MODULE 1: INTRODUCTION TO PROGRAMMING IN C#

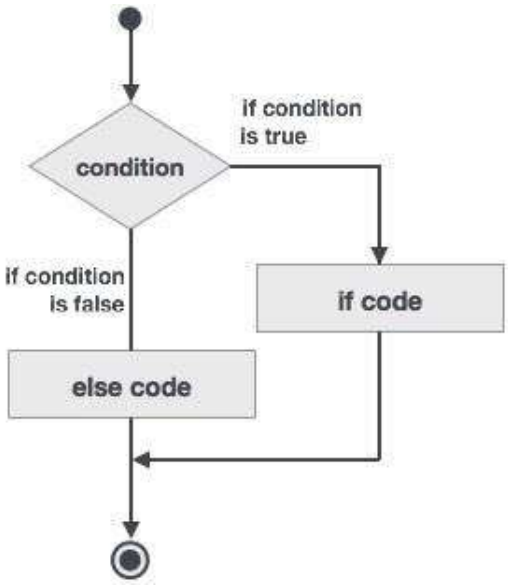
TUTORIAL 2: CONDITIONAL STATEMENTS, ITERATION AND LISTS

Introduction

Almost every program needs to make a test somewhere to check a particular condition. For example, a currency converter program will need to find what you want to convert to and from. It is therefore necessary to be able to ask questions within a program and act accordingly. Typically, this will take the form:

```
if(some condition is true) then
      do something
else
      do something else
```

We would represent this in flowchart form as:



A typical section of program code might be as follows:

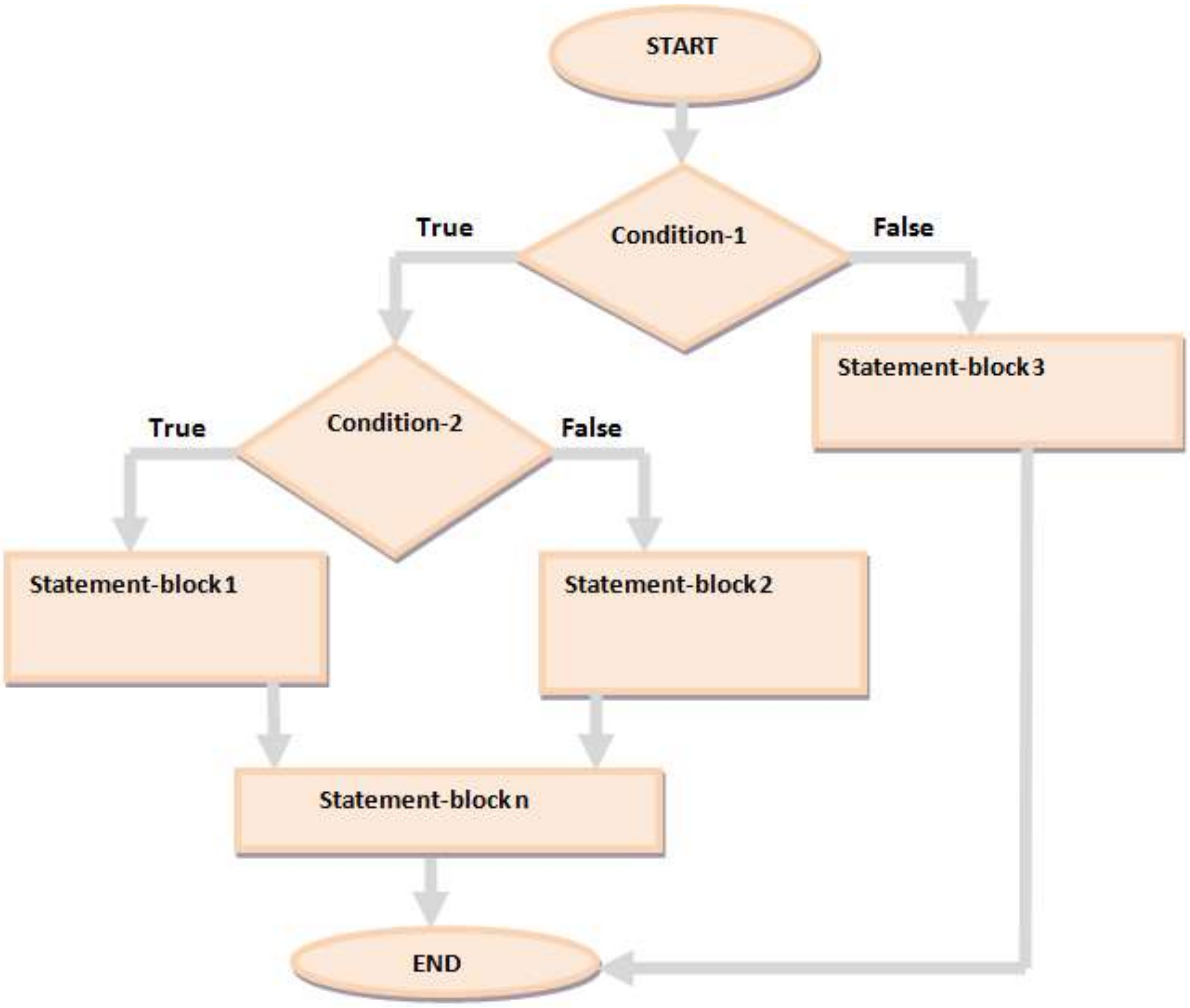
```
Console.Write("Please enter your first number (x): ");
int x = Convert.ToInt32(Console.ReadLine());
Console.Write("Please enter your second number (y): ");
int y = Convert.ToInt32(Console.ReadLine());
if (x > y)
{
    Console.WriteLine("x is greater than y");
}
else
{
    Console.WriteLine("y is greater than or equal to x");
}
```

