1 Döngüler

Belirli bir iş birden çok kez tekrarlanacaksa, programda bu iş bir kez yazılır ve döngü yapıları o deyim(ler)i istenildiği kadar tekrarlar. C dilinde bu işi yapan üç ayrı yapı vardır:

1. while
2. do . . . while . . .
3. for

1.1 while döngüsü

Bir ya da bir grup deyimin, belli bir koşul sağlandığı sürece tekrarlanması için kullanılan bir denetim yapısıdır. Sözdizimi şöyledir:

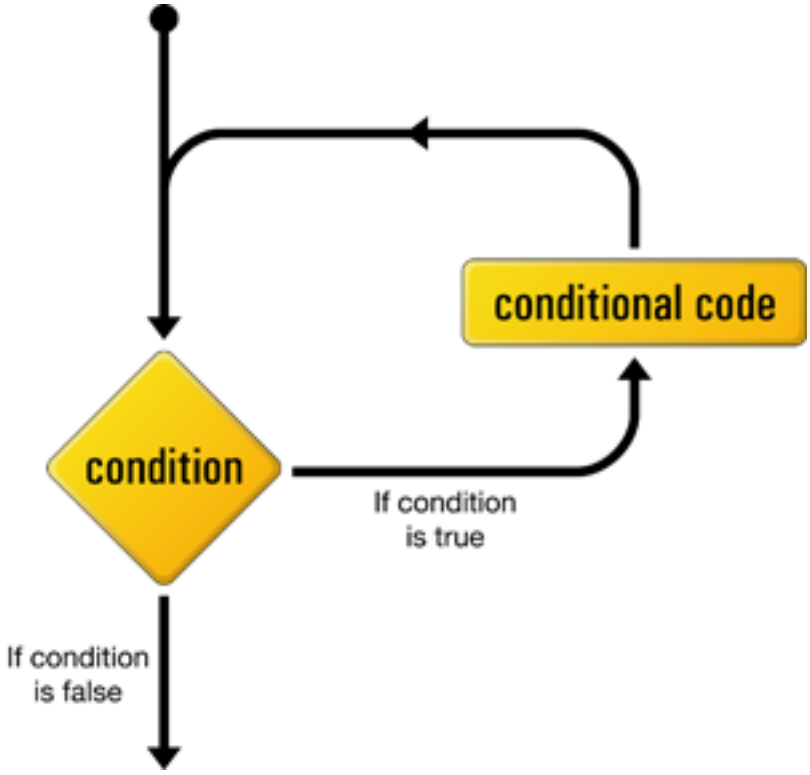
Tanım 1.1.

```
1   while (boolean)
    deyim
```

Deyim sayısı birden çoksa, onları bir blok içine alırız:

Tanım 1.2.

```
    while (boolean)
{
3   deyimler
}
```



Şekil 1.1: while döngüsü

boolean (mantıksal deyim) *true* (doğru) ise deyim ya da blok içindeki deyimler yürütülür. Sonra program akışı başladığı *while* deyimine döner ve tekrar *boolean* mantıksal deyimini denetler. Mantıksal deyim doğru ise deyim ya da blok içindeki deyimler yeniden yürütülür. Bu döngü, mantıksal deyim *false* (yanlış) değerini alana kadar yinelenir. Dolayısıyla, yinelenen deyim(ler)in, mantıksal deyimin değerini sonlu sayıda yineleme sonunda değiştirmesi gerekir. Aksi halde, *sonsuz döngü* dediğimiz olgu ortaya çıkar. Bu durum olunca, döngü deyim(ler)i, kesintisiz devam eder. Program hatası olan *sonsuz döngü*'den sakınmak gerekir.

Aşağıdaki döngü 1, 2, 3, 4, 5 sayılarını yazar:

Program 1.1.

