While loop

The **while** statement is the simplest of the four loops that C++ provides, and it has a definition very similar to that of an if statement:

```
while (expression)
  statement;
```

A while statement is declared using the **while** keyword. When a while statement is executed, the expression is evaluated. If the expression evaluates to true (non-zero), the statement executes.

However, unlike an if statement, once the statement has finished executing, control returns to the top of the while statement and the process is repeated.

Let's take a look at a simple while loop. The following program prints all the numbers from 0 and 9:

```
#include <stdio.h>

int main(void)
{
  int count = 0;
  while (count < 10)
  {
    printf("%d ",count);
    count++;
  }
  printf("done!\n");
  return 0;
}
```

Let's take a closer look at what this program is doing. First, *count* is initialized to 0. 0 < 10 evaluates to true, so the statement block executes. The first statement prints 0, and the second increments *count* to 1. Control then returns back to the top of the while statement. 1 < 10 evaluates to true, so the code block is executed again. The code block will repeatedly execute until count is 10, at which point 10 < 10 will evaluate to false, and the loop will exit.

It is possible that a while statement executes 0 times. Consider the following program:

```
#include <stdio.h>

int main(void)
{
  int count = 15;
  while (count < 10)
  {
    printf("%d ",count);
    count++;
  }
```

```
  printf("done!\n");
  return 0;
}
```

The condition 15 < 10 immediately evaluates to false, so the while statement is skipped. The only thing this program prints is **done!**

Infinite loops

On the other hand, if the expression always evaluates to true, the while loop will execute forever. This is called an ***infinite loop***. Here is an example of an infinite loop:

```
#include <stdio.h>

int main(void)
{
  int count = 0;
  while (count < 10) // this condition will never be false
  {
    printf("%d ",count); // this line will repeatedly execute
  }
  return 0;
}
```

Because *count* is never incremented in this program, *count* < 10 will always be true. Consequently, the loop will never terminate, and the program will print "0 0 0 0 0 ..." forever.

We can declare an intentional infinite loop like this:

```
#include <stdio.h>

int main(void)
{
  while (1)
  {
    // this loop will execute forever
  }
  return 0;
}
```

The only way to exit an infinite loop is through a ***return*** statement, a ***break*** statement, an ***exit*** statement, a ***goto*** statement, an exception being thrown, or the user killing the program.

Loop variables

Often, we want a loop to execute a certain number of times. To do this, it is common to use a loop variable, often called a ***counter***. A ***loop variable*** is an integer variable that is declared for the sole purpose of counting how many times a loop has executed. In the examples above, the variable *count* is a loop variable. Loop variables are often given simple names, such as *i*, *j*, or *k*.

Each time a loop executes, it is called an ***iteration***.

Consider the program that for a given *n* finds the sum:

$$S = \frac{1}{1\cdot 2} + \frac{1}{2\cdot 3} + \ldots + \frac{1}{n\cdot(n+1)}$$

```
#include <stdio.h>

int i, n;
double s;

int main(void)
{
  scanf("%d",&n);
  s = 0; i = 1;
  while(i <= n) // loop executes (iterates) n times
  {
    s += 1.0 / i / (i+1);
    i++;
  }
  printf("%lf\n",s);
  return 0;
}
```

Realization with for loop:

```
#include <stdio.h>

int i, n;
double s;

int main(void)
{
  scanf("%d",&n);
  s = 0;
  for(i = 1; i <= n; i++)
    s += 1.0 / (i * (i+1));
  printf("%.6lf\n",s);
  return 0;
}
```

Nested loops

It is also possible to ***nest*** loops inside of other loops. In the following example, the inner loop and outer loops each have their own counters. However, note that the loop expression for the inner loop makes use of the outer loop's counter as well!

```
#include <stdio.h>

int main(void)
{
 int outer = 1;
 while (outer <= 5)
 {
   // loop between 1 and inner
   int inner = 1;
```

```
    while (inner <= outer)
      printf("%d ",inner++);

    // print a newline at the end of each row
    printf("\n");
    outer++;
   }
   return 0;
}
```

Find the sum of digits of a number

```
#include <stdio.h>

int n, s;

int main(void)
{
  scanf("%d",&n);
  s = 0;
  while(n > 0)
  {
    s = s + n % 10;
    n = n / 10;
  }
  printf("%d\n",s);
  return 0;
}
```

Read till the end of file

Next program reads two integers and prints their sum. We read the input data till the end of file. Function ***scanf*** returns the number of successfully read arguments.

```
i = scanf("%d",&a); // i is assigned 1
i = scanf("%d %d %d",&a,&b,&c); // i is assigned 3
```

When we shall reach the end of file, ***scanf*** returns -1.

End Of File symbol (EOF) can be entered from the keyboard as ^z (Ctrl - *z*).

```
#include <stdio.h>

int a, b;

int main(void)
{
  while(scanf("%d %d",&a,&b) == 2)
    printf("%d\n",a+b);
  return 0;
}
```

Two dimensional loop

Let's print the multiplication table *n* * *n*.

```
#include <stdio.h>

int i, j, n;
```

```c
int main(void)
{
  scanf("%d",&n);
  for(i = 1; i <= n; i++)
  {
    for(j = 1; j <= n; j++)
      printf("%2d ",i * j);
    printf("\n");
  }
  return 0;
}
```