2020

VISUAL PROGRAMMING

FINAL PAPER

Name: Muhammad Abdullah Minhas

I.D : 13864

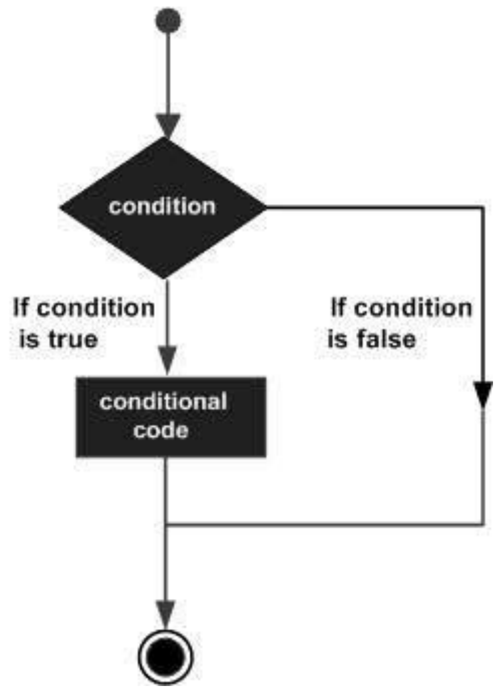
Teacher: Sir Ayub

Date: 23rd Sep 2020

Question no 1

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

Ans) Decision making structures requires the programmer to specify one or more conditions to be evaluated or tested by the program.

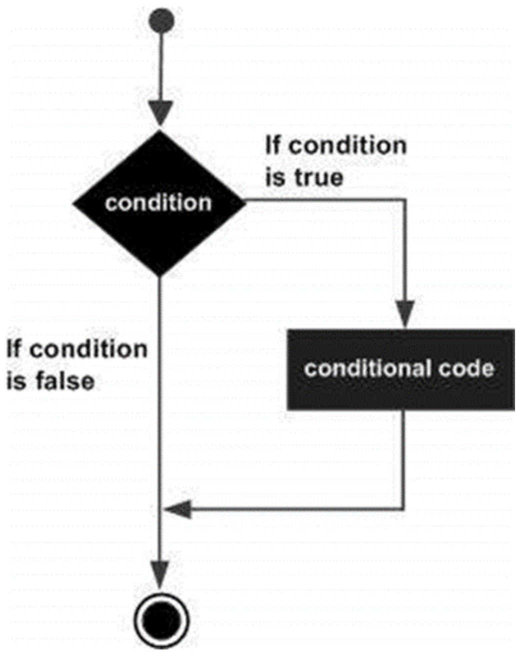


There are three types of decision making statements;

1) **if statement**
2) **if else statement**
3) **if else if statement**

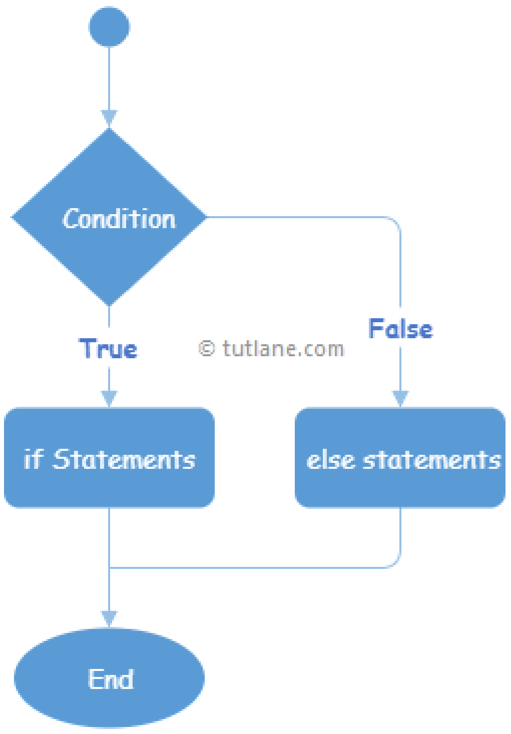
if Statement :

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



if else statement:

If the expression evaluates to true, then the if block of code is executed, otherwise else block of code is executed.