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

int main()
 {
  /**
   * sayı değişkeni
   * int tipi bir değişkendir.
   */
  setlocale(LC_ALL, " ") ;
  int sayi;         // Yazılacak sayıları tutacak değişken
  sayi = 1;         // değişkene verilen ilk değer
  while (sayi < 6) {      // mantıksal deyim
    printf("%d  " , sayi);
    sayi = sayi + 1;    // sayıya 1 ekle
  }
  printf("Son");
}
```

Aşağıdaki program $10! = 1*2*3*4*5*6*7*8*9*10$ çarpansal (faktöryel) değerini bulur ve yazar. Döngü sayacı olan n en büyük değeri olan 10'dan başlayarak, her adımda 1'er azalarak en küçük değeri olan 1 değerine kadar iniyor.

Program 1.2.

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

int faktoryel(int n) {
  int fak;
  fak = 1;
  while (n >= 1) {
    fak *= n;
    n--;
  }
  return fak;
```

```
}

int main() {
  int m;
  printf("Hangi sayının faktoryelini istiyorsunuz= \n ");
  scanf("%d", &m);

  printf(" %d sayısının faktoryeli %d dir \n", faktoryel(m));
  return 0;
}

/**
  10 sayısınınfaktoryeli  3628800 dir
*/
```

1.1.1 while Döngüsünde break deyimi

Bazen while döngüsüne giren program akışını, döngü sona ermeden durdurup, akışı döngüden sonraki ilk deyime götürmek gerekebilir. Bunun için *break;* deyimi kullanılır. Program 1.3 bu işin nasıl yapıldığını gösteriyor.

Program 1.3.

```
#include<stdio.h>

int main()
{
      int i;

  i = 0;
  while ( i < 20 )
      {
            i++;
            if ( i == 10)
                break;
      }
      return 0;
}
```

1.1.2 while Döngüsünde continue deyimi

Bazen *while* döngüsüne giren program akışını, döngünün belirli bir deyiminde durdurup, sonraki deyime geçmesini isteyebiliriz. Bunun için *continue;* deyimi kullanılır. Program 1.4 bu işin nasıl yapıldığını gösteriyor.

Program 1.4.

```
#include<stdio.h>

int main()
{
  int i;

  i = 0;
      while ( i < 20 )
      {
            i++;
            continue;
            printf("Hiç bir iş yapmaz!\n");
      }
      return 0;
}
```

1.2 do ... while ... döngüsü

Bir mantıksal deyim sağlandığı sürece bir blok içindeki deyim(ler)in tekrarlanmasını sağlayan denetim yapısıdır. *while...* yapısından farkı, mantıksal denetimi blok sonunda yapıyor olmasıdır. Dolayısıyla, bu yapıda, bloktaki deyimler en az bir kez çalışır. Sonra denetim olur. Mantıksal-deyim sağlanıyorsa, program akışı döngü başına döner. Bu eylem, mantıksal-deyim `false` olana kadar devam eder. Sonsuz döngüden sakınmak için, sonlu adımda mantıksal-deyimin false değer alması gerekir.

İşlenecek deyim tek ise sözdizimi şöyledir:

Tanım 1.3.

```
do
  deyim
while (boolean);
```

Bu demektir ki, tek deyim { } bloku içine yazılmayabilir.

İşlenecek deyim birden çok ise, sözdizimi şöyledir:

Tanım 1.4.

```
do {
      deyimler
   } while (boolean);
```

Bu demektir ki, birden çok deyim varsa, onlar mutlaka { } bloku içine yazılır. Döngü, sondaki boolean *false* değerini alana kadar tekrarlanır.

Program 1.5 1'den 10'a kadar tamsayıları aynı satıra yazdırıyor.

Program 1.5.

```
#include <stdio.h>

int main() {
  int sayi = 1;
  do {
    printf ("%d \t" , sayi);
    sayi++;
  } while (sayi <= 10);
}
```

```
/**
  1 2 3 4 5 6 7 8 9 10
*/
```

Genel olarak, *while* yapısı *do-while* yapısına dönüştürülebilir. Tabii, bunun tersi de yapılabilir; yani *do-while* yapısı *while* yapısına dönüştürülebilir. Örnek 1.6 programı Örnek 1.5 programına denktir. *do-while* yapısında deyimlerin en az bir kez çalıştığına dikkat edilmelidir.

