Part 5: C++ Repetition Statements

Dr. Andi Toce
Lecture Notes for MAC 101 (Introduction to Computer Science)

Last updated / viewed: March 26, 15

Table of Contents

1. C++ Loops. The *while* and *do…while* Statement.

Loops are a very important and powerful component of any programming language. It allows the repetition of one or more statements as long as a condition is satisfied. The most basic loop structure in C++ is the *while* statement.

```
while (condition)
        statement;
```

```
while (condition){
        statement1;
        statement2;
        statement3;
}
```

PrintNumbersWhileLoop.cpp

```
#include <iostream>
using namespace std;

// print the first 10 positive integers
int main ()
{
    int number = 1;

    while (number <= 10){
        cout << number << endl;
        number++;
    }

    return 0;
}
```

Output

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

Infinite Loops

Try now: Remove the statement number++; form the previous program and run it. What do you notice? Why is it happening?

Try now: Modify the previous program so that it prints the first n positive integers in reverse order from largest to smallest.

Example:

Input: Enter a positive integer: 6 Output: 6 5 4 3 2 1

Try now: Modify the previous program so that it prints the first n positive even integers in reverse order from largest to smallest.

Example:

Input: Enter a positive integer: 6 Output: 6 4 2

The do…while statement. Is similar to the while statement. The list of statements is executed before the condition is checked. So the do…while statement is executed at least once.

```
do {
    statement1;
    statement2;
    statement3;
}  while(condition)
```

Examples comparing while and do…while statements

PrintNumbersWhileLoop.cpp

```
#include <iostream>
using namespace std;

// print the first 10 positive integers
int main ()
{
    int number = 1;

    while (number <= 10){
        cout << number << endl;
        number++;
    }

    return 0;
}
```

PrintNumbersDoWhileLoop.cpp

```
#include <iostream>
using namespace std;

// print the first 10 positive integers
int main ()
{
    int number = 1;

    do {
        cout << number << endl;
        number++;
    } while (number <= 10);

    return 0;
}
```

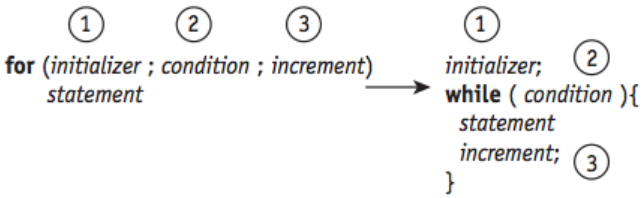
2. The *for* Statements

Lets start by saying that everything that can be done using the for loop can also be done using the while loop. However the **for** statement is more concise and more commonly used.

The for loop statement

```
for(initializer; condition; increment){
    statement1;
    statement2;
}
```

Comparison for and while loops



Example comparing the for loop and while loop

PrintNumbersWhileLoop.cpp

```
#include <iostream>
```

PrintNumbersDoWhileLoop.cpp

```
#include <iostream>
```

4 [PART 5: C++ REPETITION STATEMENTS]

```
using namespace std;

// print the first 10 positive integers
int main ()
{
    int number = 1;

    while (number <= 10){
        cout << number << endl;
        number++;
    }

    return 0;
}
```

```
using namespace std;

// print the first 10 positive integers
int main ()
{
    for(int number=1; number <=10; number++) {
        cout << number << endl;
    }

    return 0;
}
```

Another example: Find the first n factorials.

Factorial.cpp

```
#include <iostream>
using namespace std;

//A program that displays the first n factorials

int main() {

        int factorial;
        int n;

        cout <<"Enter an integer:";
        cin >> n;

    cout << "N \t Factorial of N \n";

    for ( int i = 1; i <= n; ++i ) { //repeat n times
        factorial = 1;
         for ( int j = 1; j <= i; ++j ) //calculate each factorial
                factorial *= j;
         cout << i << '\t' << factorial << '\n';
    }

    return 0;
}
```

Output

```
Enter an integer:11
N        Factorial of N
1        1
2        2
3        6
4        24
5        120
6        720
7        5040
8        40320
9        362880
10       3628800
11       39916800
```

Question: Can we optimize the code in Factorial.cpp? Do we need a nested for loop?

Try now: Write a program that prints the shape below using only the following print statements: cout << "*"; and cout << "\n";

```
*
**
***
****
*****
******
*******
********
```

Try now: Write a program that prints the following diamond shape. You may use output statements that print either a single asterisk (*****) or a single blank. Maximize your use of repetition (with nested **for** structures) and minimize the number of output statements.

```
    *
   ***
  *****
 *******
*********
 *******
  *****
   ***
    *
```

3. The *break* and *continue* Statement

The break statement when used within any of the loop statements it causes the immediate exit from the loop statement.

Example break statement.

ExampleBreakStatement.cpp

```
#include <iostream>
using namespace std;

int main ()
{
    for(int number=1; number <=10; number++) {

        if(number == 5) break;

        cout << number << endl;
    }

    return 0;
}
```

Output

```
1
2
3
4
```

The continue statement when used within any of the loop statements it causes the remaining statements in the body of the loop to be skipped. The loop then continues with the next iteration.

Example continue statement.

ExampleContinueStatement.cpp

```
#include <iostream>
using namespace std;

int main ()
{
    for(int number=1; number <=10; number++) {

        if(number == 5) continue;

        cout << number << endl;
    }

    return 0;
}
```

Output

```
1
2
3
4
6
7
8
9
10
```