Note how the curly braces { } are used to define a section of code that is to be executed. It may contain several program statements as illustrated:

```
if(some condition is true)
{
      do thing 1
      do thing 2
}
else
{
      do thing 3
      do thing 4
}
```

Not all situations can be dealt with by a simple either/or strategy and we now explore the various possibilities.

Nested If Statements

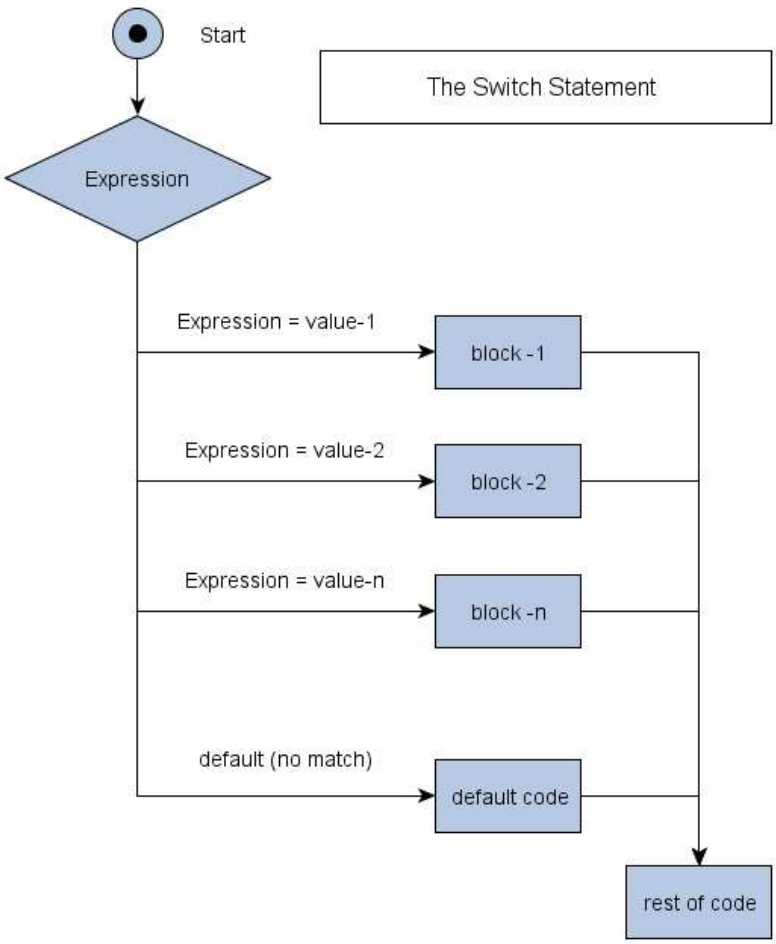


An example of typical code is:

```
static void Main(string[] args)
{
    Console.Write("What is your name? ");
    String yourName = Console.ReadLine();
    if(yourName=="Jack")
    {
        Console.WriteLine("Hello Jack");
    }
    else
        if (yourName == "Jill")
        {
            Console.WriteLine("Hello Jill");
        }
    else
    {
        Console.WriteLine("I don't know you {0}",yourName);
    }
}
```

Some nested ifs can get a bit complicated and it is helpful to use suitable indentation.

Switch Statements



Sometimes a switch statement offers a better solution to the nested if e.g.

```
static void Main(string[] args)
{
    Console.Write("Enter a value between 1 and 3 ");
    int myVal = Convert.ToInt32(Console.ReadLine());
    switch (myVal)
    {
        case 1:
            Console.WriteLine("ONE");
            break;
        case 2:
            Console.WriteLine("TWO");
            break;
        case 3:
            Console.WriteLine("THREE");
            break;
        default:
            Console.WriteLine("Invalid Value");
            break;
    }
}
```

Additional Notes (Conditional Statements)

When checking things, the computer returns a *true* or *false* result. This may seem obvious and simple, but you actually need to put a bit of work into understanding the underlying principles which are more complicated than you might think!