Syntax

```
if(boolean_expression) {
   /* statement(s) will execute if the boolean expression is true */
} else {
   /* statement(s) will execute if the boolean expression is false
*/
}
```

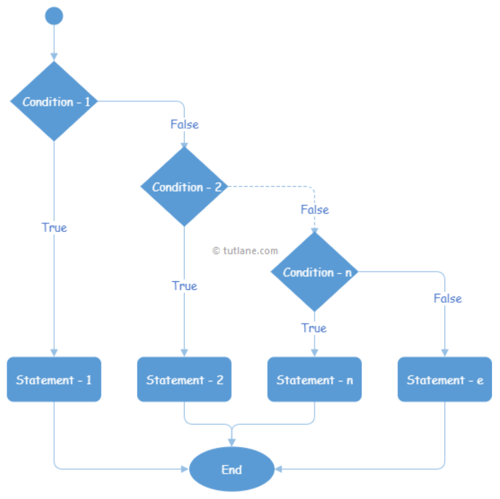
if else if statement:

When using if, else if, else statements there are few points to keep in mind.

•An if can have zero or one else's and it must come after any else if's.

•An if can have zero to many else if's and they must come before the else.

•Once an else if succeeds, none of the remaining else if's or else's will be tested.



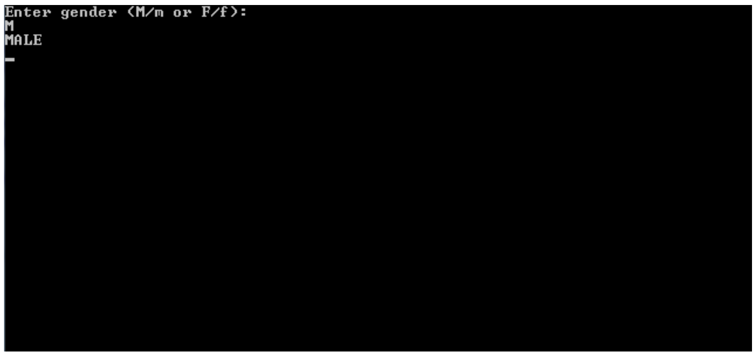
Syntax

```
if(boolean_expression 1) {
   /* Executes when the boolean expression 1 is true */
}
else if( boolean_expression 2) {
   /* Executes when the boolean expression 2 is true */
}
else if( boolean_expression 3) {
   /* Executes when the boolean expression 3 is true */
} else {
   /* executes when the none of the above condition is true */
}
```

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

Ans) **code**

```
using System;

class Program
{

    static void Main(string[] args)
    {

        char gender;

        //Reading gender from user
        Console.WriteLine("Enter gender (M/m or F/f): ");
        gender = Convert.ToChar(Console.ReadLine());


    // checking vowel and consonant
        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();

    }
```

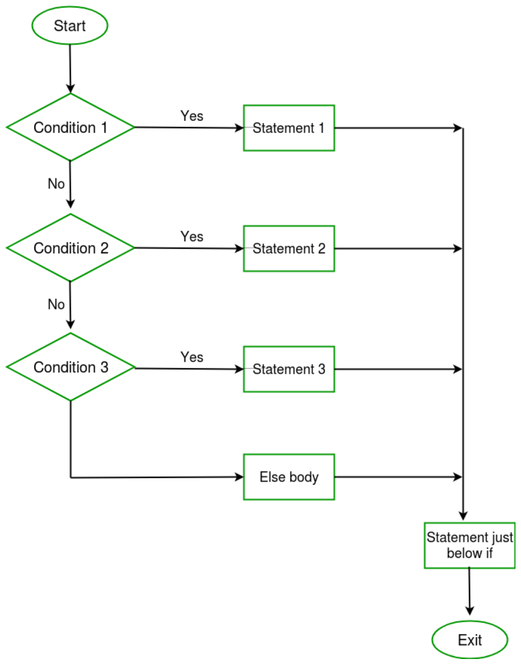


```
Enter gender (M/m or F/f):
M
MALE
_
```

