Computer Programming [Lab]

Lab # 02: Control Structures [Repetition]

Agenda

- Recall Previous Concepts of control structures selection
- Conditional Operator (Ternary Operator)
- Pseudo code, Algorithm and code Indentation etc.
- Repetition (Loops) and Types
- While Loop
- Do-while Loop
- For Loop

Conditional Operator (Ternary Operator)

More concise and compact way to use if-else statements is use of conditional operator. The conditional operator, written as **? :** , is a ternary operator, which means that it takes three arguments. The syntax for using the conditional operator is:

expression1 ? expression2 : expression3

This type of statement is called a **conditional expression**. The conditional expression is evaluated as follows: If **expression1** evaluates to a nonzero integer (that is, to true), the result of the conditional expression is **expression2**. Otherwise, the result of the conditional expression is **expression3**.

Consider the following statements:

```
if (a >= b)
    max = a;
else
    max = b;
```

You can use the conditional operator to simplify the writing of this if. . .else statement as follows:

```
max = (a >= b) ? a : b;
```

Pseudo code

An outline of a program, written in a form that can easily be converted into real

programming statements. Pseudo code cannot be compiled nor executed. It is simply one step - an important one - in producing the final code. There are no real formatting or syntax rules.

Pseudo code is a "text-based" detail of the problem.

Examples:

```
If student's grade is greater than or equal to 60
    Print "passed"
else
    Print "failed"
------------------------
If (Name1 is entered and Name2 is entered) then return 1
If (Name1 is not entered)
  Send Error Message
```

Advantages

- Easily modified
- Provide additional level at which inspection can be performed.
- It saves lot of your time
- Converting a pseudo code to programming language is very easy, reliable and a professional way of programming.

Code indentation and its importance

The purpose of code indentation and style is to make the program more readable and understandable. It saves lots of time while we revisit the code and use it. A style guide provides a road map which the developer should follow and implement in coding.
Easier to read

- Easier to understand
- Easier to modify
- Easier to maintain
- Easier to enhance

Indentation:

The indentation comes first in formatting section. Use the following things for indentation Spaces, Tabs, comments etc.

*Tips for professional coding

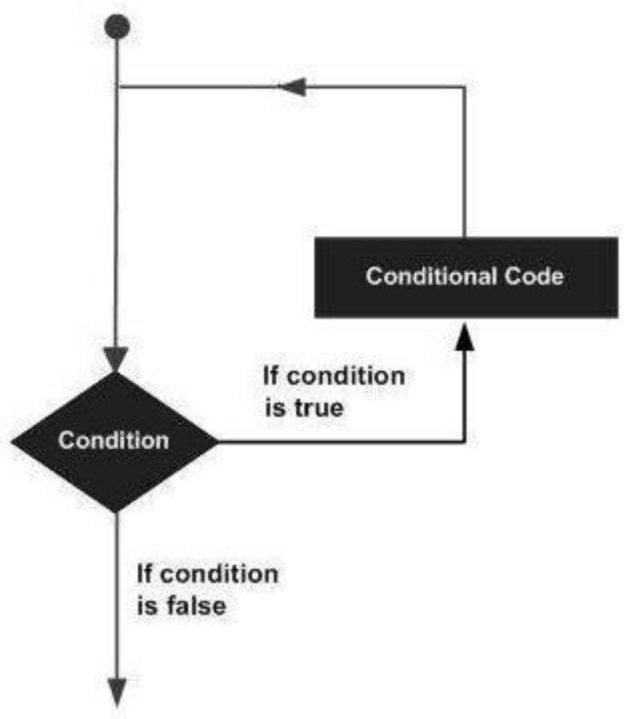
- Must write your Name, program creation Date and Problem Question in multiple line comments at the start of every program.
- The matching braces should be in the same column. It means the start and close brace should have the same column location.
- All binary operators should maintain a space on either side of the operator.
- All commas and semicolons must be followed by single whitespace.
- All keywords like if, while, for, switch, and catch should be followed by a single space.
- Maximum line length should not exceed 120 characters. If it needs to be increased then put it in a different line

Read more from here: http://mrbool.com/importance-of-code-indentation/

Control Structures (Repetition)

Provide alternatives to sequential program execution and are used to alter the sequential flow of There may be a situation, when you need to execute a block of code several number of times. In general statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. Programming languages provide various control structures that allow for more complicated execution paths.