When we are talking, we frequently make use of such words as 'and', 'or' and 'not' but these have particular meanings in programming that might lead to confusion when we are trying to make decisions based upon a number of facts. We need to understand how a computer understands and processes these basic commands as well as some more complicated conditions. This topic is known as 'computer logic' and follows the formal rules set out in 'boolean algebra', named after George Boole who first introduced the concepts in *The Mathematical Analysis of Logic* (1847). We therefore refer to related programming aspects such as boolean variables (that can take the values 1 or 0, corresponding to *true* or *false* respectively).

It is beyond the scope of this tutorial to explore this topic in depth, but students should eventually acquaint themselves with more details including De Morgan's Laws.

For essential computing, the programmer should be able to use the operators AND, OR and NOT which are represented in C# computer code as:

```
If ((x<5) && (y>12)){
      do something
}
```

BOTH conditions must be true in order to return a 'true' result. The symbol '&&' is used to represent the operation 'AND'. Similarly, we use the symbol '|' to represent the OR operation where either (or both) conditions being true will give a result of true. Also, we use the symbol '!' to perform the NOT operation to

change a logic state from true to false or false to true. There is also an 'exclusive OR' operation (x ^ y). It is similar to the OR operation but it does not return a 'true' result when *both* operands are true.

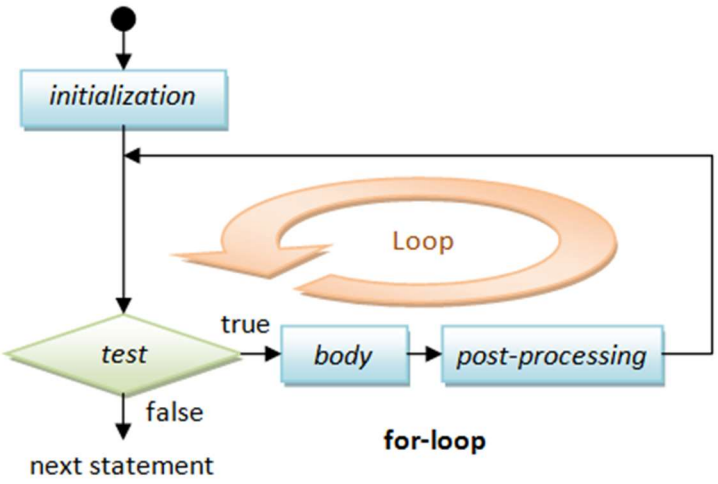
Note that we have looked at nested if statements. It is also possible to have nested switch statements but these will not be necessary for our immediate requirements.

Iteration

'Iteration' refers to a section of program code that repeats itself a number of time. Such a structure is normally called a '**program loop**'. There are different types of loops which can be used for different programming situations. A typical loop is the *for* loop where a given set of program instructions is repeated until some specified condition is true. A typical example is given below:

```
for (int i=0; i<3; i++)
{
    Console.WriteLine(i);
}
```

In the above, we use the integer variable i to control the number of repetitions of the code within the curly braces { and }. The format is **for** (*starting value, condition, increment*).



The for loop is generally used when a fixed number of repetitions is required. It has associated jump statements *break* and *continue* to allow us to test a condition within the loop and take some action other than the normal program flow within the loop. Other types of loops are available when we want to repeat a section of code while a particular condition is true. We also look at how program loops can used with lists of items and how we can search for a stored data value.

The break statement

This is used with for loops when we have found some condition to be true and we to exit the loop and transfer control to the statement immediately following the loop. It is also used in the switch statement to provide an exit when a particular condition is met.

The continue statement

This forces the loop to skip the remaining code in the loop so that it starts its next iteration if the loop condition permits.

The while loop

A condition is tested at the beginning of the loop and iteration continues until that condition is false e.g.

```
int i = 0;
while(i>=0)
{
    Console.Write("Enter a value (-1 to quit) ");
    i=Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("Your value was {0}", i);
}
```

Note that we had to initialize the value of i to some non-negative value because the test is at the start.

The do … while loop

A condition is tested at the end of the loop and iteration continues until that condition is false e.g.

```
int i;
do
{
    Console.Write("Enter a value (-1 to quit) ");
    i = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("Your value was {0}", i);
} while (i >= 0);
```

This time, we did not have to initialize i because its value is set within the loop and not tested until the end.

