**Iteration statements (loops).**

Loops repeat a statement a certain number of times, or while a condition is fulfilled. They are introduced by the keywords *while*, *do*, and *for*.

**The while loop**

The simplest kind of loop is the *while-loop*. Its syntax is:

*while (expression) statement*

The *while-loop* simply repeats statement while expression is true. If, after any execution of statement, expression is no longer true, the loop ends, and the program continues right after the loop. For example, let's have a look at a countdown using a while-loop:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 | // custom countdown using while  #include <iostream>  using namespace std;  int main ()  {  int n = 10;  while (n>0) {  cout << n << ", ";  --n;  }  cout << "liftoff!\n";  } | 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, liftoff! | [Edit & Run](https://cplusplus.com/doc/tutorial/control/) |

The first statement in main sets *n* to a value of *10*. This is the first number in the countdown. Then the while-loop begins: if this value fulfills the condition *n>0* (that *n* is greater than zero), then the block that follows the condition is executed, and repeated for as long as the condition (n>0) remains being true.

The whole process of the previous program can be interpreted according to the following script (beginning in *main*):

1. *n* is assigned a value
2. The *while* condition is checked (n>0). At this point there are two possibilities:
   * condition is true: the statement is executed (to step 3)
   * condition is false: ignore statement and continue after it (to step 5)
3. Execute statement:  
   *cout << n << ", ";  
   --n;*  
   (prints the value of n and decreases n by 1)
4. End of block. Return automatically to step 2.
5. Continue the program right after the block:  
   print *liftoff!* and end the program.

A thing to consider with while-loops is that the loop should end at some point, and thus the statement shall alter values checked in the condition in some way, so as to force it to become false at some point. Otherwise, the loop will continue looping forever. In this case, the loop includes --n, that decreases the value of the variable that is being evaluated in the condition (n) by one - this will eventually make the condition (n>0) false after a certain number of loop iterations. To be more specific, after 10 iterations, n becomes 0, making the condition no longer true, and ending the while-loop.

Note that the complexity of this loop is trivial for a computer, and so the whole countdown is performed instantly, without any practical delay between elements of the count (if interested, see [sleep\_for](https://cplusplus.com/sleep_for) for a countdown example with delays).

**The do-while loop**

A very similar loop is the *do-while loop*, whose syntax is:

*do statement while (condition);*

It behaves like a *while-loop*, except that condition is evaluated after the execution of statement instead of before, guaranteeing at least one execution of statement, even if condition is never fulfilled. For example, the following example program echoes any text the user introduces until the user enters goodbye:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 | // echo machine  #include <iostream>  #include <string>  using namespace std;  int main ()  {  string str;  do {  cout << "Enter text: ";  getline (cin,str);  cout << "You entered: " << str << '\n';  } while (str != "goodbye");  } | Enter text: hello  You entered: hello  Enter text: who's there?  You entered: who's there?  Enter text: goodbye  You entered: goodbye | [Edit & Run](https://cplusplus.com/doc/tutorial/control/) |

The *do-while* loop is usually preferred over a *while-loop* when the statement needs to be executed at least once, such as when the condition that is checked to end of the loop is determined within the loop statement itself. In the previous example, the user input within the block is what will determine if the loop ends. And thus, even if the user wants to end the loop as soon as possible by entering goodbye, the block in the loop needs to be executed at least once to prompt for input, and the condition can, in fact, only be determined after it is executed.

**The for loop**

The *for loop* is designed to iterate a number of times. Its syntax is:

*for (initialization; condition; increase) statement;  
//instead of increase we can use decrease also*

Like the *while-loop*, this loop repeats statement while condition is true. But, in addition, the *for loop* provides specific locations to contain an initialization and an increase expression, executed before the loop begins the first time, and after each iteration, respectively. Therefore, it is especially useful to use counter variables as condition.

It works in the following way:

1. initialization is executed. Generally, this declares a counter variable, and sets it to some initial value. This is executed a single time, at the beginning of the loop.
2. condition is checked. If it is true, the loop continues; otherwise, the loop ends, and statement is skipped, going directly to step 5.
3. statement is executed. As usual, it can be either a single statement or a block enclosed in curly braces {}.
4. increase is executed, and the loop gets back to step 2.
5. the loop ends: execution continues by the next statement after it.

