Bapatla Engineering College::Bapatla

II/IV B .Tech(supplementary)Degree examination
November 2016

Scheme of evaluation for OOP Using C# (14CSIT306)

---

1. Answer all questions. **1x12=12M**

a. List important features of C#. **(1M)**

1. Simple
2. Modern
3. Object oriented
4. Type safe
5. Interoperability

b. Describe Call Method. **(1M)**

If we want to call an instance method (non-static), we need an instance of the class to call the method .but if we want call non-instance method (static) we need not create an instance of the class. We can call this method by using class name with method name using (dot operator)

c. Write various methods to create string object. **(1M)**

Ans) we can create string object using one of the following methods:

- By assigning a string literal to a String variable
- By using a String class constructor
- By using the string concatenation operator (+)
- By retrieving a property or calling a method that returns a string
- By calling a formatting method to convert a value or an object to its string representation

d. List various operators that cannot be overloaded. **(1M)**

Ans) ., ?:, ->, new, is, sizeof , typeof, +=, -=, *=, /=, %=

e. Use of indexers. **(1M)**

Ans).An **indexer** allows an object to be indexed such as an array. When you define an indexer for a class, this class behaves similar to a **virtual array**. You can then access the instance of this class using the array access operator ([ ]).

f. Write short notes on Enumerations. **(1M)**

Ans).An enumeration is a set of named integer constants. An enumerated type is declared using the **enum** keyword

g. What is exception? **(1M)**An exception is a problem that arises during the execution of a program .such as an attempt to divide by zero.

h. Write various stream classes. **(1M)**
C# defines two types of stream classes. They are,
1. **Byte Stream:** It provides a convenient means for handling input and output of byte.
2. Character Stream: It provides a convenient means for handling input and output of characters. Character stream uses Unicode and therefore can be internationalized.

i. What are events? **(1M)**
Events are user actions such as key press, clicks, mouse movements, etc., or some occurrence such as system generated notifications.

j. Define Namespace. **(1M)**
A **namespace** is designed for providing a way to keep one set of names separate from another.

k. List pre-processor directives in C#. **(1M)**
#define, #undef, #if, #elif, #else, #endif, #line, #error, #region

l. What is meant by collection class? **(1M)**
The Collection classes are specialized classes for data storage and retrieval. These classes provide support for stacks, queues, lists, and hash tables. Most collection classes implement the same interfaces.

UNIT-I

2. a. Describe an Overview of C#. **6M**

C# is a modern, general-purpose, object-oriented programming language developed by Microsoft and approved by European Computer Manufacturers Association (ECMA) and International Standards Organization (ISO).C# was developed by Anders Hejlsberg and his team during the development of .Net Framework. C# is designed for Common Language Infrastructure (CLI). 2M
The following reasons make C# a widely used professional language: 2M

- It is a modern, general-purpose programming language
- It is object oriented.
- It is component oriented.
- It is easy to learn.
- It can be compiled on a variety of computer platforms.
- It is a part of .Net Framework.
  All OOP languages, including C#, have three traits in common: encapsulation, polymorphism, and inheritance. 2M
  Encapsulation: *Encapsulation* is a programming mechanism that binds together code and the data it manipulates, and that keeps both safe from outside interference and misuse.

Polymorphism: *Polymorphism* (from Greek, meaning "many forms") is the quality that allows one interface to access a general class of actions.
Inheritance
Inheritance is the process by which one object can acquire the properties of another object. This is important because it supports the concept of hierarchical classification.

2. b. Explain Arrays with C# programs. 6M

An *array* is a collection of variables of the same type that are referred to by a common name. In C#, arrays can have one or more dimensions. The principal advantage of an array is that it organizes data in such a way that it can be easily manipulated. 2M
One-Dimensional: to declare a one dimensional array, you will typically use this general form:
type[ ] *array-name* = new *type*[*size*];
Here is an example
int[] sample = new int[10];
Example program **Note: Any relevant program may also be considered**

```
// Demonstrate a one-dimensional array.                    2M
using System;
        class ArrayDemo {
        static void Main()
        {
        int[] sample = new int[10];
        int i;
        for(i = 0; i < 10; i = i+1)
        sample[i] = i;
        for(i = 0; i < 10; i = i+1)
        Console.WriteLine("sample[" + i + "]: " + sample[i]);
                }
        }
```

