C++ Conditional Statements & Iteration Statements

Control Structure

The work of control structures is to give flow and logic to a program. There are three types of basic control structures in C++.

Sequence Structure

Sequence structure refers to the sequence in which a program executes instructions one after another.

Selection Structure

Selection structure refers to the execution of instructions according to the selected condition, which can be either true or false. There are two ways to implement selection structures: by using if-else statements or switch case statements.

Loop Structure

Loop structure refers to the execution of an instruction in a loop until the condition becomes false.

C++ If Else

If else statements are used to implement a selection structure. Like any other programming language, C++ also uses the if keyword to implement the decision control instruction.

The condition for the if statement is always enclosed within a pair of parentheses. If the condition is true, then the set of statements following the if statement will execute. And if the condition evaluates to false, then the statement will not execute; instead, the program skips that enclosed part of the code.

An expression in if statements is defined using relational operators. The statement written in an if block will execute when the expression following if evaluates to true. But when the if block is followed by an else block, then when the condition written in the if block turns to be false, the set of statements in the else block will execute.

Following is the syntax of if-else statements:

```
if (condition) {
   statements;
```

```
} else {
   statements;
}
```

One example where we could use the if-else statement is:

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

int main()
{
   int age;
   cout << "Enter a number: ";
   cin >> age;
   if (age >= 50)
   {
      cout << "Input number is greater than 50!" << endl;
   }
   else if (age == 50)
   {
      cout << "Input number is equal to 50!" << endl;
   }
   else
   {
      cout << "Input number is less than 50!" << endl;
   }
}
```

Input

```
Enter a number: 51
```

Output

Input number is greater than 50!

Note: The else if statement checks for a different condition if the conditions checked above it evaluate to false.

C++ Switch Case

The control statement that allows us to make a decision effectively from the number of choices is called a switch, or a switch case-default since these three keywords go together to make up the control statement.

Switch executes that block of code, which matches the case value. If the value does not match with any of the cases, then the default block is executed.

Following is the syntax of switch case-default statements:

```
switch ( integer/character expression )

{

case {value 1} :

   do this;


case {value 2} :

   do this;


default :

   do this;

}
```

The expression following the switch can be an integer expression or a character expression. Remember, that case labels should be unique for each of the cases. If it is the same, it may create a problem while executing a program. At the end of the case labels, we always use a colon (:). Each case is associated with a block. A block contains multiple statements that are grouped together for a particular case.

The break keyword in a case block indicates the end of a particular case. If we do not put the break in each case, then even though the specific case is executed, the switch will continue to execute all the cases until the end is reached. The default case is optional. Whenever the expression's value is not matched with any of the cases inside the switch, then the default case will be executed.

One example where we could use the switch case statement is:

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

int main()
{
  int i = 2;
  switch (i)
  {
  case 1:
    cout << "Statement 1" << endl;
    break;

  case 2:
    cout << "Statement 2" << endl;
    break;

  default:
    cout << "Default statement!" << endl;
  }
}
```

Output:

Statement 2

The test expression of a switch statement must necessarily be of an integer or character type and the value of the case should be an integer or character as well. Cases should only be inside the switch statement and using the break keyword in the switch statement is not necessary.

C++ Iteration Statements

C++ Loops

The need to perform an action, again and again, with little or no variations in the details each time they are executed is met by a mechanism known as a loop. This involves repeating some code in the program, either a specified number of times or until a particular condition is satisfied. Loop-controlled instructions are used to perform this repetitive operation efficiently, ensuring the program doesn't look redundant at the same time due to the repetitions.

Following are the three types of loops in C++ programming:

- For Loop
- While Loop
- Do While Loop

For Loop

A for loop is a repetition control structure that allows us to efficiently write a loop that will execute a specific number of times. The for-loop statement is very specialized. We use a for loop when we already know the number of iterations of that particular piece of code we wish to execute. Although, when we do not know about the number of iterations, we use a while loop which is discussed next.

Here is the syntax of a for loop in C++ programming.

for (initialise counter; test counter; increment / decrement counter){ // set of statements}

Here,

- **Initialize counter:** It will initialize the loop counter value. It is usually i=0.
- **Test counter:** This is the test condition, which if found true, the loop continues, otherwise terminates.
- **Increment/decrement counter:** Incrementing or decrementing the counter.
- **Set of statements:** This is the body or the executable part of the for loop or the set of statements that has to repeat itself.

One such example to demonstrate how a for loop works is:

#include <iostream>

using namespace std;

```
int main()
{
   int num = 10;
   int i;
   for (i = 0; i < num; i++)
   {
      cout << i << " ";
   }
   return 0;
}
```

Output:

0 1 2 3 4 5 6 7 8 9

First, the initialization expression will initialize loop variables. The expression i=0 executes once when the loop starts. Then the condition i < num is checked. If the condition is true, then the statements inside the body of the loop are executed. After the statements inside the body are executed, the control of the program is transferred to the increment of the variable i by 1. The expression i++ modifies the loop variables. Iteratively, the condition i < num is evaluated again.

The for loop terminates when i finally becomes greater than num, therefore, making the condition i<num false.

While Loop

A While loop is also called a pre-tested loop. A while loop allows a piece of code in a program to be executed multiple times, depending upon a given test condition which evaluates to either true or false. The while loop is mostly used in cases where the number of iterations is not known. If the number of iterations is known, then we could also use a for loop as mentioned previously.

Following is the syntax for using a while loop.

```
while (condition test){    // Set of statements}
```

The body of a while loop can contain a single statement or a block of statements. The test condition may be any expression that should evaluate as either true or false. The

loop iterates while the test condition evaluates to true. When the condition becomes false, it terminates.

One such example to demonstrate how a while loop works is:

```
#include <iostream>

using namespace std;


int main()

{

  int i = 5;

  while (i < 10)

  {

    cout << i << " ";

    i++;

  }


  return 0;

}
```

Output:

5 6 7 8 9

Do While Loop

A do-while loop is a little different from a normal while loop. A do-while loop, unlike what happens in a while loop, executes the statements inside the body of the loop before checking the test condition.

So even if a condition is false in the first place, the do-while loop would have already run once. A do-while loop is very much similar to a while loop, except for the fact that it is guaranteed to execute the body at least once.

Unlike for and while loops, which test the loop condition first, then execute the code written inside the body of the loop, the do-while loop checks its condition at the end of the loop.

Following is the syntax for using a do-while loop.

do{ statements;} while (test condition);

First, the body of the do-while loop is executed once. Only then, the test condition is evaluated. If the test condition returns true, the set of instructions inside the body of the loop is executed again, and the test condition is evaluated. The same process goes on until the test condition becomes false. If the test condition returns false, then the loop terminates.

One such example to demonstrate how a do-while loop works is:

```
#include <iostream>

using namespace std;


int main()

{

  int i = 5;

  do

  {

    cout << i << " ";

    i++;

  } while (i < 5);


  return 0;

}
```

Output:

5

Here, even if i was less than 5 from the beginning, the do-while let the print statement execute once and then terminate.