<u>Question no 2</u>

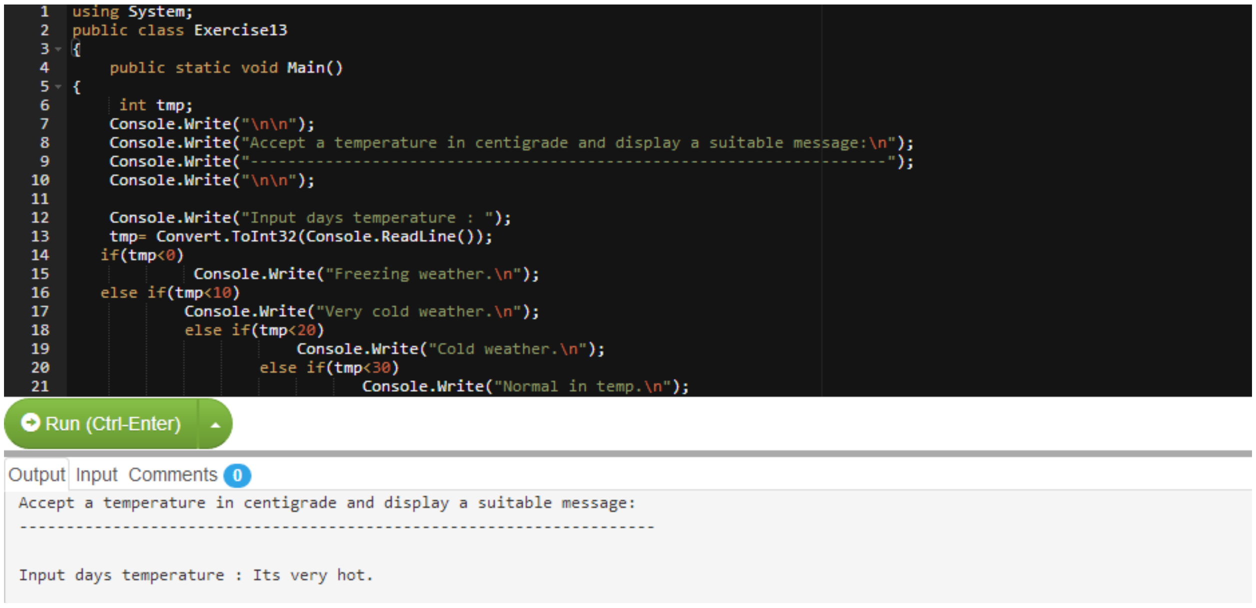
Q2. a. What is the role of “If else if” in decision making explain with the help of flow chart ?

Ans) The C# if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the C# else-if ladder is bypassed. If none of the conditions are true, then the final else statement will be executed.



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

Ans)
```
using System;
public class Exercise13
{
   public static void Main()
{
    int tmp;
   Console.Write("\n\n");
   Console.Write("Accept a temperature in centigrade and display a
suitable message:\n");
   Console.Write("-----------------------------------------------------
-----------");
   Console.Write("\n\n");

   Console.Write("Input temperature : ");
   tmp= Convert.ToInt32(Console.ReadLine());
  if(tmp<0)
          Console.Write("Freezing weather.\n");
  else if(tmp<10)
         Console.Write("Very cold weather.\n");
         else if(tmp<20)
                 Console.Write("Cold weather.\n");
               else if(tmp<30)
                      Console.Write("Normal in temp.\n");
                    else if(tmp<40)
                            Console.Write("Its Hot.\n");
                         else
                              Console.Write("Its very hot.\n");
}
}
```



```
using System;
public class Exercise13
{
    public static void Main()
{
     int tmp;
    Console.Write("\n\n");
    Console.Write("Accept a temperature in centigrade and display a suitable message:\n");
    Console.Write("-------------------------------------------------------------------");
    Console.Write("\n\n");

    Console.Write("Input days temperature : ");
    tmp= Convert.ToInt32(Console.ReadLine());
   if(tmp<0)
             Console.Write("Freezing weather.\n");
   else if(tmp<10)
            Console.Write("Very cold weather.\n");
            else if(tmp<20)
                        Console.Write("Cold weather.\n");
                    else if(tmp<30)
                               Console.Write("Normal in temp.\n");
```

```
Accept a temperature in centigrade and display a suitable message:
--------------------------------------------------------------------

Input days temperature : Its very hot.
```