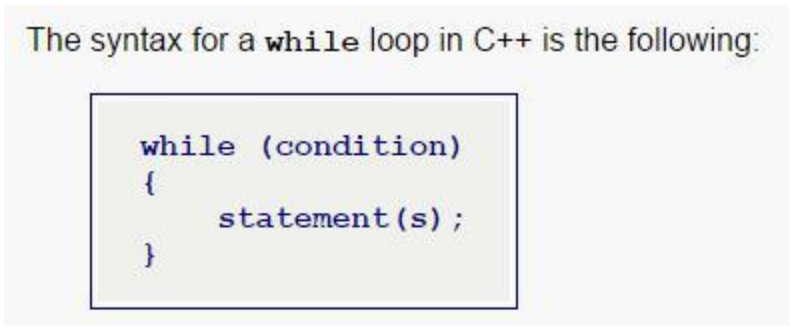
A loop statement allows us to execute a statement or group of statements multiple times and following is the general from of a loop statement in most of the programming languages:



While Loop

A while loop statement repeatedly executes a target statement as long as a given condition is true.

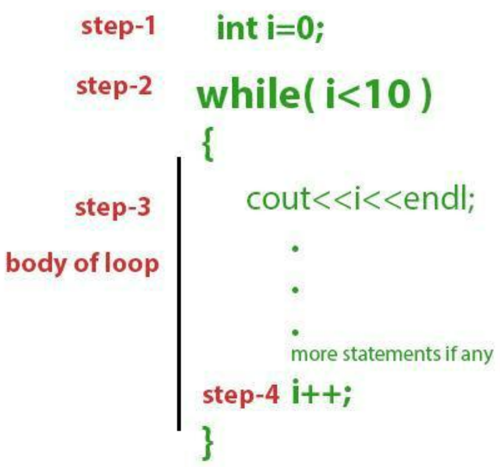
Syntax



The syntax for a **while** loop in C++ is the following:

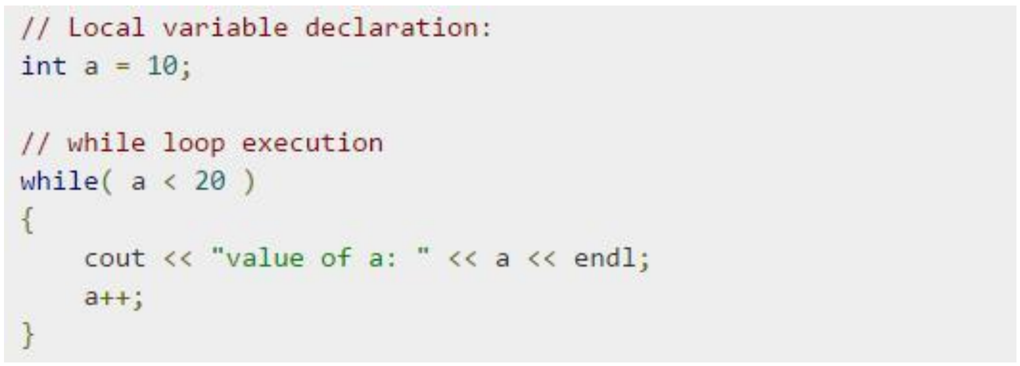
```
while (condition)
{
      statement(s);
}
```

Here, statement(s) may be a single statement or a block of statements. The condition may be any expression, and true is any non-zero value. The loop iterates while the condition is true. When the condition becomes false, program control passes to the line immediately following the loop.



```
int i=0;
while( i<10 )
{
    cout<<i<<endl;
        .
        .
        .
    i++;
}
```

Example



```
// Local variable declaration:
int a = 10;

// while loop execution
while( a < 20 )
{
    cout << "value of a: " << a << endl;
    a++;
}
```

For Loop

A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

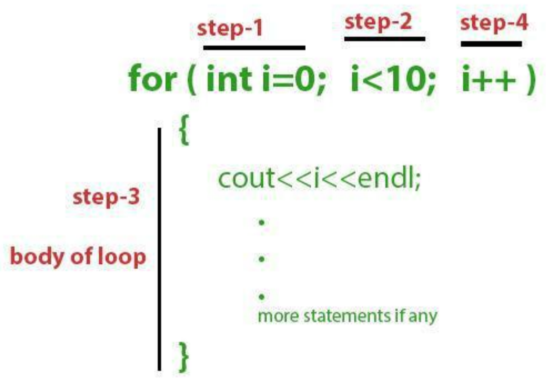
Syntax:

```
for (initialization; condition; increment step)
{
     statement(s);
}
```

Here is the flow of control in a **For loop**:

- The **initialization** step is executed first, and only once. This step allows you to declare and initialize any loop control variables.
- Next, the **condition** is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement just after the **For loop**.
- After the body of **for loop** executes, the flow of control jumps back up to

  the **increment** statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.
- The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, **for loop** terminates.



Example

```
for (int i = 0; i < 5; i++)
{
    cout << "Executing the body of the loop" << endl;
}
```

Do While Loop

Unlike **for** and **while** loops, which test the loop condition at the top of the loop, the **do...while** loop checks its condition at the bottom of the loop.