Program 1.6.

```
#include <stdio.h>

int main() {
  int sayi = 1;
  while (sayi <= 10) {
    printf ("%d \t" , sayi);
    sayi++;
  }
  return 0;
}
```

Program 1.7, do-while döngüsü ile 100e kadar tamsayıların toplamını buluyor.

Program 1.7.

```
#include <stdio.h>

main() {
  int n = 1;
  int toplam = 0;

  do {
    toplam += n;
    n++;
  } while (n <= 100);
  printf("100 e kadar tamsayıların toplamı = %d \n" ,  toplam);
}
```

```
/**
  100 e kadar tamsayıların toplamı = 5050
*/
```

Program 1.8, 100 den geriye doğru 9 ar 9 ar sayıyor.

Program 1.8.

```
#include <stdio.h>

int main() {

  int sayi = 100;
  while (sayi >= 0) {
    printf ("%d \t \n" , sayi);
    sayi = sayi-9;
  }
  return 0;
```

```
/**
100   91   82   73   64   55   46   37   28   19   10   1
*/
```

1.3 For döngüsü

İstenen deyim(ler)in kaç kez tekrarlanacağı biliniyorsa, *for* döngü yapısını kullanmak kolaydır. Tekrarlanacak deyim tek ise sözdizimi şöyledir:

Tanım 1.5.

```
for (sayacın_ilk_değeri; tekrarlama_koşulu; sayaç_değerinin_değişimi)
  deyim
```

Tekrarlanacak deyimler birden çoksa, onlar bir blok içine alınır; sözdizimi şöyle olur:

Tanım 1.6.

```
for (sayacın_ilk_değeri; tekrarlama_koşulu; sayaç_değerinin_değişimi)
{
   deyimler
}
```

Program 1.9, 1'den 100'e kadar tamsayıları topluyor.

Program 1.9.

```c
#include <stdio.h>

main() {
  int i, toplam = 0;

  for (i = 1; i <= 100; i++) {
    toplam = toplam + i;
  }
  printf("100 e kadar tamsayıların toplamı = %d \n" , toplam);
}
```

```c
/**
  100 e kadar tamsayıların toplamı = 5050
*/
```

Bu yapıyı açıklayalım:

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

ifadesindeki i sayaç değişkenidir; *int* tipindendir. Döngünün nereden başlayıp nerede biteceğini belirler. Sayacın ilk değeri, ona başlangıç değerini atayan bir C deyimi ile yapılır. *int* $i = 1$ deyimi, i sayacına ilk değer olarak 1 sayısını atayan bir deyimdir. Sayaç istenilen sayıdan başlatılabilir. Artarak ya da azalarak son değere kadar gider.

$i <= 100$ ifadesi, for döngüsünün tekrarlama koşuludur; bir mantıksal deyimdir. Döngünün her adımında $i <= 100$ olup olmadığı denetlenir. Deyim sağlanıyorsa; yani *true* değerini alıyorsa, döngü sonraki adımı atar. Deyim $false$ değerini alınca for döngüsü biter.

$i++$ ifadesi sayacın değerini her adımda 1 artırır. Tabii, $i++$ yerine $i = i+1$ deyimi de yazılabilir. Ama birinci yazılış CPU'da daha hızlı çalışır. Sayaç bazen başka biçimde artar ya da azalabilir. Örneğin, sayacın 5 er 5 er artması isteniyorsa, $i = i+5$ yazılır. Sayacın 2 şer 2 şer azalması isteniyorsa, $i = i - 2$ yazılır. Sayaç azalarak gidecekse, sayacın başlangıç değeri bitiş değerinden büyük olmalıdır. Özetle, Sayacın ilk değerinden son değerine gidişini sağlayan herhangi bir C deyimi geçerlidir.

Tanım 1.6'de ( ) parantezi içindeki deyimlerden birisi, ikisi ya da her üçü de boş olabilir. Boş deyimler ötekilerden ve birbirlerinden noktalı virgül (;) ile ayrılır. Boş deyimin daima *true* değerine sahip olduğu varsayılır. Dolayısıyla, değişim kuralı boş olduğunda *sonsuz döngüye* girilir. Örneğin $for(;;)$ deyimi sonsuz döngüdür. Bu döngü ancak *break* deyimi ile kesilebilir. Bu deyim $while(true)$ deyimine denktir.