Input is 48

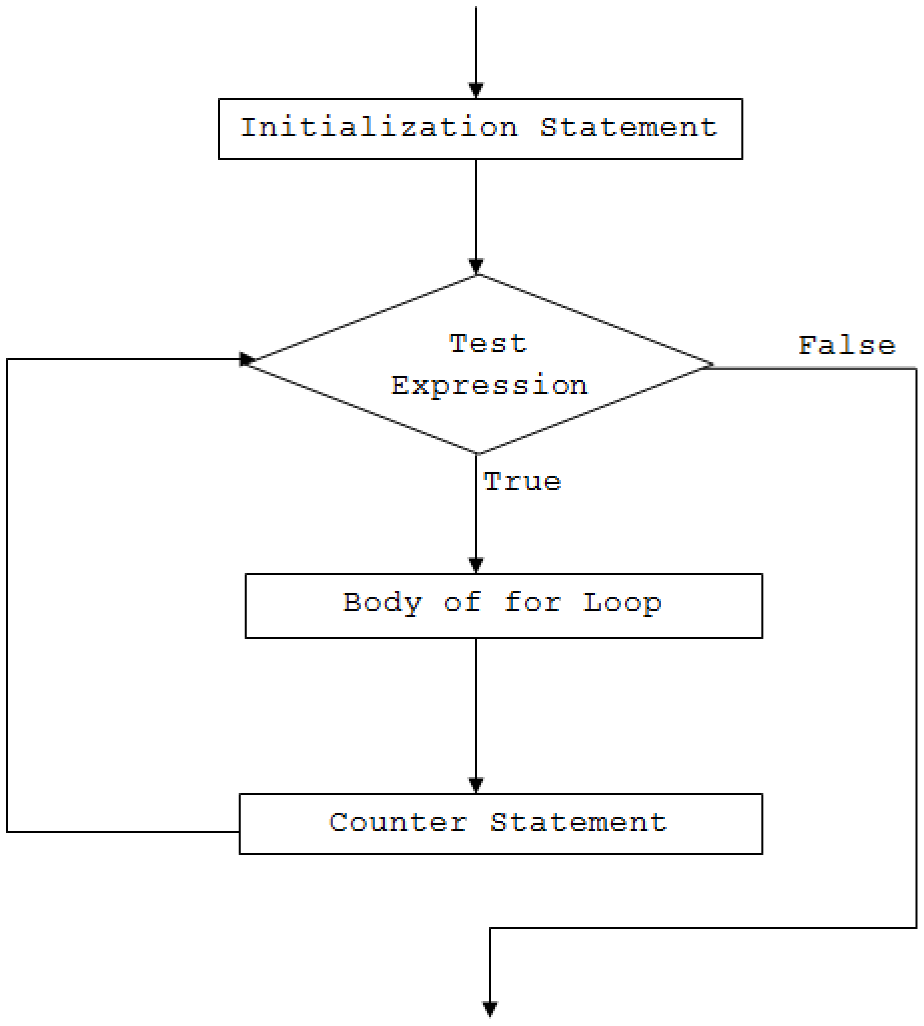
Question no 3

Q3. **a**. What is the role of Loops in C# explain with the help of a flow chart?
Ans) In programming, it is often desired to execute certain block of statements for a specified number of times. A possible solution will be to type those statements for the required number of times

How for loop works?

1. C# for loop has three statements: initialization, condition and iterator.
2. The initialization statement is executed at first and only once. Here, the variable is usually declared and initialized.
3. Then, the condition is evaluated. The condition is a boolean expression, i.e. it returns either true or false.
4. If the condition is evaluated to true:
    a. The statements inside the for loop are executed.
    b. Then, the iterator statement is executed which usually changes the value of the initialized variable.
    c. Again the condition is evaluated.
    d. The process continues until the condition is evaluated to false.

5. If the condition is evaluated to false, the for loop terminates.



b. How many loops are supported by C #, give separate example for each
loop?
Ans) There are three types of loops supported by C#

- **While Loop**
- Do-While loop
- For loop

While Loop:

A while loop is the most straightforward looping structure. The basic format of while loop is as follows:

Syntax

```
while (condition) {
            statements;
}
```

CODE

```
#include<stdio.h>
#include<conio.h>
int main()
{
   int num=1;   //initializing the variable
   while(num<=10) //while loop with condition
   {
         printf("%d\n",num);
         num++;       //incrementing operation
   }
   return 0;
}
```

Do-While loop

A do-while loop is similar to the while loop except that the condition is always executed after the body of a loop. It is also called an exit-controlled loop.

