Programming Fundamentals

Muhammad Ateeq

[Updated: 06-03-2023]

Iterations/Loops

Handling Inputs

So far we have used cout to output on screen. How do we take input in C++?
In C++, input can be taken from the user using the standard input stream std::cin. The cin object is defined in the iostream header file and is used to read input from the standard input device (usually the keyboard).

To read input from the user, you need to use the >> operator, also known as the extraction operator. The >> operator is used to extract values from the input stream and store them in variables. For example, to read an integer value from the user and store it in a variable x, you can use the following code:

```
int x;
cin >> x;
```

Similarly, you can read other types of values like floating-point numbers and strings using the >> operator. For example:

```
double y;
cin >> y;

string s;
cin >> s;
```

Note that when reading a string, cin will read only one word (i.e., a sequence of characters separated by whitespace). If you want to read a full line of input, you can use the std::getline() function instead. For example:

```
string line;
getline(cin, line);
```

This will read a full line of input (including whitespace) and store it in the line variable.

Building Blocks of Logic

[Repeat] What is three main building blocks of logic used to write computer programs?
Sequence, selection, and iteration are three fundamental programming concepts that are used to control the flow of execution in a program:

Sequence: Sequence refers to the order in which instructions are executed in a program. In a sequential program, each instruction is executed in order, one after the other.
Selection: Selection is the process of making decisions in a program based on certain conditions. This is typically done using conditional statements such as "if-else" statements, which allow the program to execute different instructions based on whether a condition is true or false.
Iteration: Iteration, also known as looping, is the process of repeating a sequence of instructions a certain number of times, or until a certain condition is met. This is typically done using loop structures such as

"while" loops or "for" loops, which allow the program to repeat a block of code until a certain condition is met.

Together, these three concepts form the basic building blocks of most computer programs and allow programmers to control the flow of execution in a program based on certain conditions and requirements.

We have already seen what "sequence" and "decision/selection" are. What is iteration though?
In C++, iteration or loop refers to a programming structure that allows you to repeatedly execute a block of code until a certain condition is met. The loop construct provides a convenient way to perform repetitive tasks with minimal code.

There are three types of loops in C++:

for loop: The for loop executes a block of code repeatedly until a specified condition is met. The syntax of a for loop is as follows:

```
for (initialization; condition; update) {
   // code to be executed
}
```

while loop: The while loop executes a block of code repeatedly as long as the specified condition is true. The syntax of a while loop is as follows:

```
while (condition) {
   // code to be executed
}
```

do-while loop: The do-while loop is similar to the while loop, but the condition is tested at the end of the loop. This means that the block of code is executed at least once, even if the condition is false. The syntax of a do-while loop is as follows:

```
do {
   // code to be executed
} while (condition);
```

Loops are an essential programming construct that allows you to write efficient and concise code for repetitive tasks.

Why are iterations/loops/repetitions needed in programming?
Iteration or loop is an essential programming construct that is needed for various reasons, some of which are:

To execute repetitive tasks: Loops allow you to execute a block of code repeatedly, which is useful when you need to perform the same task multiple times with different inputs or conditions.
To process collections of data: Loops are commonly used to process collections of data such as arrays, lists, and other data structures. By using loops, you can perform operations on each element of the collection.
To simplify code: Loops allow you to write concise code by reducing redundancy. Instead of writing the same code multiple times, you can use a loop to perform the same task repeatedly.

To implement complex algorithms: Many complex algorithms require repetitive execution of certain steps. Loops make it easier to implement such algorithms by providing a way to execute the same steps repeatedly.

Overall, iteration or loop is an important programming construct that makes it possible to write efficient, concise, and flexible code for a wide range of applications. Without loops, programming tasks that involve repetitive operations would be much more time-consuming and error-prone.

How does for loop work in C++?

A for loop in C++ allows you to execute a block of code repeatedly until a certain condition is met. It is typically used when you know the number of times the loop should be executed in advance. Here is a simple example that demonstrates how a for loop works in C++:

```
include <iostream>
using namespace std;
int main() {
    for (int i = 0; i < 5; i++) {
        cout << "The value of i is: " << i << endl;
    }
    return 0;
}
```

In this example, we use a for loop to print the value of the variable i five times. The for loop has three components:

Initialization: In this case, we initialize the variable i to 0.
Condition: The loop will continue executing as long as the condition is true. In this case, the condition is i < 5, which means the loop will execute as long as the value of i is less than 5.
Update: After each iteration of the loop, the update statement i++ is executed, which increments the value of i by 1.

The output of the program will be:

```
The value of i is: 0
The value of i is: 1
The value of i is: 2
The value of i is: 3
The value of i is: 4
```

As you can see, the loop executed five times, and the value of `i` was printed to the console on each iteration.

In summary, a for loop in C++ allows you to execute a block of code a fixed number of times, with an initialization, a condition, and an update statement. It is a powerful tool that can simplify code and make it more efficient by reducing redundancy.

How does while loop work in C++?

A while loop in C++ allows you to execute a block of code repeatedly as long as a certain condition is true. It is typically used when you don't know the number of times the loop should be executed in advance. Here is a simple example that demonstrates how a while loop works in C++:

```
#include <iostream>
using namespace std;
int main() {
    int i = 0;
    while (i < 5) {
        cout << "The value of i is: " << i << endl;
        i++;
    }
    return 0;
}
```

In this example, we use a while loop to print the value of the variable i five times. The while loop has only one component:

Condition: The loop will continue executing as long as the condition is true. In this case, the condition is i < 5, which means the loop will execute as long as the value of i is less than 5.