Two-Dimensional Arrays Note: Note: Any relevant program may also be considered
To declare a two-dimensional integer array **table** of size 10, 20, you would write
Syntax:
Type[,] array-variable=new type[rows , colmn];
Example:
int[,] table = new int[10, 20];

```
// Demonstrate a two-dimensional array.                    2M
using System;
        class TwoD
         {
        static void Main() {
        int t, i;
        int[,] table = new int[3, 4];
```

```
                for(t=0; t < 3; ++t)
                {
                for(i=0; i < 4; ++i)
                {
                table[t,i] = (t*4)+i+1;
                Console.Write(table[t,i] + " ");
                }
                Console.WriteLine();
                     }
                   }   }
```

(OR)

3. a. Write any C# program using classes and objects. 6M

// A program that uses the Building class. **Note: Any relevant program may also be considered**

```
using System;
        class Building
         {
        public int Floors; // number of floors
        public int Area; // total square footage of building
        public int Occupants; // number of occupants
        }
                // This class declares an object of type Building.
        class BuildingDemo
         {
        static void Main()
                {
                Building house = new Building(); // create a Building object
                int areaPP; // area per person
                // Assign values to fields in house.
                house.Occupants = 4;
                house.Area = 2500;
                house.Floors = 2;
                // Compute the area per person.
                areaPP = house.Area / house.Occupants;
                Console.WriteLine("house has:\n " +
                house.Floors + " floors\n " +
                house.Occupants + " occupants\n " +
                house.Area + " total area\n " +
                areaPP + " area per person");
                  } }
```

3. b. Describe Strings with C# program. **6M**

String is one of C#'s most important data types because it defines and supports character strings. In many other programming languages, a string is an array of characters. This is not the case with C#. In C#, strings are objects. Thus, **string** is a reference type. Although **string** is a built-in data type in C#, 2M

Constructing Strings: 2M

The easiest way to construct a **string** is to use a string literal. For example, here **str** is a **string** reference variable that is assigned a reference to a string literal:

```
string str = "C# strings are powerful.";
```

In this case, **str** is initialized to the character sequence "C# strings are powerful." You can also create a **string** from a **char** array.

For example:

```
char[] charray = {'t', 'e', 's', 't'};
string str = new string(charray);
```

Once you have created a **string** object, you can use it nearly anywhere that a quoted string is allowed. For example, you can use a **string** object as an argument to **WriteLine( )**,
as shown in this example:

// Introduce string. **Note: Any relevant program may also be considered** 2M

```
using System;
        class StringDemo
        {
        static void Main()
                {
        char[] charray = {'A', ' ', 's', 't', 'r', 'i', 'n', 'g', '.' };
        string str1 = new string(charray);
        string str2 = "Another string.";
        Console.WriteLine(str1);
        Console.WriteLine(str2);
                }
        }
```

UNIT-II

4. a. Write a C# program using Operator Overloading. **6M**

// An example of operator overloading. **Note: Any relevant program may also be considered**

```
using System;
// A three-dimensional coordinate class.
        class ThreeD {
        int x, y, z; // 3-D coordinates
        public ThreeD() { x = y = z = 0; }
        public ThreeD(int i, int j, int k) { x = i; y = j; z = k; }
        // Overload binary +.
        public static ThreeD operator +(ThreeD op1, ThreeD op2)
        {
        ThreeD result = new ThreeD();
        /* This adds together the coordinates of the two points
        and returns the result. */
        result.x = op1.x + op2.x; // These are integer additions
        result.y = op1.y + op2.y; // and the + retains its original
        result.z = op1.z + op2.z; // meaning relative to them.
        return result;
        }
        // Show X, Y, Z coordinates.
        public void Show()
        {
        Console.WriteLine(x + ", " + y + ", " + z);
        }
        }
                class ThreeDDemo
                {
                static void Main()
                 {
                ThreeD a = new ThreeD(1, 2, 3);
                ThreeD b = new ThreeD(10, 10, 10);
                ThreeD c;
                Console.Write("Here is a: ");
                a.Show();
                Console.WriteLine();
                Console.Write("Here is b: ");
                b.Show();
                Console.WriteLine();
                c = a + b; // add a and b together
                Console.Write("Result of a + b: ");
                c.Show();
                Console.WriteLine();
                }
        }
```

