VISUAL PROGRAMMING LAB 06
WORKBOOK

Instructor
Md. Rashedul Islam

SUBMIT WITHIN: 12:45 PM (TODAY, 18 July 2021)

SEND PROGRAMS IMAGES TO

rashed.class.official17@gmail.com

Email Subject: LAB06_CSC440_ID_SUMMER2021

C# program to calculate the sum of all digits of a number using recursion

Recursion is the process of defining a problem (or the solution to a problem) in terms of (a simpler version of) itself. For example, we can define the operation "find your way home" as:

> If you are at home, stop moving.
>
> Take one step toward home.
>
> "find your way home".

Here the solution to finding your way home is two steps (three steps). First, we don't go home if we are already home. Secondly, we do a very simple action that makes our situation simpler to solve. Finally, we redo the entire algorithm.

All recursive algorithms must have the following:

> Base Case (i.e., when to stop)
>
> Work toward Base Case
>
> Recursive Call (i.e., call ourselves)

The "work toward base case" is where we make the problem simpler (e.g., divide list into two parts, each smaller than the original). The recursive call, is where we use the same algorithm to solve a simpler version of the problem. The base case is the solution to the "simplest" possible problem (For example, the base case in the problem 'find the largest number in a list' would be if the list had only one number... and by definition if there is only one number, it is the largest).

The source code to calculate the sum of all digits of a number using recursion is given below.

```
using System;
using System.Text.RegularExpressions;
namespace CLASS02
{
    class Program
    {

        static void Main()
        {
            int num = 0;
            int sum = 0;

            Console.Write("Enter the number: ");
            num = int.Parse(Console.ReadLine());
```

```
            sum = SumOfDigit(num);

            Console.WriteLine("Sum of digits: " + sum);

            Console.ReadKey();
        }
        public static int SumOfDigit(int number)
        {
            if (number == 0)
            {
                return 0;
            }
            else
            {
                int rem = 0;

                rem = number % 10;
                return (rem + SumOfDigit(number / 10));
            }
        }


    }

}
```

C# program to demonstrate the class and object creation

As we know that, C# is an object oriented programming language. Class and Object are the important part of the Object Oriented programming approach.

In this program, we are going to implement a program using class and object in C#, here we will also use constructor to initialize data members of the class.

Syntax of class:

class <class_name>

{

    //private data members

    //constructors

    //public member functions

}

Syntax of object creation:

<class_name> <object_name> = new <contructor>;

Example:

Sample s = new Sample ();

Note: First alphabet of class name should be capital conventionally.

```
using System;
using System.Text.RegularExpressions;
namespace CLASS02
{
    class Program
    {

        static void Main()
        {
```

```
            Sample S1 = new Sample();

            S1.print();

            Sample S2 = new Sample();

            S2.read();
            S2.print();

            Console.ReadKey();
        }



    }



}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CLASS02
{
    class Sample
    {
        private int X;
        private int Y;

        public Sample()
        {
            X = 5;
            Y = 10;
        }

        public void read()
        {
            Console.Write("Enter value of X: ");
            X = Convert.ToInt32(Console.ReadLine());

            Console.Write("Enter value of Y: ");
            Y = Convert.ToInt32(Console.ReadLine());
        }

        public void print()
        {
            Console.WriteLine("Value of X: " + X);
            Console.WriteLine("Value of Y: " + Y);
        }

    }
}
```

'this' reference in C#.Net with Example

In C#.Net 'this' is a reference of current object, which is accessible within the class only.

To access an element of class by referencing current object of it, we use this keyword, remember following points:

- this keyword is used.
- this cannot be used with the static member functions.

```
using System;
using System.Text.RegularExpressions;
namespace CLASS02
{
    class Program
    {

        static void Main()
        {
            Sample S;

            S = new Sample();

            S.setValues(10, 20);
            S.printValues();

            Sample S1;

            S1 = new Sample();

            S1.setValues(50, 60);
            S1.printValues();

            Console.ReadKey();
        }



    }

    class Sample
    {
        private int a;
        private int b;

        public Sample()
        {
            a = 0;
```

```
            b = 0;
        }

        public void setValues(int a, int b)
        {
            this.a = a;
            this.b = b;
        }
        public void printValues()
        {
            Console.WriteLine("A: " + a + " B: " + b);
        }

    }


}
```

Cascaded Method call in C#

In Object Oriented programming approach, generally we call functions using its object name, for example there is an object named obj of class xyz and method name is myFun() then we can call it by using obj.myFun().

