NAME = AHMAD ULLAH KHAN

ID = 6958

SEMESTER = 4^{TH}

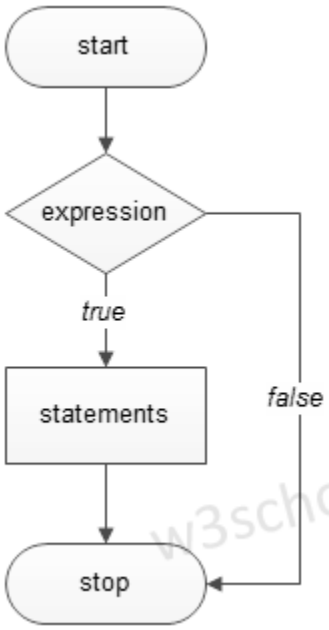
DATE = 23/09/2020

Q1. a. What is decision making in C # explain with the help of flow charts?

Ans:

These types of statements are used by programmers to determine one or more conditions evaluated by the program at run-time. Specific blocks of code associated with these statements will be executed only when the condition is determined.

The flowchart of the Decision-making technique in C can be expressed as:

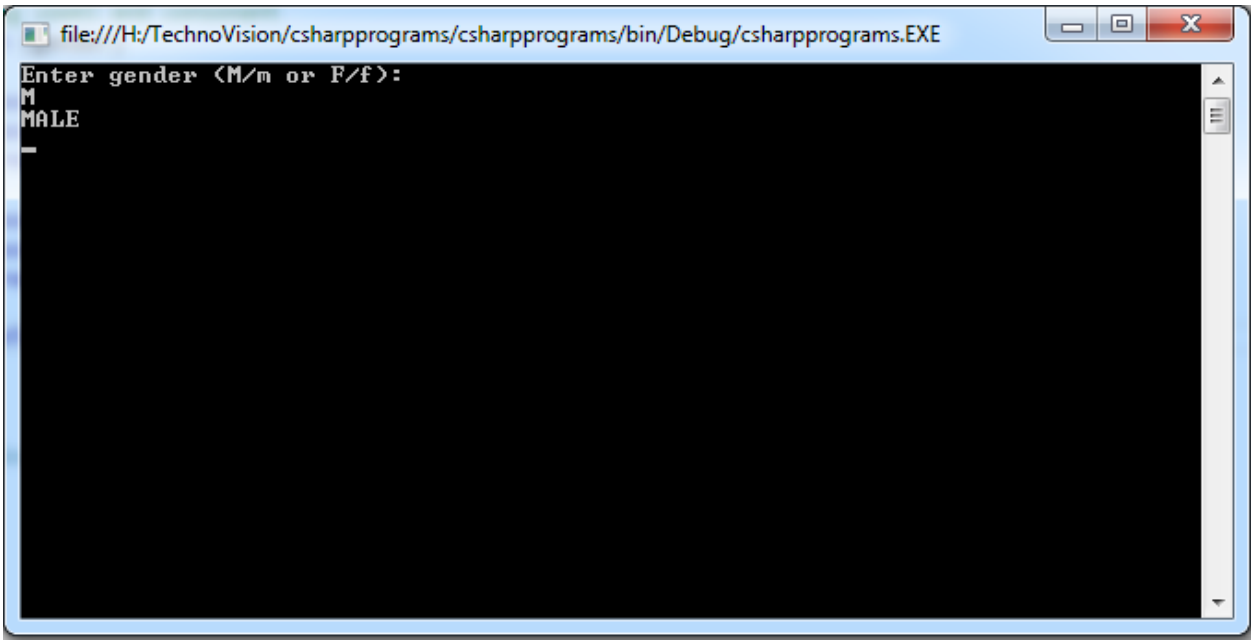


If the boolean expression evaluates to true, then the block of code inside the if statement is executed. If boolean expression evaluates to false, then the first set of code after the end of the if statement(after the closing curly brace) is executed.

. b: Write a program in C # in which different genders are to be separated based on user input?

```
using System;

class Program

{

   static void Main(string[] args)

   {

      char gender;

      Console.WriteLine("Enter gender (M/m or F/f): ");

      gender = Convert.ToChar(Console.ReadLine());

      switch (gender)

      {

         case 'M':

         case 'm': Console.WriteLine("MALE");

         break;

         case 'F':

         case 'f': Console.WriteLine("FEMALE");

         break;

         default: Console.WriteLine("Unspecified Gender");

            break;

      }
```

```
        Console.ReadLine();
    }
}
```