Inside the while loop, we print the value of i to the console using the cout statement, and then increment the value of i by 1 using the i++ statement. The output of the program will be:

```
The value of i is: 0
The value of i is: 1
The value of i is: 2
The value of i is: 3
The value of i is: 4
```

As you can see, the loop executed five times, and the value of i was printed to the console on each iteration. In summary, a while loop in C++ allows you to execute a block of code repeatedly as long as a certain condition is true. It is a useful tool when you don't know the exact number of times the loop should be executed in advance.

How does do-while loop work in C++?

A do-while loop in C++ allows you to execute a block of code repeatedly as long as a certain condition is true, but unlike the while loop, it always executes the code block at least once. Here is a simple example that demonstrates how a do-while loop works in C++:

```
#include <iostream>
using namespace std;
int main() {
    int i = 0;
    do {
        cout << "The value of i is: " << i << endl;
        i++;
    } while (i < 5);
    return 0;
}
```

In this example, we use a do-while loop to print the value of the variable i five times. The do-while loop has two components:

Code block: Inside the do-while loop, we print the value of i to the console using the cout statement, and then increment the value of i by 1 using the i++ statement.
Condition: The loop will continue executing as long as the condition is true. In this case, the condition is i < 5, which means the loop will execute as long as the value of i is less than 5.

The output of the program will be:

```
The value of i is: 0
The value of i is: 1
The value of i is: 2
The value of i is: 3
The value of i is: 4
```

As you can see, the loop executed five times, and the value of i was printed to the console on each iteration.

The key difference between a do-while loop and a while loop is that the code block inside a do-while loop is executed at least once, regardless of whether the condition is true or false. This makes do-while loops useful in situations where you want to ensure that a certain code block is executed at least once, even if the condition is false from the beginning.

In summary, a do-while loop in C++ allows you to execute a block of code repeatedly as long as a certain condition is true, but always executes the code block at least once. It is a useful tool when you want to ensure that a certain code block is executed at least once.

When to use for, while or do while loop?
In C++, for, while, and do-while loops are all used to execute a block of code repeatedly, but they have different syntax and are used in different situations. Here are some general guidelines for when to use each type of loop:

for loop: Use a for loop when you know the exact number of times the loop should be executed in advance. The syntax of a for loop allows you to initialize a counter variable, specify the condition under which the loop should continue executing, and update the counter variable at the end of each iteration.
while loop: Use a while loop when you don't know the exact number of times the loop should be executed in advance, but you know the condition under which the loop should stop executing. The syntax of a while loop allows you to specify the condition under which the loop should continue executing, and the loop will continue executing as long as the condition is true.
do-while loop: Use a do-while loop when you want to ensure that a certain code block is executed at least once, regardless of whether the condition is true or false from the beginning. The syntax of a do-while loop is similar to a while loop, except that the condition is checked at the end of each iteration instead of at the beginning.

In general, you can use any type of loop to accomplish the same task, but some types of loops may be more appropriate than others depending on the specific requirements of your program. It is always a good practice to choose the loop that makes your code easy to understand, efficient, and error-free.

Nested Loops

What does nesting mean in C++?
In programming, nesting generally refers to the concept of placing one code block or construct inside another code block or construct. In C++, nesting most commonly refers to the concept of using one loop inside another loop or using one conditional statement inside another conditional statement.

For example, nested loops are commonly used to traverse at two (multiple) levels. In this case, one loop is used to iterate through (for example) the rows and another loop is used to iterate through the columns. Similarly, nested conditional (if-else) statements are used to evaluate complex logical expressions.

The main advantage of nesting constructs in programming is that it allows you to perform complex operations by breaking them down into smaller, more manageable parts. However, nesting can also make code more difficult to read and understand if used excessively. Therefore, it is important to use nesting judiciously and to ensure that the code remains clear and easy to understand.

How does nesting of loops work in C++?
In C++, a nested loop is a loop inside another loop. It is a loop construct where one loop is inside another loop. The inner loop is executed fully in each iteration of the outer loop. The basic syntax of a nested loop is as follows:

```
for (initialization; condition; increment) {
   // outer loop code
   for (initialization; condition; increment) {
      // inner loop code
   }
}
```

Here's an example of a nested loop in C++ that prints a multiplication table:

```
#include <iostream>
using namespace std;
int main() {
   int n = 10;

   for (int row = 0; row < 10; row++) { //outer loop
      for (int col = 0; col < 10; col++) { //inner loop
            cout << "#";
      }
   cout << "" << endl; //adds new line
   }

   return 0;
}
```

In this example, the outer loop runs from 1 to 10, and the inner loop also runs from 1 to 10. The inner loop prints the '#' symbol on every iteration. The outer loop controls the number of rows, and the inner loop controls the number of columns. As a result, this program will print 10 rows of '#' symbols each containing 10 columns.

A few more examples!

1. ***Print a square of asterisks:***

```
#include <iostream>
using namespace std;
int main() {
    int side = 5;

    for (int i = 1; i <= side; i++) {
        for (int j = 1; j <= side; j++) {
            cout << "*";
        }
        cout << endl;
    }

    return 0;
}
```

This program uses two nested loops to print a square of asterisks with a specified side length. The outer loop controls the number of rows, and the inner loop controls the number of columns.

2. ***Print a triangle of numbers:***

```
#include <iostream>
using namespace std;
int main() {
    int rows = 5;

    for (int i = 1; i <= rows; i++) {
        for (int j = 1; j <= i; j++) {
            cout << j << " ";
        }
        cout << stdendl;
    }

    return 0;
}
```

This program uses two nested loops to print a triangle of numbers, where each row contains the numbers from 1 up to the row number. The outer loop controls the number of rows, and the inner loop controls the number of numbers printed in each row. Here is the sample output:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```