But, in C#.Net, we can call multiple functions in a single statement; it is called cascaded method calling in C#.

We have already disused about this reference in C# (it is a reference of current object), with the help of this reference, we can achieve cascading function calling.

```
using System;
using System.Text.RegularExpressions;
namespace CLASS02
{
    class Program
    {

        static void Main()
        {
            Demo D;

            D = new Demo();

            D.FUN1().FUN2().FUN3();

            Console.ReadKey();
        }



    }

    class Demo
    {

        public Demo FUN1()
        {
            Console.WriteLine("\nFUN1 CALLED");

            return this;
        }

        public Demo FUN2()
        {
            Console.WriteLine("\nFUN2 CALLED");

            return this;
        }
```

```
        public Demo FUN3()
        {
            Console.WriteLine("\nFUN3 CALLED");

            return this;
        }

    }


}
```

C# program for Default Arguments

C#.Net has the concept of Default Arguments, which are also known as Optional Arguments in C#.

Understand the concept of Default Arguments by these points:

Every default argument contains a default value within the function definition.

If we do not pass any argument for default argument then, it uses default value.

Given default value for default argument must be a constant.

Default argument cannot be used for constructor and indexer etc.

```
using System;
using System.Text.RegularExpressions;
namespace CLASS02
{
    class Program
    {

        static void Main()
        {
            Demo D = new Demo();

            //passing one argument other will be assigned
            //with default arguments
            D.setValue(5);
            D.printValue();
            //passing two arguments other will be assigned
            //with default arguments
            D.setValue(5, 8);
            D.printValue();
            //passing all arguemnts
            D.setValue(5, 8, 13);
            D.printValue();

            Console.ReadKey();
        }
    }

    class Demo
    {
        private int a, b, c;

        //function definition with default arguments
        public void setValue(int X, int Y = 10, int Z = 20)
```

```
        {
            a = X;
            b = Y;
            c = Z;
        }

        //printing the values
        public void printValue()
        {
            Console.WriteLine("Values are : " + a + ", " + b + ", " + c);
        }

    }
}
```

Call non-trailing arguments as default argument in C#

As we know that, In C++ we can call only trailing argument as a default argument. But in C# we can call non-trailing argument as default argument. We can make only trailing argument as a default argument, but we can call non-trailing arguments.

To call non-trailing argument as a default argument, we need to use parameter name with colon operator.

```
using System;

namespace CLASS02
{
    class Program
    {

        static void Main()
        {
            EMP E1 = new EMP();

            E1.setEmp("Sandy", 25, salary: 48500);
            E1.printEmp();

            EMP E2 = new EMP();

            E2.setEmp("Mark", a: 33, 34000);
            E2.printEmp();

            Console.ReadKey();
        }
    }

    class EMP
    {
        private string name;
        private int age;
        private int salary;

        public void setEmp(string name, int a = 18, int salary = 20000)
        {
            this.name = name;
            this.age = a;
            this.salary = salary;


        }

        public void printEmp()
        {
            Console.WriteLine("\nEmployee Record: ");
```

```
            Console.WriteLine("\tName  : " + name);
            Console.WriteLine("\tAge   : " + age);
            Console.WriteLine("\tSalary: " + salary);

        }
    }
}
```

Pass object as argument into method in C#

As we know that we can pass primitive (basic) data types in methods as arguments. Similarly, we can pass objects in methods too.

Here is an example that will accept objects as arguments in the methods.

```
using System;

namespace CLASS02
{
    class Program
    {

        static void Main()
        {
            //objects creation
            Sample S1 = new Sample();
            Sample S2 = new Sample();
            Sample S3 = new Sample();

            //passing integers
            S1.setValue(10);
            S2.setValue(20);

            //passing objects
            S3.AddOb(S1, S2);
            //printing the objects
            S1.printValue();
            S2.printValue();
            S3.printValue();

            Console.ReadKey();
        }
    }

    class Sample
    {
        //private data member
        private int value;

        //method to set value
        public void setValue(int v)
        {
            value = v;
        }

        //method to print value
        public void printValue()
        {
            Console.WriteLine("Value : " + value);
```

```
        }
        //method to add both objects, here we are passing
        //S1 and S2 which are objects of Sample class
        public void AddOb(Sample S1, Sample S2)
        {
            //adding the value of S1 and S2,
            //assigning sum in value of current object
            value = S1.value + S2.value;
        }
    }
}
```