Program 1.10.

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

```
main() {
  int i, toplam = 0;

  for (; ; ) {
    toplam = toplam + i;
    printf("%d ye kadar tamsayıların toplamı = %d \n" , i, toplam);
  }
}
```

```
/**
  Sonsuz döngü ...
*/
```

Program 1.11, 100'den geriye doğru 3 er sayarak, çıkan sayıları topluyor.

Program 1.11.

```
#include <stdio.h>

int main() {

  int sayi = 100;
  int toplam = 0;
  while (sayi >= 0) {
    toplam = toplam + sayi;
    sayi = sayi-3;
  }
  printf("Toplam = %i ", toplam);
  return 0;
}
```

```
/**
  Toplm = 1717
*/
```

Örnek 1.12, 100'den 0'a kadar tamsayıları 5'er atlayarak topluyor. 1.11 örneğinin aksine, sayaç 100'den başlıyor, 5'şer azalarak 0'a gidiyor.

Program 1.12.

```
#include <stdio.h>

#define START  0  /* döngü başlangıcı */
#define ENDING 9  /* döngü sonu */
#define MAX(A,B)  ((A)>(B)?(A):(B))  /* Max macro definition */
#define MIN(A,B)  ((A)>(B)?(B):(A))  /* Min macro definition */

main()
{
int index,mn,mx;
int count = 5;

   for (index = START;index <= ENDING;index++) {
```

```
        mx = MAX(index,count);
        mn = MIN(index,count);
        printf("max = %d ve min = %d\n",mx,mn);
    }
}
```

```
/**
  max = 5  ve  min = 0
  max = 5  ve  min = 1
  max = 5  ve  min = 2
  max = 5  ve  min = 3
  max = 5  ve  min = 4
  max = 5  ve  min = 5
  max = 6  ve  min = 5
  max = 7  ve  min = 5
  max = 8  ve  min = 5
  max = 9  ve  min = 5
*/
```

Uyarı 1.1.

Döngü bloku içinde tanımlanan değişkenler, döngünün yerel değişkenleridir. Onlara döngü bloku dışındaki deyimler erişemez. Ama, döngü bloku dışındaki bir kapsanma alanındaki değişkenlere, o kapsanma alanına erişebilen bütün deyimler erişebilir.

Yerel değişkenlere ilk değerleri atanmalıdır; aksi halde bazı derleyiciler bellek adresinde bulduğu değeri kullanır; koşma hatası doğar. Bazı derleyiciler ise, değişken adı 'xxx' iken;

```
   [Error] 'xxx' undeclared (first use in this function)
```

derleme hatası verir.

Program 1.13'da son printf() metodu *toplam* adlı yerel değişkene erişemiyor; derleme harası oluşuyor.

Program 1.13.

```
#include <stdio.h>

int main() {
  int i;
  for(i=1; i >=10; i++) {
    int toplam ;
    toplam = toplam +i
  }
  printf ("%d \n" , toplam);
}
```

```
9 20  D:\Dev-CppPrj\01week\hata.c [Error] 'toplam' undeclared (first
 use in this function)
```

Program 1.14 programı 1'den 10'a kadar tamsayıları toplamak için yazılmıştır. Program 1.13'nin biraz değişik biçimidir. printf() fonksiyonu döngü bloku içine alınmıştır. Program *gcc* derleyicisinde derleme hatası vermiyor. Ama toplam değişkenine ilk değer atanmadığından, koşma hatası doğuyor. Bu hata önceden görülp düzeltilmezse, uygulamada çok büyük yanlışlar doğurabilir.