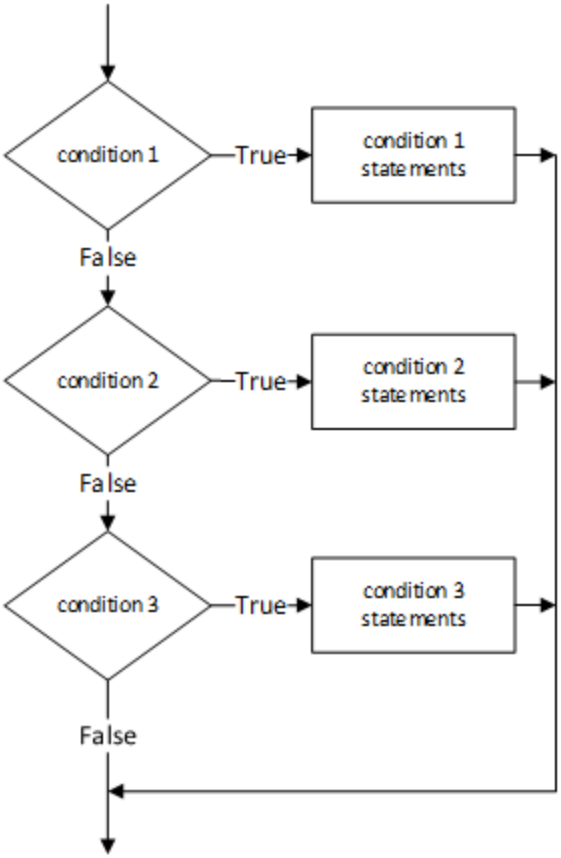


Q2. a. What is the role of "If else if" in decision making explain with the help of a flow chart ?

Ans:

The if-else-if statement executes one condition from multiple statements. The execution starts from top and checked for each if condition. The statement of if block will be executed which evaluates to be true. If none of the if condition evaluates to be true then the last else block is evaluated.

Testing a condition is inevitable in programming. We will often face situations where we need to test conditions (whether it is true or false) to control the flow of program. These conditions may be affected by user's input, time factor, current environment where the program is running, etc.



b. Write a program in C # in which different weather conditions are mentioned?

```
using System;

namespace DecisionMaking {
  class Program {
    static void Main(string[] args) {

      int a = 45;

      if (a == 35) {
        Console.WriteLine("today weather 35C");
```

```
    }
    else if (a == 40) {
      Console.WriteLine("today weather 40C");
    }
    else if (a == 45) {
      Console.WriteLine("today weather 45C");
    } else {
      Console.WriteLine("None of the values is matching");
    }
    Console.WriteLine("Exact value of a is: {0}", a);
    Console.ReadLine();
  }  }}
```

Output

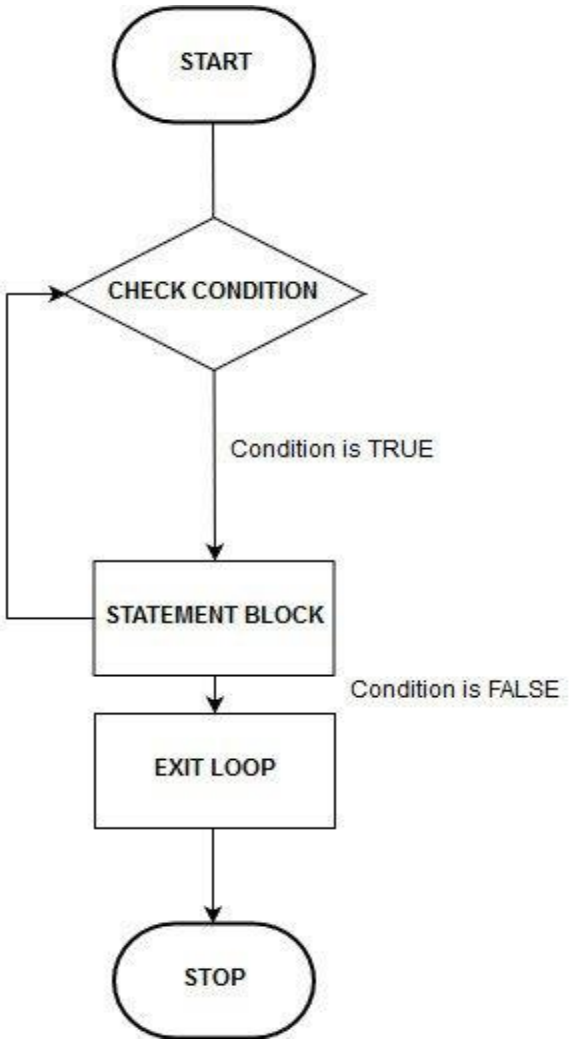
today weather 45C

Exact value of a is: 45

Q3. a. What is the role of Loops in C# explain with the help of a flow chart?

Ans: Looping in a programming language is a way to execute a statement or a set of statements multiple times depending on the result of the condition to be evaluated to execute statements. The result condition should be true to execute statements within loops.This loop allows using three statements, first is the counter initialization, next is the condition to check it and then there is an increment/decrement operation to change the counter variable. You will understand it once we see some programs.

