Decision Structures

In our daily life, we often encounter many options and we have to select one. Our selections decide the way of our life. In every separation point we make our decision and go on our way. While programming, we usually need to decide the path of the program flow according to the parameters and conditions. Actually the ability of making decision is one of the key points of intelligent programming. Thanks to control structures (decision & repetition) we are able to evaluate the condition and select the flow path. We have 4 types of decision structures in C++:

- If
- If/Else
- Conditional Operator (?)
- Switch

The If/else statement chooses between two alternatives. This statement can be used without the else, as a simple if statement. Another decision statement, switch, creates branches for multiple alternative sections of code, depending on the value of a single variable. Finally, the conditional operator is used in specialized situations.

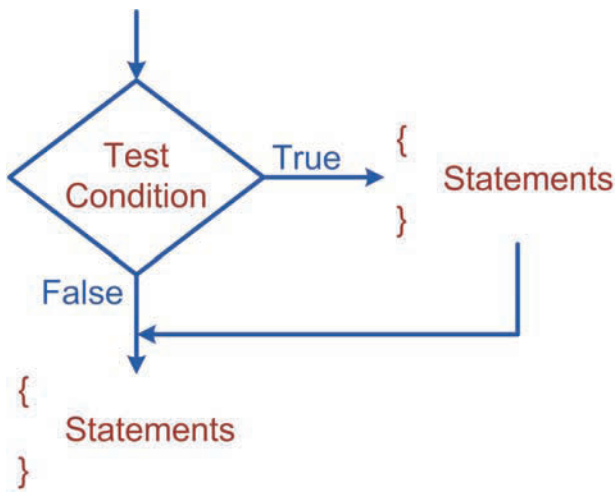
Relational and Equality Operators tests some kind of relation between two entities. These include numerical equality and inequalities. These operators usually return true or false, depending on whether the conditional relationship between the two operands holds or not. An expression created using a relational operator forms what is known as a **condition**. Decision structures evaluate the conditions.

C++ Relational and Equality Operators are:

>	:	Greater
>=	:	Greater or Equal
<	:	Smaller
<=	:	Smaller or Equal
==	:	Equal
!=	:	Not Equal

The if structure

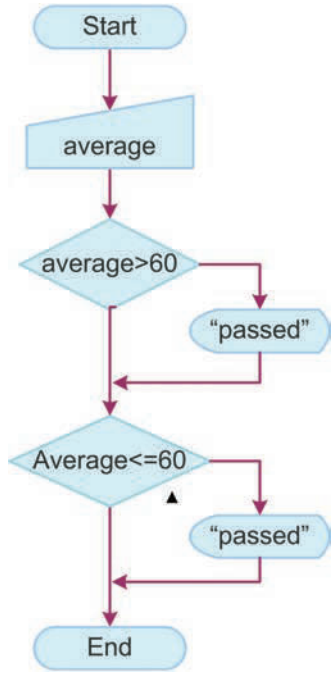
The "if structure" is used to execute statement(s) only if the given condition is satisfied. The Computer evaluates the condition first, if it is true, statement is executed if not the statement is skipped and the program continues right after this conditional structure. This can be illustrated as:



The following code prints "passed" or "failed" depending on the average of the student by using two separate if structures.

```
if (average>60)
{
  cout<<"passed";        //if the  average is greater than 60
                         //print passed.
}

if (average<=60)
{
  cout<<"failed";        //if the  average is less than or equal
                         //to 60 print failed.
}
```



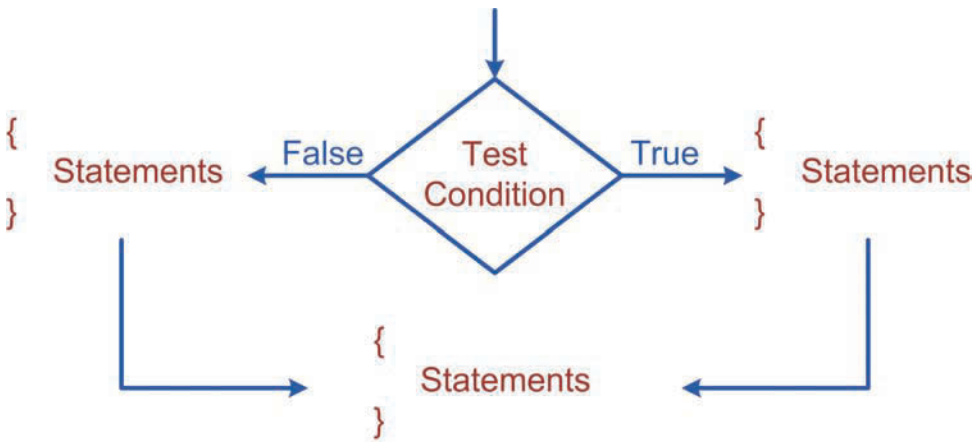
Using "if" in a Flowchart

The first **if** structure checks whether the average is greater than 60, if so writes "passed" to the output. The second **if** structure checks whether the average is less than or equal to 60 if so writes "failed" to the output.