Uyarı 1.2. *Unutmayınız, derleme hataları yalnız programı yazana zarar verebilir; çünkü çalışmaz. Ama koşma hataları, onu kullanan herkese zarar verebilir.*

Program 1.14.

```
#include <stdio.h>

int main() {
  int i;
  for(i=1; i <=10; i++) {
    int toplam ;
    toplam = toplam +i;
    i++;
    printf ("%d \n" , toplam);
  }
}
```

```
/**
  0
*/
```

for döngüsünde sayaç değişkenleri `int` tipinden seçilmek zorunda değildir. İyi sıralanmış (complete ordered) her veri tipinden sayaç değişkeni seçilebilir. `char` kümesi iyi sıralanmış bir veri tipidir. Dolayısyla sayaçlar `char` tipi olabilir.

Program 1.15 programı, *char* veri tipinin for yapısında sayaç olarak kullanılabileceğini gösteriyor. Bu döngü alfabenin küçük harflerini birer boşluk ara vererek tek satıra yazar.

Program 1.15.

```
#include <stdio.h>

int main() {
  char ch = 'a';
  do {
    printf ("%c " , ch);
    ch++;
  } while (ch <= 'z');
}
```

```
/**
  a b c d e f g h i j k l m n o p q r s t u v w x y z
*/
```

Program 1.16.

```
#include<stdio.h>

int main()
{
    int n, ilk = 0, ikinci = 1, sonraki, c;

    printf("Kaç terim istiyorsunuz? \n");
    scanf("%d",&n);

    printf("Fibonacci serisinin ilk %d terimi :-\n" ,n);

    for ( c = 0 ; c < n ; c++ )
    {
        if ( c <= 1 )
            sonraki = c;
        else
        {
            sonraki = ilk + ikinci;
            ilk = ikinci;
            ikinci = sonraki;
        }
        printf("%d\n"  ,sonraki);
    }

    return 0;
}
```

```
/**
  Kaç terim istiyorsunuz?
  5
  ibonacci serisinin ilk %d terimi :
  0
  1
  1
  2
  3
*/
```

1.4 for Döngüsü İfadeleri

Normal halde bir for döngüsünde *(birinci ; ikinci ; üçüncü)* ifade diye adlandırılan üç ifade vardır. İfadeler (;) ile birbirlerinden ayrılır. *Birinci ifade* sayacın başlangıç değerini belirler. *İkinci ifade* döngünün ne zaman sona ereceğini belirler. O nedenle , ona *döngü koşulu* denilir. *Üçüncü ifade* sayacın nasıl artacağını ya da azalacağını belirtir. Sayaç artarak ya da azalarak döngü koşulunun dışına çıkınca döngü sona erer.

Bunların varlığı ve kullanımlarıyla ilgili koşullar aşağıda özetlenmiştir.

```
for (sayac = başlangıç; döngü_koşulu; sayacın_değişimi)
```

Kural 1 : *for döngüsünde birden çok koşul kullanılabilir*

Program 1.17.

```
/* () içinde ikinci ifade birden çok koşul içerebilir */
 #include<stdio.h>
int main(){
    int i,j,k;
    for(i=0,j=2,k=1;i<=4;i++){
        printf("%d ",i+j+k);
    }
return 0;
}
```

```
/**
Çıktı:
3 4 5 6 7
```

Kural 2 : *for döngüsünde sayacın başlangıcı yazılmayabilir*

Program 1.18.

```
/* () içinde birinci ifade yazılmayabilir */
 #include<stdio.h>

void main(){
    int i=1;
    for(;i<=4;i++){
        printf("%d ",i);
    }
return 0;
}
```

```
/**
Çıktı:
1 2 3 4
```

Kural 3 : *for döngüsünde sayacın tipi () içine yazılamaz*

Program 1.19.

```
/* Bu program yanlıştır. sayacın veri tipi () içinde belirtilemez */
 #include<stdio.h>
int main(){

    for(int i=0;i<=10;i++){
        printf("%d ",i);
    }
```

```
/**
Çıktı:
1 2 3 4
```

Kural 4 : *for döngüsünde ikinci ifade birden çok koşul içerebilir*

Program 1.20.

```
/* ikinci ifade birden çok koşul içerebilir */
#include<stdio.h>
int main(){

    for(int i=0;i<=10;i++){
        printf("%d ",i);
    }
```

```
/**
Çıktı:
2 2 2
```

