Conditional Statements

C if...else statements

Sometimes, we wish to execute one set of instructions if a particular condition is met, and another set of instructions if it is not. This kind of situation is dealt with in C language using a decision control system.

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. Comparing two values using relational operators allows us to determine whether they are equal, unequal, greater than, or less than.

If we want to execute a particular code in some situation and its vice versa/opposite/different code if that situation doesn't occur, then if..else statements can be used. It all depends on the condition. If the condition returns a true value, the situation has occurred, and the true part of the code will be executed. If the condition returns a false value, the false part of the code will be executed.

Conditions	**Meaning**
a==b	a is equal to b
a!=b	a is not equal to b
a<b	a is less than b
a>b	a is greater than b
a<=b	a is less than or equal to b
a>=b	a is greater than or equal to b

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 <stdio.h>

int main() {
   int num = 10;
   if (num <= 10) {
      printf("Number is less than equal to 10.");
   } else {
      printf("Number is greater than 10.");
   }
   return 0;
}
```

Output

Number is less than equal to 10.

Ladder if-else

If we want to check multiple conditions, then ladder if-else can be used. If the previous condition returns false, then only the next condition will be checked.

Syntax:

```
if (/*condition*/) {
```

```
    // statements
} else if (/*condition*/) {
    // statements
} else if (/*condition*/) {
    // statements
}
```

Nested if-else

We can also write an entire if-else statement within either the body of another if statement or the body of an else statement. This is called the 'nesting' of ifs.

```
if (/*condition*/) {
    if (/*condition*/) {
        // statements
    } else {
        // statements
    }
} else {
    // statements
}
```

Switch Case Statements

What is Switch?

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. The expression in switch returns an integral value, which is then compared with different cases. 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 expression )
{
case {value 1} :
   do this ;

case {value 2} :
   do this ;

case {value 3} :
   do this ;

default :
   do this ;
}
```

Understanding the syntax:

The expression following the switch can be an integer expression or a character expression. The case value 1, and 2 are case labels that are used to identify each case individually. 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.

Whenever the switch is executed, the value of test-expression is compared with all the cases present in switch statements. When the case is found, the block of statements associated with that particular case will execute. The break keyword 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, C's 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 <stdio.h>


int main()
{
  int i = 2;


  switch (i)
  {
  case 1:
    printf("Statement 1");
    break;


  case 2:
    printf("Statement 2");
    break;


  case 3:
    printf("Statement 3");
    break;


  default:
    printf("No valid i to switch to.");
```

```
    break;
  }
  return 0;
}
```

Output

Statement 2

Different to if-else. How?

There is one problem with the if statement: the program's complexity increases whenever the number of if statements increases. If we use multiple if-else statements in the program, the code might become difficult to read and comprehend. Sometimes it also even confuses the developer who himself wrote the program. Using the switch statement is the solution to this problem.

Furthermore,

- Switch statements cannot evaluate float conditions, and the test expression can only be an integer or a character, whereas if statements can evaluate float conditions as well.
- Switch statements cannot evaluate relational operators hence they are not allowed in switch statements, whereas if statements can evaluate relational operators.
- Cases in the switch can never have variable expressions; for example, we cannot write case a + 3:.

Rules for Switch statements

- The test expression of Switch must necessarily be an int or char.
- The value of the case should be an integer or character.
- Cases should only be inside the switch statement.
- Using the break keyword in the switch statement is not necessary.
- The case label values inside the switch should be unique.

It is not necessary to use the break keyword after every case. Break keywords should only be used when we want to terminate our case at that time, otherwise we won't.

We can also use nested switch statements i.e., switch inside another switch. Also, the case constants of the inner and outer switch may have common values without any conflicts.