Flow chart



b. How many loops are supported by C #, give separate example for each loop?

Ans:

they come in 4 different variants, are supported by c#

while loop

The while loop is probably the most simple one, so we will start with that. The while loop simply executes a block of code as long as the condition you give it is true. A small example, and then some more explanation:

```
using System;

namespace ConsoleApplication1
{
   class Program
   {
   static void Main(string[] args)
   {
      int number = 0;

      while(number < 5)
      {
      Console.WriteLine(number);
      number = number + 1;
      }

      Console.ReadLine();
   }
   }
}
```

The do loop

The opposite is true for the do loop, which works like the while loop in other aspects through. The do loop evaluates the condition after the loop has executed, which makes sure that the code block is always executed at least once.

```
using System;

public class Program

{

        public static void Main()
```

```
        {
                int i = 0;
                do
                {
                        Console.WriteLine("i = {0}", i);
                        i++;
                        if (i > 5)
                                break;
                } while (i < 10);
        }
}
```

The for loop

The for loop is a bit different. It's preferred when you know how many iterations you want, either because you know the exact amount of iterations, or because you have a variable containing the amount. Here is an example of the for loop.

```
using System;

namespace ConsoleApplication1
{
   class Program
   {
   static void Main(string[] args)
   {
```

```
        int number = 5;

        for(int i = 0; i < number; i++)
        Console.WriteLine(i);

        Console.ReadLine();
    }
    }
}
```

The foreach loop

The last loop we will look at, is the foreach loop. It operates on collections of items, for instance arrays or other built-in list types. In our example we will use one of the simple lists, called an ArrayList

```
using System;
using System.Collections;

namespace ConsoleApplication1
{
    class Program
    {
    static void Main(string[] args)
    {
        ArrayList list = new ArrayList();
        list.Add("John Doe");
        list.Add("Jane Doe");
        list.Add("Someone Else");

        foreach(string name in list)
        Console.WriteLine(name);

        Console.ReadLine();
    }
    }
}
```

Q4. Why do the developers prefer for loops instead other loops justify your answer with the help of an C # coded program ?

Ans:

A for loop is intended to iterating over a range or collection. It's implicitly a traversal of a set, with defined beginning and end points. In ordinary usage it's going to end without explicit user intervention, although it's explicit breakouts or skips can be added with break, continue or similar instructions. From a logic and code-flow standpoint, a for loop is about dealing with repetitive data. "Do this to every item in this list" or "Collect the data from every one of these records".

A other loops, on the other hand, is about program state, not about the data. It's generally used for putting the program into a mode and keeping it their until some condition triggers a mode switch: lots of servers are written inside a single loop.

```
using System;

namespace MyApplication
{
  class Program
  {
    static void Main(string[] args)
    {
      for (int i = 0; i <= 10; i = i + 2)
      {
        Console.WriteLine(i);
      }
    }
  }
}
```

Q5. a. What is encapsulation and its role in object oriented programming ?

In object-oriented computer programming languages, the notion of encapsulation refers to the bundling of data, along with the methods that operate on that data, into a single unit. Many programming languages use encapsulation frequently in the form of classes. A class is a program-code-template that allows developers to create an object that has both variables (data) and behaviors (functions or methods). A class is an example of encapsulation in that it consists of data and methods that have been bundled into a single unit.

Encapsulation may also refer to a mechanism of restricting the direct access to some components of an object, such that users cannot access state values for all of the variables of a particular object. Encapsulation can be used to hide both data members and data functions or methods associated with an instantiated class or object.

Encapsulation is one of the fundamental concepts in object-oriented programming (OOP). It describes the idea of bundling data and methods that work on that data within one unit, e.g., a class in Java.

This concept is also often used to hide the internal representation, or state, of an object from the outside. This is called information hiding. The general idea of this mechanism is simple. If you have an attribute that is not visible from the outside of an object, and bundle it with methods that provide read or write access to it, then you can hide specific information and control access to the internal state of the object.

Access Modifiers

Java supports four access modifiers that you can use to define the visibility of classes, methods, and attributes.

These modifiers are, starting from the most to the least restrictive one:

- private

- no modifier
- protected
- public

Private

This is the most restrictive and most commonly used access modifier. If you use the *private* modifier with an attribute or method, it can only be accessed within the same class. Subclasses or any other classes within the same or a different package can't access this attribute or method.

No modifier

When you don't provide any access modifier for your attribute or method, you can access it within your class and from all classes within the same package. That's why it's often called package-private.

Protected

Attributes and methods with the access modifier *protected* can be accessed within your class, by all classes within the same package, and by all subclasses within the same or other packages.

Public

This is the least restrictive access modifier. Methods and attributes that use the public modifier can be accessed within your current class and by all other classes.