Kural 5 : *Çoklu döngü koşulu*

Program 1.21.

```
/* ikinci ifade birden çok koşul içerebilir */
#include<stdio.h>
int main(){
    int i,j=2;

    for(i=0;j>=0,i<=5;i++){
        printf("%d ",i+j);
        j--;
    }
return 0;
}
```

```
/**
Çıktı:
2 2 2 2 2 2
```

Kural 6 : *for döngüsünde ikinci ifade olmayabilir*

Program 1.22.

```
/* ikinci ifade olmayabilir */
#include<stdio.h>
int main(){
    int j;

    for(j=0; ;j++){
        printf("%d ",j);
        if(j>=2)
            break;
    }

return 0;
}
```

```
/**
Çıktı:
0 1 2
```

Kural 7 : *ikinci ifade sayacın başlangıcını belirtebilir*

Program 1.23.

```
/* ikinci ifade sayacın başlangıcını belirtebilir */
 #include<stdio.h>
int main(){
    int i;

    for(;i=0,i<=3 ;i++){
        printf("%d ",i);
    }

return 0;
}
```

```
/**
Çıktı:
Sonsuz döngü
```

Kural 8 : *ikinci ifade sayacın artışını belirtebilir*

Program 1.24.

```
/* ikinci ifade sayacın artışını belirtebilir */
 #include<stdio.h>
int main(){
    int i=0;

    for(;i+=2,i<5 ;i++){
        printf("%d ",i);
    }
return 0;
}
```

```
/**
Çıktı:
2
```

Kural 9 : *Döngü koşulunun sıfır olması*

Program 1.25.

```
/*
ikinci ifade 0 dan farklı ise döngü koşulu doğru olur.
*/
 #include<stdio.h>
int main(){
    int i;

```

```
    for(i=0;-5 ;i++){
        printf("%d ",i);
        if(i==3)
            break;
    }

return 0;
}

/**
Çıktı:
0 1 2 3
```

Kural 10 : *Döngü koşulunun sıfır olması*

Program 1.26.

```
/*
ikinci ifade 0 ise döngü hiçbir adım atmaz.
*/
     int i;

    for(i=5;0 ;i++){
        printf("%d ",i);
    }
return 0;
}

/**
Çıktı:
0 1 2 35
```

Kural 11 : *üçüncü ifade birden çok artış içerebilir*

Program 1.27.

```
/* ikinci ifade birden çok artış içerebilir */
 #include<stdio.h>
int main(){
    int i,j,k;

    for(i=0,j=0,k=0;i<=5,j<=4,k<=3;i++,++j,k+=2){
        printf("%d ",i+j+k);
    }

return 0;
}

/**
Çıktı:
0 4
```

Kural 12 : *üçüncü ifade birden çok artış içerebilir*

Program 1.28.

```
/* ikinci ifade birden çok artış içerebilir */
#include<stdio.h>
void main(){
    int i,j=0;

    for(i=0;i<=3;++i,i++,++j ){
        printf("%d %d ",i,j);
    }
return 0;
}
```

```
/**
Çıktı:
0 0 2 1
```

Kural 13 : *üçüncü ifade olmayabilir*

Program 1.29.

```
/* Üçüncü ifade olmyabilir */
#include<stdio.h>
int main(){
    int i;

    for(i=0;i<=3; ){
        printf("%d ",i++);
    }

return 0;
}
```

```
/**
Çıktı:
0 1 2 3
```

Kural 14 : *Blok simgesi ne zaman gerekir?*

Program 1.30.

```
/* Döngü blokunda tek deyim varsa { } blok prantezi kullanılmayabilir
  */
#include<stdio.h>
int main(){
    int i,j=0;

    for(i=0;i<=3;++i,i++,++j )
        printf("%d %d ",i,j);
    }

return 0;
}
```

```
/**
Çıktı:
0 0 2 1
```