Here is the countdown example using a for loop:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 | // countdown using a for loop  #include <iostream>  using namespace std;  int main ()  {  for (int n=10; n>0; n--) {  cout << n << ", ";  }  cout << "liftoff!\n";  } | 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, liftoff! | [Edit & Run](https://cplusplus.com/doc/tutorial/control/) |

The three fields in a *for-loop* are optional. They can be left empty, but in all cases the semicolon signs between them are required. For example, for (;n<10;) is a loop without *initialization* or *increase* (equivalent to a while-loop); and for (;n<10;++n) is a loop with *increase*, but no *initialization* (maybe because the variable was already initialized before the loop). A loop with no *condition* is equivalent to a loop with true as condition (i.e., an infinite loop).

Because each of the fields is executed in a particular time in the life cycle of a loop, it may be useful to execute more than a single expression as any of *initialization*, *condition*, or *statement*. Unfortunately, these are not statements, but rather, simple expressions, and thus cannot be replaced by a block. As expressions, they can, however, make use of the comma operator (,): This operator is an expression separator, and can separate multiple expressions where only one is generally expected. For example, using it, it would be possible for a for loop to handle two counter variables, initializing and increasing both:

|  |  |  |
| --- | --- | --- |
| 1 2 3 4 | for ( n=0, i=100 ; n!=i ; ++n, --i )  {  // whatever here...  } |  |

This loop will execute 50 times if neither *n* or *i* are modified within the loop:  
  
https://cplusplus.com/doc/tutorial/control/for_loop.png  
  
*n* starts with a value of 0, and *i* with 100, the condition is *n!=i* (i.e., that n is not equal to i). Because *n* is increased by one, and *i* decreased by one on each iteration, the loop's condition will become false after the 50th iteration, when both *n* and *i* are equal to 50.

**Range-based for loop**

The *for-loop* has another syntax, which is used exclusively with ranges:

*for ( declaration : range ) statement;*

This kind of for loop iterates over all the elements in *range*, where *declaration* declares some variable able to take the value of an element in this range. Ranges are sequences of elements, including arrays, containers, and any other type supporting the functions *begin* and *end;* Most of these types have not yet been introduced in this tutorial, but we are already acquainted with at least one kind of range: *strings*, which are sequences of characters.

An example of range-based for loop using *strings*:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 | // range-based for loop  #include <iostream>  #include <string>  using namespace std;  int main ()  {  string str {"Hello!"};  for (char c : str)  {  cout << "[" << c << "]";  }  cout << '\n';  } | [H][e][l][l][o][!] | [Edit & Run](https://cplusplus.com/doc/tutorial/control/) |

Note how what precedes the colon (*:*) in the for loop is the declaration of a *char* variable (the elements in a *string* are of type *char*). We then use this variable, *c*, in the statement block to represent the value of each of the elements in the *range*.

This loop is automatic and does not require the explicit declaration of any counter variable.

Range based loops usually also make use of type deduction for the type of the elements with *auto*. Typically, the *range-based loop* above can also be written as:

|  |  |  |
| --- | --- | --- |
| 1 2 | for (auto c : str)  cout << "[" << c << "]"; |  |

Here, the type of *c* is automatically deduced as the type of the elements in *str*.

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | // this\_thread::sleep\_for example  #include <iostream> // std::cout, std::endl  #include <thread> // std::this\_thread::sleep\_for  #include <chrono> // std::chrono::seconds  #include <windows.h>  #include<conio.h> //for \_getch()  using namespace std;  int main()  {  string line = "=";  string unit;  cout << "Press enter to launch." << endl;  cin.get();  for (int i = 0; i <= 100; ++i)  {  cout << unit << endl;  cout << "\x1B[2J\x1B[H"; //clean screen  unit += line;  cout << i << "%" << endl;  Sleep(8);  }  std::cout << "Press enter to launch." << endl;  std::cin.get();  int k = 100;  do {  std::cout << "\033c";  k--;  std::cout << k << "%\n"; //clean screen  std::this\_thread::sleep\_for(std::chrono::milliseconds(80));  //or just seconds instead of milliseconds  } while (k > 0);  cout << "Press enter to continue." << endl;  \_getch();  cout << "Completed\n";  system("cls"); //clean screen  system("pause");  return 0;  } | \*check the program for useful features | [Edit & Run](https://cplusplus.com/doc/tutorial/control/) |

\*run the program above.