The basic format of while loop is as follows:

<u>Syntax</u>

```
 do {
  statements
} while (expression);
```

<u>CODE:</u>

```
#include<stdio.h>
#include<conio.h>
int main()
{
   int num=1;   //initializing the variable
   do   //do-while loop
   {
         printf("%d\n",2*num);
         num++;       //incrementing operation
   }while(num<=10);
   return 0;
}
```

For loop

A for loop is a more efficient loop structure in 'C' programming. The general structure of for loop is as follows:

Syntax

```
for (initial value; condition; incrementation or
decrementation )
{
  statements;
}
```

Statements

- The initial value of the for loop is performed only once.
- The condition is a Boolean expression that tests and compares the counter to a fixed value after each iteration, stopping the for loop when false is returned.
- The incrementation/decrementation increases (or decreases) the counter by a set value.

CODE

```
#include<stdio.h>
int main()
{
	int number;
	for(number=1;number<=10;number++)	//for
loop to print 1-10 numbers
	{
		printf("%d\n",number);		//to print the
number
	}
	return 0;
}
```

Question no 4

Q4. Why do the developers prefer for loops instead other loops justify your answer with the help of an C # coded program ?
Ans) The basic difference between for loop and other loops is “The one notable difference between a for() and while() loop is that a "continue" statement in a while() loop will branch to the top of the loop, while one in a for()

loop will branch to the third part of the for() clause [the one after the condition, usually used to bump variables.

Explanation

- for-loops are for counting. counting up, counting down.
- while / do-while constructs are for all other conditions. c!=EOF, diff<0.02, etc. Iterators/Enumerators are counters very suitable for for-loops

"Between a WHILE and FOR loop, you can use them interchangeably. To be a purist, you could make your decision base on the nature of the conditions. If you're performing a count-based loop, then a FOR loop would make the most sense."

For loop

```
for( cur = 0; cur < myList.Length; cur++ ){
   doSomething( myList[cur] );
}
```

If you're performing a logic-based loop, then a WHILE would make for the cleanest implementation

While loop

```
Iterator i = myObject.getIterator();
while( i.hasNext() ){
   doSomething( i.next() );
}
```

Question no 5

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

Ans) **Encapsulation:**

It is one of the fundamentals of OOP (object-oriented programming). It refers to the bundling of data with the methods that operate on that data. Encapsulation is used to hide the values or state of a structured data object inside a class, preventing unauthorized parties' direct access to them. Publicly accessible methods are generally provided in the class (so-called getters and setters) to access the values, and other client classes call these methods to retrieve and modify the values within the object.

ROLE OD ENCAPSULATION

Encapsulation is a good idea for several reasons:

- the functionality is defined *in one place* and not in multiple places.
- it is defined in a logical place – the place where the data is kept.
- data inside our object is not modified unexpectedly by external code in a completely different part of our program.
- when we use a method, we only need to know what result the method will produce – we don't need to know details about the object's internals in order to use it. We could switch to using another object which is completely different on the inside, and not have to change any code because both objects have the same interface.

b. Why access specifiers are used in encapsulation justify your answer with the help C # coded example?

Ans) An access specifier defines the scope and visibility of a class member. C# supports the following access specifiers:

1.Public
2.Private
3.Protected
4.Internal
5.Protected internal

Public Access Specifier
Public access specifier allows a class to expose its member variables and member functions to other functions and objects. Any public member can be accessed from outside the class.

Private Access Specifier
Private access specifier allows a class to hide its member variables and member functions from other functions and objects. Only functions of the same class can access its private members. Even an instance of a class cannot access its private members.

Protected Access Specifier

Protected access specifier allows a child class to access the member variables and member functions of its base class. This way it helps in implementing inheritance.

