Chapter 7: Control Statements

HSSLIVE.IN

Selection Statements

The statements provided by C++ for the selected execution of some statements are called **decision making statements** or **selection statements**. `if` and `switch` are the two types of selection statements in C++. The selection will be based on conditions. The conditions are formed by relational or logical expressions.

`if` statement

The `if` statement is used to select a set of statements for execution based on a condition. In C++, conditions (otherwise known as test expressions) are provided by relational or logical expressions. The syntax (general form) of `if` statement is as follows:

```
if (test expression)
{
     statement block;
}
```

Here the `test expression` represents a condition which is either a relational expression or logical expression. If the test expression evaluates to True (non-zero value), a statement or a block of statements associated with `if` is executed. Otherwise, the control moves to the statement following the `if` construct.

`if – else` statement

The syntax is:

```
if (test expression)
{
     statement block 1;
}
else
{
     statement block 2;
}
```

If the `test expression` evaluates to True, only the `statement block 1` is executed. If the `test expression` evaluates to False `statement block 2` is executed.

When we write an `if` statement inside another `if` block, it is called **nesting**.

The `else if` ladder

When we want to select one action from more than two actions, different conditions will be given and each condition will decide which statement is to be selected for execution. For this, we can use **`else if`** ladder, also referred to as the **`else if`** staircase because of its appearance. It is also known as `if...else if` statement. The general form of else if ladder is:

```
if (test expression 1)
   statement block 1;
      else if (test expression 2)
          statement block 2;
              else if (test expression 3)
                  statement block 3;
                     ..............
```


Joy John's CS Capsule

```
else
    statement block n;
```

At first, the `test expression 1` is evaluated and if it is True, the `statement block 1` is executed and the control comes out of the ladder. That means, the rest of the ladder is bypassed. If `test expression 1` evaluates to False, then the `test expression 2` is evaluated and so on. If any one of the test expressions evaluates to True, the corresponding statement block is executed and control comes out of the ladder. If all the test expressions are evaluated to False, the `statement block n` after the final `else` is executed.

HSSLIVE.IN

`switch` statement

We have seen the concept of multiple branching with the help of `else if` ladder. Some of these programs can be written using another construct of C++ known as **`switch`** statement. This selection statement successively tests the value of a variable or an expression against a list of integers or character constants. The syntax of **`switch`** statement is as follows:

```
switch(expression)
{
    case constant_1     : statement block 1;
                          break;
    case constant_2     : statement block 2;
                          break;
    case constant_3     : statement block 3;
                          break;
                        :
                        :
    case constant_n-1   : statement block n-1;
                          break;
    default             : statement block n;
}
```

In the syntax `switch`, `case`, `break` and `default` are keywords. The expression is evaluated to get an integer or character constant and it is matched against the constants specified in the case statements. When a match is found, the statement block associated with that case is executed until the `break` statement or the end of `switch` statement is reached. If no match is found, the statements in the `default` block get executed. The default statement is optional and if it is missing, no action takes place when all matches fail.

The following are the requirements to implement a multi branching using switch statement:

- Conditions involve only equality checking. In other cases, it should be converted into equality expression.
- The first operand in all the equality expressions should be the same variable or expression.
- The second operand in these expressions should be integer or character constants.

switch statement	else if ladder
• Permits multiple branching	• Permits multiple branching
• Evaluates conditions with equality operator only	• Evaluates any relational or logical expression
• Case constant must be an integer or a character type value	• Condition may include range of values and floating point constants
• When no match is found, default statement is executed	• When no expression evaluates to True, else block is executed
• `break` statement is required to exit from `switch` statement	• Program control automatically goes out after the completion of a block
• More efficient when the same variable or expression is compared against a set of values for equality	• More flexible and versatile compared to `switch`

The conditional operator (?:)

HSSLIVE.IN

C++ has a ternary operator. It is the **conditional operator** (**?**:) consisting of the symbols **?** and **:** (a question mark and a colon). It requires three operands. It can be used as an alternative to if...else statement. Its general form is:

```
Test expression ? True_case code : False_case code;
```

`Test expression` can be any relational or logical expression and `True_case code` and `False_case code` can be constants, variables, expressions or statement. The operation performed by this operator is shown below with the help of an `if` statement.

```
if (Test expression)
{
     True_case code;
}
else
{
     False_case code;
}
```

Looping (Iteration) Statements

Four elements of a loop:

1. **Initialisation:** The loop control variable (Variable used in the condition) gets its first value. The initialisation statement is executed only once, at the beginning of the loop.
2. **Test expression:** It is a relational or logical expression whose value is either True or False. It decides whether the loop-body will be executed or not. If the test expression evaluates to True, the loop-body gets executed, otherwise it will not be executed.
3. **Update statement:** The update statement modifies the loop control variable by changing its value. The update statement is executed before the next iteration.
4. **Body of the loop:** The statements that need to be executed repeatedly constitute the body of the loop.

`while` statement

It is an entry-controlled loop. The condition is checked first and if it is True the body of the loop will be executed. That is the body will be executed as long as the condition is True. The syntax of **`while`** loop is:

```
initialisation of loop control variable;
while(test expression)
{
    body of the loop;
    updation of loop control variable;
 }
```

`for` statement

It is also an entry-controlled loop in C++. All the three loop elements (initialisation, test expression and update statement) are placed together in **`for`** statement. The syntax is:

```
for (initialisation; test expression; update statement)
{
    body-of-the-loop;
}
```

HSSLIVE.IN

do...while statement

In the case of `for` loop and `while` loop, the test expression is evaluated before executing the body of the loop. If the test expression evaluates to False for the first time itself, the body is never executed. Its syntax is :

```
initialisation of loop control variable;
do
{
    body of the loop;
    updation of loop control variable;
 } while(test expression);
```

Placing a loop inside the body of another loop is called **nesting** of a loop. When we nest two loops, the outer loop counts the number of completed repetitions of the inner loop. Here the loop control variables for the two loops should be different.

break statement	continue statement
• Used with `switch` and loops. • Brings the program control outside the `switch` or loop by skipping the rest of the statements within the block. • Program control goes out of the loop even though the test expression is True.	• Used only with loops. • Brings the program control to the beginning of the loop by skipping the rest of the statements within the block. • Program control goes out of the loop only when the test expression becomes False.