If we have a single statement in a "if" then it can be written without braces; but if we have a compound statement to be executed we need to enclose them between braces ({ }).

The if/else structure

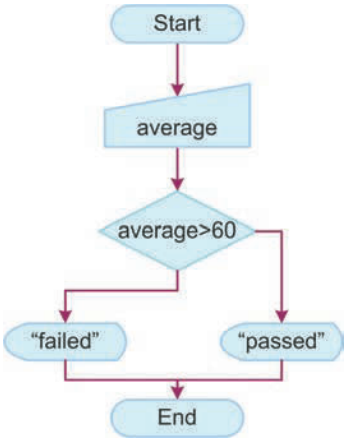
We can additionally specify what we want to do if the condition does not hold with **else**. While the **if** statement lets a program decide whether a statement or block is executed, the **if/else** statement lets a program decide which of two statements or blocks is executed. This can be illustrated as :



If any student passed, namely the average is bigger than 60, it is unnecessary to check if he failed. If he passed it is obvious that he didn't fail, and in the same manner if he didn't pass it is obvious that he failed.

Truth in C++

C++ has a very simple logic to cope with truth. Any value other than zero is regarded as true. Thus 1 and -2 are both true just like 0.0001.Since they are all different from zero. In the same logic 0, 0.0 or +0.0 and -0 are all accepted as false.



Using "if/else" in a Flowchart

```
if (average>60)
  cout<<"passed";          //if the  average is greater than 60
                           //prints passed.
else
  cout<<"failed";          //if condition does not hold then
                           //prints failed.
```

Exercise: Odd or Even

Write a program that decides if a number is odd or even.

Input : An integer.

Output : "Odd" or "Even".

Hint : Even numbers are divisible by 2. That is to say when an even number is divided by 2, the remainder is 0. Use modulus operator (%) to obtain the remainder.

Let's improve our average example. What if we have an average value less than 0? As a programmer we should always keep in mind the unexpected cases. We would be able to respond to the user that any value less than 0 is not valid.

```
if (average>60)
{
  cout<<"Passed";
}
else if (average<0)
{
  cout<<"Wrong Input";
}
else
{
  cout<<"Failed";
}
```

The **if/else** structure above checks the first condition, if it is satisfied, prints "Passed" and skips the rest of the structure. If the first condition is not satisfied, the second condition is checked, and so on. If none of the conditions is held, the last statement is executed.

Exercise: Sign

Write a program that decides the sign of a number.

Input: An integer.

Output: Positive, Negative or Zero.

Logical Operators

Logical operators simplify nested **if** and **if/else** structures. They combine multiple logical expressions and return a single result (True or False). There are three logical operators in C++:

- ! (Not)
- && (And)
- || (Or)

! (Logical Not) operator has only one operand (unary operator) and returns the opposite of it. **Not Operator** gives true if the operand is false, and gives false if the operand is true. For example:

```
!(5 > 7)          //evaluates to true.
!true             //evaluates to false.
```

X	!X
0	1
1	0

The Truth Table of the NOT Operator (!)

&& (Logical And) operator has two operands (binary operator). It returns true only if both operands are true, and returns false otherwise. So we may need this operator when we have to perform a task if two conditions are fulfilled at the same time. For example we want to determine if a given integer (num) is divisible by 3 and 7.

```
if ((num % 3 == 0) && (num % 7 == 0))
  cout<<"It is divisible by 3 and 7";
```

X	Y	X && Y
0	0	0
0	1	0
1	0	0
1	1	1

The Truth Table of the AND Operator (&&)

|| (Logical Or) operator has two operands. It returns false if both operands are false, and returns true otherwise. It can be used in a case if at least one of two conditions has to be true to perform a task. For example we want to check if a number is divisible by 3 or 7.

```
if ((num % 3 == 0) || (num % 7 == 0))
  cout<<"It is divisible by 3 or 7";
```

X	Y	X && Y
0	0	0
0	1	1
1	0	1
1	1	1

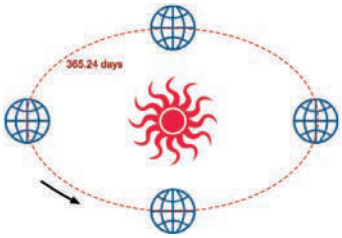
The Truth Table of the OR Operator (||)

Exercise: Letter

Write a program that checks if an entered character is between 'a' and 'z' or 'A' and 'Z'. Namely check if it is a letter or not.

Input: One character.

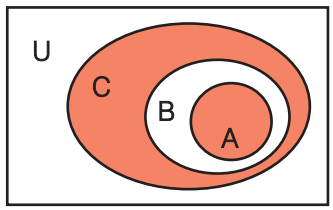
Output: "IT IS A LETTER" or "IT IS NOT A LETTER".