Kural 15 : *Blok simgesi ne zaman gerekir?*

Program 1.31.

```
/* Döngü blokunda tek deyim varsa \{ \} blok prantezi
 kullanılmayabilir */
#include<stdio.h>
int main(){
    int x,y=5;
       for(x=0;x<3;x++)
         if(y>=5)
             printf(" %d",x);
return 0;
}
```

```
/**
Çıktı:
0 1 2
```

Kural 16 : *Döngü bloku boş olabilir*

Program 1.32.

```
/* Döngünün  gövdesi hiç olmayabilir */
#include<stdio.h>
int main(){
    int i;

    for(i=0;i<=10;i++);
    printf("%d",i);

return 0;
}
```

```
/**
Çıktı:
11
```

Kural 17 : *Döngüde { } parantezi blok işlevini görür.*

Program 1.33.

```
/* Döngüde \{ \} parantezi blok işlevini görür. */
#include<stdio.h>
int main(){
    int i;

    for(i=0;i<=2;i++){
          int i=8;
          printf("%d ",i);

    }
    printf("%d",i);

return 0;
}
```

```
/**
Çıktı:
8  8  8 3
```

1.5 Sorular

1. Aşağıdaki döngü ne yapar?

```
for (int n = 1 ;  n <= 10 ;  n++ )
 printf("%i ",  n );
```

2. Aşağıdaki döngü ne yapar?

```
for (int  n = 1 ;  n < 10 ;  n++ )
 printf("%i ",  n );
```

3. Aşağıdaki döngü ne yapar?

```
for (int  n = 1 ;  n >= 0 ;  n-- )
 printf("%i ",  n );
```

4. Aşağıdaki dört döngüyü karşılaştırınız. Herbirinin yaptığı işi ve aralarındaki farkı açıklayınız. Hangisi en kötü programcılık örneği sayılmalıdır?

5.
```
  for (int n = 1; n <= 10; n++) {
    printf("%d ", 2*n );
                  }
```

6.
```
  for (int n = 2; n <= 20; n = n + 2) {
  printf("%i ",  n );
                  }
```

7.
```
    for (int n = 2; n <= 20; n++) {
    if ( n % 2 == 0 ) //  n çift mi?
    printf("%i ",  n );
                  }
```

8.
```
                  for (int n = 1; n <= 1; n++) {
                    printf("2 4 6 8 10 12 ");
                    printf("14 16 18 20");
                  }
```

9. 1 + 2 + 3 + ... serisinin ilk 1000 teriminin toplamını bulan bir C programı yazınız.

10. 20 + 21 + 22 + ... serisinin ilk 100 teriminin toplamını bulan bir C programı yazınız.

11. 1! + 2! + 3! + ... serisinin ilk 10 teriminin toplamını bulan bir C programı yazınız.

12. *while* döngüsü ile *do-while* döngüsü arasındaki fark nedir?

13. *for* döngüsü ne zaman kullanılabilir?

14. *for* döngüsünde sayaç hangi veri tip(ler)inden olabilir?

15. *for* döngüsünde sonsuz döngü ne zaman doğar?

16. *while* döngüsünde sonsuz döngü ne zaman doğar?

17. *do-while* döngüsünde sonsuz döngü ne zaman doğar?

18. Aşağıdaki *for* döngüsü ne yapar?

```
int N;  for ( N = 3;  N <= 36;  N = N + 3 ) {
      printf("%i ", N );
  }
```

19. Aşağıdaki *for* döngüsü ne yapar?

```
int N;  for (N = 3;  N <= 36;  N++ ) {
      if ( N % 3 == 0 )
          printf("%i ", N );
  }
```

20. Blok nedir? C dilinde blok nerelerde kullanılır?

21. 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54 57 60 sayılarını yazan bir *for* döngüsü yazınız.

22. 5 10 15 20 25 30 35 40 45 50 55 60 sayılarını yazan bir *while* döngüsü yazınız.

23. 10 20 30 40 50 60 sayılarını yazan bir *do-while* döngüsü yazınız.

24. Örnek 1.34 programının çıktısı nedir?

Program 1.34.

```
#include <stdio.h>

int main() {
  int N;
  N = 1;
  while (N <= 24) {
    N = 2 * N;
    printf("%i ", N );
  }
}
```

1.6 Alıştırmalar

1. 2'nin katlarını 65536 olana kadar yazdıran. Program 1.35'i çözümleyiniz.

