case**, sizi istisnai koşulları işleme ihtiyacına odaklayarak bunu önlemeye yardımcı olur. Bazen varsayılan işleme gerek yoktur. Bir switch deyimindeki case yan tümceleri ve default case yan tümcesi herhangi bir sırada oluşabilse de, default yan tümceyi en sona yerleştirmek yaygın bir uygulamadır. Varsayılan yan tümce son olduğunda, break ifadesi gerekli değildir. Ancak birçok programcı, diğer durumlarla netlik ve simetri için break ifadesini dahil eder.

Aşağıdaki switch çoklu seçim deyimi akış şeması, her case'in break deyiminin switch deyiminden hemen çıktığını açıkça ortaya koymaktadır.



Şekil 4.5'teki switch ifadesinde, aşağıdaki satırlar programın yeni satır, sekme ve boş karakterleri atlamasına neden olur. Karakterleri birer birer okumak sorunlara neden olabilir. Programın karakterleri okuması için Enter'a basarak bilgisayara göndermelisiniz. Bu, yeni satır karakterini işlemek istediğimiz karakterden sonra girdi değişkenine yerleştirir.

```
case '\n': // ignore newlines,
case '\t': // tabs,
case ' ': // and spaces in input
break;
```

Çoğu zaman, programın düzgün çalışması için bu yeni satır (ve diğer boşluk karakterleri) özellikle göz ardı edilmelidir. Switch ifademizdeki önceki durumlar, girişte her yeni satır, sekme veya boşlukla karşılaşıldığında varsayılan durumdaki hata mesajının yazdırılmasını engeller. Bu örnekteki her girdi, döngünün iki yinelemesine neden olur; birincisi harf notu için ve ikincisi '\n' için. Araya giren ifadeler olmadan birkaç durum etiketinin listelenmesi, her bir durum için aynı eylemlerin gerçekleştiği anlamına gelir.
switch deyimini kullanırken, her durumun yalnızca sabit bir ifadeyi test edebileceğini unutmayın. İfade, sabit bir tamsayı değeri olarak değerlendirilen karakter sabitleri ve tamsayı sabitlerinin herhangi bir kombinasyonu olabilir. Bir karakter sabiti, 'A' gibi tek tırnak içinde belirli bir karakter olarak gösterilebilir. Karakter sabitleri olarak tanınmak için karakterlerin tek tırnak içine alınması gerekir; çift tırnak içindeki karakterler dize olarak tanınır. Tamsayı sabitleri basitçe tamsayı değerleridir. Örneğimizde karakter sabitleri kullandık.