A **do...while** loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

Syntax:

The syntax of a do...while loop in C++ is:

```
do
{
    statement(s);
}
while (condition);
```

Notice that the condition appears at the end of the loop, so the statement(s) in the loop execute once before the condition is tested.

Example

```
// Local variable declaration:
int a = 10;

// do loop execution
do
{
    cout << "value of a: " << a << endl;
    a = a + 1;
}while( a < 20 );
```

Nested Loops

A loop can be nested inside of another loop. C++ allows at least 256 levels of nesting.

Syntax:

The syntax for a nested for loop statement in C++ is as follows:

```
for ( init; condition; increment )
{
   for ( init; condition; increment )
   {
      statement(s);
   }
   statement(s); // you can put more statements.
}
```

The syntax for a nested **while loop** statement in C++ is as follows:

```
while(condition)
{
   while(condition)
   {
      statement(s);
   }
   statement(s); // you can put more statements.
}
```

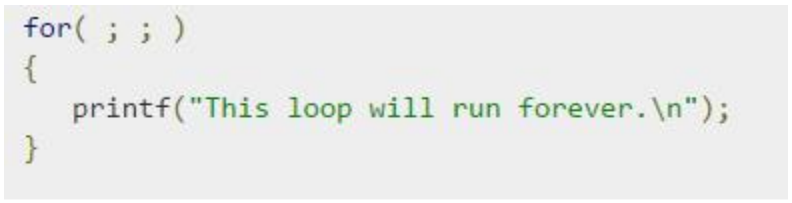
The syntax for a nested **do...while loop** statement in C++ is as follows:

```
do
{
   statement(s); // you can put more statements.
   do
   {
      statement(s);
   }while( condition );

}while( condition );
```

The Infinite Loop:

A loop becomes infinite loop if a condition never becomes false. The for loop is traditionally used for this purpose. Since none of the three expressions that form the for loop are required, you can make an endless loop by leaving the conditional expression empty.



```
for( ; ; )
{
   printf("This loop will run forever.\n");
}
```

When the conditional expression is absent, it is assumed to be true. You may have an initialization and increment expression, but C++ programmers more commonly use the for(;;) construct to signify an infinite loop.

NOTE: You can terminate an infinite loop by pressing Ctrl + C keys

Loop Control Statements:

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

C++ supports the following control statements. Click the following links to check their detail.

Break statement

- When the break statement is encountered inside a loop, the loop is immediately terminated and program control resumes at the next statement following the loop.
- It can be used to terminate a case in the switch statement (covered in the next chapter).

If you are using nested loops (i.e., one loop inside another loop), the break statement will stop the execution of the innermost loop and start executing the next line of code after the block.

Syntax

```
break;
```

Continue statement

The continue statement works somewhat like the break statement. Instead of forcing termination, however, continue forces the next iteration of the loop to take place, skipping any code in between.

For the for loop, continue causes the conditional test and increment portions of the loop to execute. For the while and do...while loops, program control passes to the conditional tests.

Syntax:

```
continue;
```

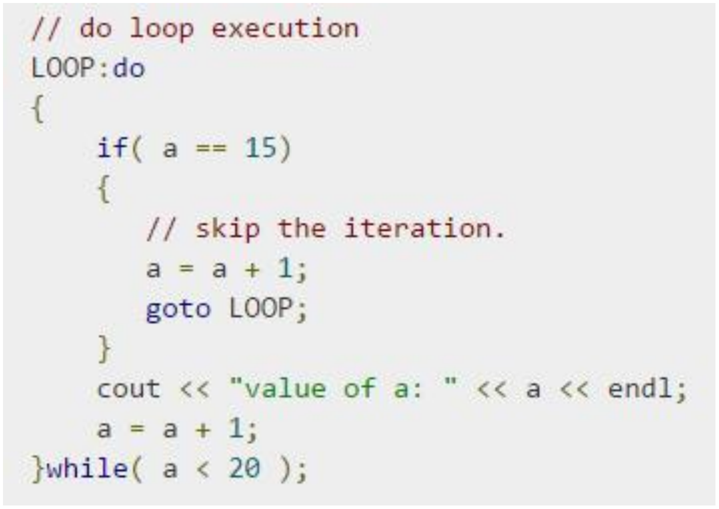
Goto statement

A goto statement provides an unconditional jump from the goto to a labeled statement in the same function.

NOTE: Use of goto statement is highly discouraged because it makes difficult to trace the control flow of a program, making the program hard to understand and hard to modify. Any program that uses a goto can be rewritten so that it doesn't need the goto.

Where **label** is an identifier that identifies a labeled statement. A labeled statement is any statement that is preceded by an identifier followed by a colon (:).

Syntax:



```
// do loop execution
LOOP:do
{
    if( a == 15)
    {
       // skip the iteration.
       a = a + 1;
       goto LOOP;
    }
    cout << "value of a: " << a << endl;
    a = a + 1;
}while( a < 20 );
```