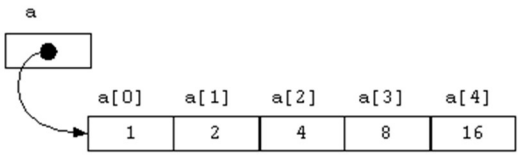
Additional Notes (Program Loops)

There is also a *foreach* statement associated with arrays and this will be covered in the next tutorial.

While we used a simple search technique to look for a specific item in an array, it should be noted that C# provides some built-in facilities (methods) that can be used with arrays to greatly simplify coding, you don't need these at the moment.

Simple Arrays as Lists

An array is simply a collection of similar type data in a numbered list with a given name. It allows us to store related items of data in a single variable and refer to individual items by their position in the list (the *element number*). We usually need program loops to access successive elements of the array.



Here is a simple section of program code that declares and initialises an array and outputs its contents:

```
String [] names = new String[] {"Fred", "Sue", "Bill", "Jane" };
for (int i = 0; i < 4; i++)
{
    Console.WriteLine(names[i]);
}
```

Note that array elements are numbered from zero upwards.

Searching a List

We would typically use a for loop to search a list (stored as an array) for a given value or piece of text. Once we have found the item, we can use the break statement to exit the loop. Examine the following example carefully:

```
String [] names = new String[] {"Fred", "Sue", "Bill", "Jane" };
int numNames = 4;
String searchName = "George";
Boolean found = false;
for (int i = 0; i < numNames; i++)
{
    if(names[i]==searchName)
    {
        found = true;
        Console.WriteLine("Name found: item = {0}", i + 1);
        break;
    }
}
if(!found)
{
    Console.WriteLine("Name not found!");
}
```

This demonstrates several techniques introduced in this tutorial. Try changing the searchName to “Bill” and check that the program finds the name in the list. Note that we have also introduced another variable type, Boolean, that can take the values *true* or *false*.

Notes on Good Practice (Conditional Statements)

Conditional statements should cover all possibilities – this typically involves a final ‘else’ statement to capture any situation that is not covered in an explicit list of conditions. It might be used to generate an error message to the user indicating an unusual problem.

Avoid situations in which a user can be trapped in tests from which they can’t exit from some interactive input where the programmer has not foreseen the range of possible responses.

Use program code structures that are most appropriate to the situation. In this tutorial, we have looked at the if-then-else construct as well as the switch option. These can provide similar facilities but the programmer should choose the one most appropriate to any given situation.

Don't ask for too much information on any on-screen form – gently take users through a series of forms in which they can respond (and obtain help if necessary). This is simpler for both the user and the programmer. Where possible make use of drop-down lists and calendars to ensure that incoming data is valid.

Avoid equality tests involving floating point numbers! There is always a limited storage involved which leads to tiny errors in the actual values. If a program is repeating a section of code until some condition is true, there is a danger of never meeting that condition if we are using comparisons with floating point values. A safer alternative is to use inequality tests e.g. less than (<) or greater than (>).

Notes on Good Practice (Program Loops)

Care should be taken when defining the condition associated with the loop control in order to avoid infinite loops as these can cause the program to never end! Because these can occur by a simple slip during program development, it is good policy to always save your program before running it (although Visual Studio will do this by default).

It is important to select the appropriate type of loop. The do-while loop will always run the loop code once because the test comes at the end. The for and while loops apply the test at the start of the loop. The for loop uses its own control variable and its value changes at the start of the loop whereas the while and do-while loops use a variable whose value can change at some point within the loop.

Program Errors

There are 3 types of program errors: **syntax** errors, **runtime** errors, and **logical** errors (or *semantic* errors).

1. Syntax errors include: missing semicolons, mismatching brackets, misspelt function names etc.

2. Runtime errors include: attempting to divide by zero, trying to open a non-existent file.

3. Logical errors occur when the programmer creates code that is technically incorrect e.g. adding instead of subtracting. These errors are the hardest to detect because the program compiles OK and the program may appear to run correctly. However, the program may, on occasions, produce strange results. We use a facility known as a debugger to help us find such errors. This allows us to set breakpoints in the program and view data values.

   Refer to the on-line tutorial regarding debugging as listed below.

On-line Resources for Further Study

Decision Making:

https://www.tutorialspoint.com/csharp/csharp_decision_making.htm

Debugging:

https://msdn.microsoft.com/en-us/library/mt243867.aspx

C# Operators:

https://msdn.microsoft.com/en-gb/library/6a71f45d.aspx

Boolean Algebra:

https://www.tutorialspoint.com/computer_logical_organization/boolean_algebra.htm

Program Loops:

https://www.tutorialspoint.com/csharp/csharp_loops.htm

Arrays (we shall deal with 2-D arrays in tutorial 4):

https://www.tutorialspoint.com/csharp/csharp_arrays.htm