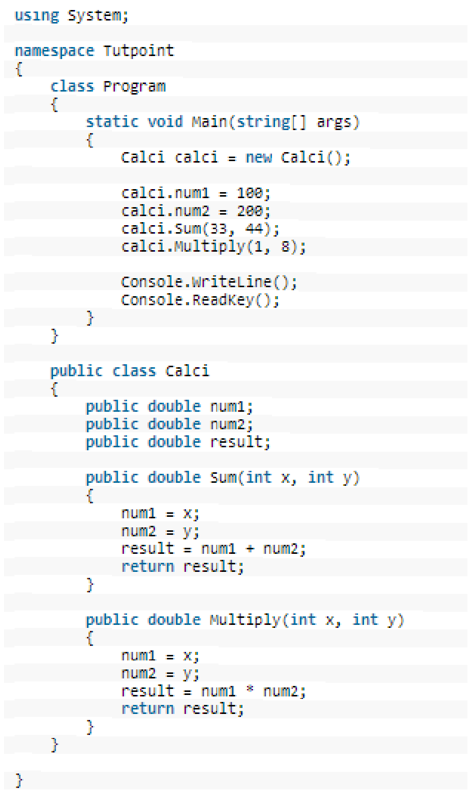
Internal Access Specifier

Internal access specifier allows a class to expose its member variables and member functions to other functions and objects in the current assembly.

Protected Internal Access Specifier

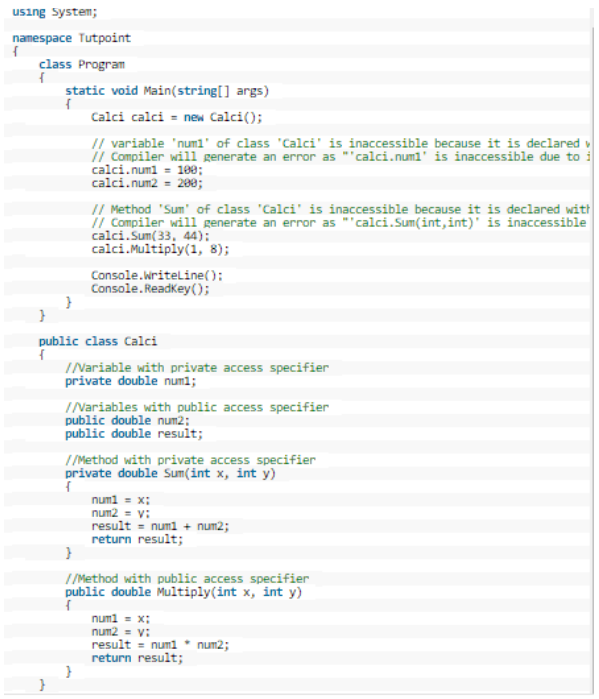
The protected internal access specifier allows a class to hide its member variables and member functions from other class objects and functions, except a child class within the same application. This is also used while implementing inheritance.

Public



```
using System;

namespace Tutpoint
{
    class Program
    {
        static void Main(string[] args)
        {
            Calci calci = new Calci();

            calci.num1 = 100;
            calci.num2 = 200;
            calci.Sum(33, 44);
            calci.Multiply(1, 8);

            Console.WriteLine();
            Console.ReadKey();
        }
    }

    public class Calci
    {
        public double num1;
        public double num2;
        public double result;

        public double Sum(int x, int y)
        {
            num1 = x;
            num2 = y;
            result = num1 + num2;
            return result;
        }

        public double Multiply(int x, int y)
        {
            num1 = x;
            num2 = y;
            result = num1 * num2;
            return result;
        }
    }

}
```

Private



```
using System;

namespace Tutpoint
{
    class Program
    {
        static void Main(string[] args)
        {
            Calci calci = new Calci();

            // variable 'num1' of class 'Calci' is inaccessible because it is declared w
            // Compiler will generate an error as "'calci.num1' is inaccessible due to i
            calci.num1 = 100;
            calci.num2 = 200;

            // Method 'Sum' of class 'Calci' is inaccessible because it is declared with
            // Compiler will generate an error as "'calci.Sum(int,int)' is inaccessible
            calci.Sum(33, 44);
            calci.Multiply(1, 8);

            Console.WriteLine();
            Console.ReadKey();
        }
    }

    public class Calci
    {
        //Variable with private access specifier
        private double num1;

        //Variables with public access specifier
        public double num2;
        public double result;

        //Method with private access specifier
        private double Sum(int x, int y)
        {
            num1 = x;
            num2 = y;
            result = num1 + num2;
            return result;
        }

        //Method with public access specifier
        public double Multiply(int x, int y)
        {
            num1 = x;
            num2 = y;
            result = num1 * num2;
            return result;
        }
    }
```