   Program 1.35.

```
#include <stdio.h>

main()
{
  long int toplam;
  toplam = 1;
  while(toplam <= 33000) {
    toplam += toplam ;
    printf("Toplam = %ld \n" , toplam);
    }
}
```

2. 20'ye kadar sayıların küplerini hesaplayan Program 1.36'yi çözümleyiniz.

   Program 1.36.

```
#include <stdio.h>

main() {
  int n, kup;
  n = 0;
  while( n++ <20 ) {
    kup = n*n*n;
    printf(" %2d %5d \n" , n,kup);
  }
}
```

3. Sayakları, küçük ve büyük harfleri yazan Program 1.37'i çözümleyiniz.

   Program 1.37.

```
#include <stdio.h>
main()
{
   int i; char c ;
   printf("\nSAYILAR :\n"); i=48 ;
     while(i<58)
       {
          printf("%c",(char)i);
          i++ ;
       }
i=65;
printf("\nBuyuk Harfler :\n");
while (i<122)
  printf("%c", (char)i++);
}
```

```
/**
SAYILAR :
0123456789
Buyuk Harfler :
ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxy
*/
```

4. Bütün ascii karakterlerini yazan Program 1.38'i çözümleyiniz.

   Program 1.38.

```
#include<stdio.h>
int main()
{
int i;
for (i = 0; i < 128; i++)
   {
    printf("%d = %c\n", i, i);
   }
}
```

5. Bütün ascii karakterlerini while döngüsü ile yazdırınız.

6. Bütün ascii karakterlerini do-while döngüsü ile yazdırınız.

7. 90 dereceye kadar 5'er artırarak deecelerin tanjantlarını bulunuz.

   Çözüm:

   Program 1.39.

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

main(){
     float pi=3.141592654;
     double aci=0.0;

     do {
          printf("aci=%f \t tanjanti=%f\n",aci,tan(aci*pi/180.0));
          aci+=5.0;
     }while(aci<=90);
}
```

```
/**
aci=0.000000    tanjanti=0.000000
aci=5.000000    tanjanti=0.087489
aci=10.000000     tanjanti=0.176327
...
aci=80.000000     tanjanti=5.671283
aci=85.000000     tanjanti=11.430058
aci=90.000000     tanjanti=-22877332.428856
*/
```

8. **Soru:** *while* döngüsü ile *do ... while ...* döngüsü arasındaki fark nedir?

   Yanıt: *while* döngüsü önce mantıksal deyimi denetler. Doğru ise döngü deyimlerini koşturur; değilse *while* yapısından sonraki deyime geçer.

   do ... while ... döngüsünde ise, döngü deyim(ler)i en az bir kez koşturulur; sonra mantıksal deyim denetlenir. *false* ise baştaki *do* deyimine döner ve deyim(ler) tekrar eder; doğru ise tekrarlama biter; program akışı *do ... while ...* yapısının dışına çıkar, yapıdan sonraki ilk deyime geçer.

9. Program 1.13'yı düzeltiniz.

10. Program 1.14'yı düzeltiniz.

11. Bütün ascii karakterlerini while döngüsü ile yazdırınız.

12. Bütün ascii karakterlerini do-while döngüsü ile yazdırınız.

13. Kullanıcının gireceği 100' den küçük bir tamsayıdan küçük olan ve 3 sayağı içeren bütün sayıları listeleyen bir C programı yazınız.

    Çözüm :

    Program 1.40.

```
#include <stdio.h>
int main() {

  int i,n;
  printf("100'den küçük bir tamsayı giriniz :  \n");
  scanf("%d"  ,&n);
  for(i=1; i<=n; i++) {
    if((i/10 == 3)||(i%10 == 3))
      printf("%d  "  ,i);
  }
  return 0;
}
```

```
/**
100'den küçük bir tamsayı giriniz :
99
3  13  23  30  31  32  33  34  35  36  37  38  39  43  53  63
 73  83  93
*/
```