Leap years are needed so that the calendar is in alignment with the earth's motion around the sun.

Let's have a look how to use logical operators to determine whether a given year is a **leap year**. Leap years are years with an extra day (February 29); this happens almost every four years. Generally, leap years are divisible by four, but century years are special, they must also be divisible by 400. Given a year decide whether it is a leap year or not.

We know that the set of numbers divisible by 400 is a subset of the set of numbers divisible by 100. And the set of numbers divisible by 100 is a subset of numbers divisible by 4.



U = All of the years

C= Years divisible by 4

B= Years divisible by 100

A= Years divisible by 400

From the description it is clear that if the number is in the red area than it is a leap year otherwise it is not. We can describe the red area as (A or (C and (Not B)))

```
if ((year%400==0)||((year%4==0)&&(!(year%100==0))))
   cout << "Leap Year";
else
   cout << "Not a Leap Year";
```

Exercise: Leap Year

Implement the leap year problem without logical operators (by nested **if/else**)

Input: An integer for year value.

Output: "IT IS A LEAP YEAR" or "IT IS NOT A LEAP YEAR".

The Conditional Operator (?)

C++ provides another selective operator that can be an alternative of simple **if/else**. If we are choosing from two options based on a condition we can simply implement it with a conditional operator. Let's implement our "pass" "fail" example with "?".

With if/else	With ?

`if (average>60)`
`  cout<<"passed";`
`else`
`  cout<<"failed";`
	`cout<<((average>60) ? "passed" : "failed");`

Here condition (average > 60) is evaluated. If it is true, "passed" is written; If not, "failed' is printed.

Our leap year problem can be written as:

```
february=((year%400==0)||((year%100!=0)&&(year%4==0))) ? 29 : 28;
```

If the condition is true it is a leap year, if false it is not.

The Switch Structure

Switch allows us to select from multiple choices based on constant values. If we are to use several "if" and "else if" instructions to check constant values it is best to implement it by using switch. For example we are getting the month number from the user and printing the month name on the screen

```
/*
PROG: c2_01switch.cpp
Read the order of a month and prints its name.
*/
int main()
{
     int month;
     cout<<"enter the month number";
     cin>>month;
     switch (month) {
            case 1 :
                  cout<<"it is january";
                  break;
            case 2 :
                  cout<<"it is february";
                  break;
            case 3 :
                  cout<<"it is march";
                  break;
            case 4 :
                  cout<<"it is april";
                  break;
            case 5 :
                  cout<<"it is may";
                  break;
            case 6 :
                  cout<<"it is june";
                  break;
            case 7 :
                  cout<<"it is july";
                  break;
            case 8 :
                  cout<<"it is august";
                  break;
            case 9 :
```

```
                cout<<"it is september";
                break;
          case 10 :
                cout<<"it is october";
                break;
          case 11 :
                cout<<"it is november";
                break;
          case 12 :
                cout<<"it is december";
                break;
          default :
                cout<<"wrong input";
    }
```

Switch evaluates the value of expression and performs all of the instructions starting from the true case of the expression's value till the ending brace.

If none of the cases has the value of expression then instructions under default part are executed. You can have only one default statement in a switch statement block.

The break statement at the end of the case statement tells C++ to exit the switch statement. C++ does not generate an error message if you omit a break statement. However, if you omit it, C++ executes all the statements in the following case statement, even if that case is false. In nearly all circumstances, this is not what you want to do.

Sometimes we need to group the cases since we have the same operation for some values. This can be understood better by an example. Again we are getting the month number from the user but this time printing how many days this month has to the screen.

```
/*
PROG: c2_02months.cpp
Read the order of a month and print how many days it has.
*/
  switch (month)
  {
       case 2 :     cout<<"it has 28 days"; break;
       case 4 : case 6 : case 9 : case 11:
                    cout<<"it has 30 days"; break;
       default :    cout<<"it is 31 days";  break;
  }
```

Exercise: Traffic Lights

Implement the program which reads a character from the user as the indication of color of the traffic light, and send a comment to the user.

Input : A character indicating color of light ('r','y','g').

Output : "WAIT", "GET READY", "GO" or "WRONG INPUT".

SUMMARY

We need **decision structures** in order to give direction to the program flow.

We execute or skip a statement block by using simple **if** statement. If the condition holds then execute, if not skip the statement block.

We choose one of two options by using an **if/else** structure. If the condition holds then execute statement block, if not execute statement block 2.

C++ provides us with three kinds of **logical operators** in order to combine simple logical expressions and construct compound complex. The C++ logical operators:

Not Operator (!) is used to get opposite of one expression.

And Operator (&&) is used to get intersection of two expressions.

Or Operator (||) is used to get union area of two expressions.

Conditional Operator (? :) can be used as an alternative of simple if /else.

Several **if**, **if/ else** structures can be expressed by a **switch** structure. If we have the case that an expression may have constant values, it is best to check them by a **switch** instead of messy **if/else** components.