Method returning object in C#

This example, shows how a method can return object in C#.Net

```
using System;

namespace CLASS02
{
    class Program
    {

        static void Main()
        {
            //declaring objects
            Sample S1 = new Sample();
            Sample S2 = new Sample();

            //setting values to the objects
            S1.setValue(10);
            S2.setValue(20);

            //adding value of both objects, result will be
            //assigned in the third object
            Sample S3 = S1.AddOb(S2);

            //printing all values
            S1.printValue();
            S2.printValue();
            S3.printValue();

            Console.ReadKey();
        }
    }

       class Sample
       {
              //private data member
              private int value;

              //method to set value
              public void setValue(int v)
              {
                     value = v;
              }

              //method to print value
              public void printValue()
              {
                     Console.WriteLine("Value : " + value);
              }

              //methdo that will return an object
              public Sample AddOb(Sample S1)
```

```
                {
                        //creating object
                        Sample S = new Sample();
                        //adding value of passed object in current object
                        //and adding sum in another object
                        S.value = value + S1.value;
                        //returning object
                        return S;
                }
        }
}
```

Single inheritance

Single inheritance enables a derived class to inherit properties and behavior from a single parent class. It allows a derived class to inherit the properties and behavior of a base class, thus enabling code reusability as well as adding new features to the existing code. This makes the code much more elegant and less repetitive. Inheritance is one of the key features of object-oriented programming (OOP).

Single inheritance is safer than multiple inheritance if it is approached in the right way. It also enables a derived class to call the parent class implementation for a specific method if this method is overridden in the derived class or the parent class constructor.

Here we will create a C# program to demonstrate the Single inheritance. Here we will create the Man, and Employee classes to implement single inheritance.

```
using System;

namespace Employee
{
    class Employee:Man
    {
        public int emp_id;
        public int emp_salary;

        public Employee(int id, int salary, string name, int age) :
base(age, name)
        {
            emp_id = id;
            emp_salary = salary;
        }
        public void Print()
        {
            Console.WriteLine("Emp ID:      " + emp_id);
            Console.WriteLine("Emp Name:    " + name);
            Console.WriteLine("Emp Salary:  " + emp_salary);
            Console.WriteLine("Emp Age:     " + age);
        }
        static void Main(string[] args)
        {
            Employee emp = new Employee(101, 1000, "Rahul", 31);
            emp.Print();

            Console.ReadKey();
        }
    }

    class Man
    {
        public string name;
        public int age;
        public Man(int age, string name)
```

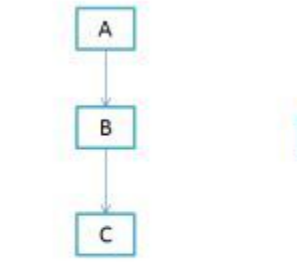
```
            {
                this.name = name;
                this.age = age;
            }
        }
    }
```

In the above program, we created two classes Man, and Employee. Here we inherited Man class into Employee class. Both classes contain constructors to initialize data members. Here we also created one more method Main() in the Employee class. Here we created the object of Employee class and print the Employee detail on the console screen.

Multilevel Inheritance

Multilevel inheritance refers to a mechanism in OO technology where one can inherit from a derived class, thereby making this derived class the base class for the new class. As you can see in below flow diagram C is subclass or child class of B and B is a child class of A. For more details and example refer – Multilevel inheritance in Java.

Multilevel-Inheritance



(d) Multilevel Inheritance

```
using System;

namespace Employee
{
    class Employee:Man
    {
        public int emp_id;
        public int emp_salary;

        public Employee(int id, int salary, string name, int age) : base(age, name)
        {
            emp_id = id;
            emp_salary = salary;
        }
        public void Print()
        {
            Console.WriteLine("Emp ID:      " + emp_id);
            Console.WriteLine("Emp Name:    " + name);
            Console.WriteLine("Emp Salary:  " + emp_salary);
            Console.WriteLine("Emp Age:     " + age);
        }
        static void Main(string[] args)
        {
            Employee emp = new Employee(101, 1000, "Rahul", 31);
            emp.Print();
        }
    }

    class Human
    {
        public string name;
```

```
        public Human(string na)
        {
            name = na;
        }
    }

    class Man : Human
    {
        public int age;
        public Man(int age, string name) : base(name)
        {
            this.age = age;
        }
    }
}
```