4. b. Describe Enumerations. **6M**

An *enumeration* is a set of named integer constants. The keyword **enum** declares an enumerated type. The general form for an enumeration is
enum *name* { *enumeration list* };
Example:
enum Apple { Jonathan, GoldenDel, RedDel , Winesap , Cortland, McIntosh }; 2M
A key point to understand about an enumeration is that each of the symbols stands for an integer value. However, no implicit conversions are defined between an **enum** type and the built-in integer types, so an explicit cast must be used. Also, a cast is required when converting between two enumeration types.
Here is a program that illustrates the **Apple** enumeration: 2M
// Demonstrate an enumeration. **Note: Any relevant program may also be considered** 2M

```
using System;
class EnumDemo {
        enum Apple { Jonathan, GoldenDel, RedDel, Winesap,Cortland, McIntosh };
        static void Main()
                {
                string[] color = {"Red","Yellow","Red","Red","Red","Reddish Green"};
        Apple i; // declare an enum variable
        // Use i to cycle through the enum.
        for(i = Apple.Jonathan; i <= Apple.McIntosh; i++)
        Console.WriteLine(i + " has value of " + (int)i);
        Console.WriteLine();
        // Use an enumeration to index an array.
        for(i = Apple.Jonathan; i <= Apple.McIntosh; i++)
                Console.WriteLine("Color of " + i + " is " +
                color[(int)i]);
                        }
                }
```

(OR)

5. a. Write a C# program for Multiple Inheritance. **6M**

When a derived class is created from more than one base class then that inheritance is called as multiple inheritance. But multiple inheritance is not supported by .net using classes and can be done using interfaces. 2M

Scheme of evaluation for OOP Using C# (14CSIT306)

//Demonstrate interface **Note: Any relevant program may also be considered** 4M

```
using System;
// Define the interface.
        public interface ISeries
         {
        int GetNext(); // return next number in series
        void Reset(); // restart
        void SetStart(int x); // set starting value
                }
        // Use ISeries to implement a series in which each
        // value is two greater than the previous one.
        class ByTwos : ISeries {
                int start;
                int val;
                public ByTwos() {
                start = 0;
                val = 0;
                        }
                public int GetNext() {
                val += 2;
                return val;
                }
                public void Reset() {
                val = start;
                }
                }
                class SeriesDemo2 {
                static void Main() {
                ByTwos twoOb = new ByTwos();
                Primes primeOb = new Primes();
                ISeries ob;
                for(int i=0; i < 5; i++) {
                ob = twoOb;
                Console.WriteLine("Next ByTwos value is " +
                ob.GetNext());
                Console.WriteLine("Next prime number is " +
                ob.GetNext());
                                }
                        }
                }
```

5.b. Write a C# program to demonstrate the use of structure. 6M

```
//Demonstrate a structure.
using System;
// Define a structure.
        struct Book {
        public string Author;
        public string Title;
        public int Copyright;
        public Book(string a, string t, int c) {
        Author = a;
        Title = t;
        Copyright = c;
                }
        }
        // Demonstrate Book structure.
        class StructDemo {
        static void Main() {
        Book book1 = new Book("Herb Schildt","C# 4.0: The Complete Reference",2010);
        //explicit constructor
        Book book2 = new Book(); // default constructor
        Book book3; // no constructor
                Console.WriteLine(book1.Title + " by " + book1.Author +" + book1.Copyright);
                Console.WriteLine();
                if(book2.Title == null)
                Console.WriteLine("book2.Title is null.");
                // Now, give book2 some info.
                book2.Title = "Brave New World";
                book2.Author = "Aldous Huxley";
                book2.Copyright = 1932;
                Console.Write("book2 now contains: ");
                Console.WriteLine(book2.Title + " by " + book2.Author + + book2.Copyright);
                Console.WriteLine();
                // Console.WriteLine(book3.Title); // error, must initialize first
                book3.Title = "Red Storm Rising";
                Console.WriteLine(book3.Title); // now OK
                }
                }
```

UNIT-III

6. a. Describe Redirecting the Standard Streams. 6M

Redirection of the standard streams can be accomplished in two ways. First, when you execute a program on the command line, you can use the **<** and **>** operators to redirect **Console.In** and/or **Console.Out**, respectively. For example, given this program: 2M

```
using System;
        class Test {
        static void Main() {
        Console.WriteLine("This is a test.");
                } }
```

Executing the program like this: Test > log

We can redirect standard input and standard output without making any changes to your program. To do so, you will use the **SetIn( )**, **SetOut( )**, and **SetError( )** methods, shown here, which are members of **Console**: 2M

static void SetIn(TextReader *newIn*)
static void SetOut(TextWriter *newOut*)
static void SetError(TextWriter *newError*)

Thus, to redirect input, call **SetIn( )**, specifying the desired stream. You can use any input stream as long as it is derived from **TextReader**. To redirect output, call **SetOut( )**, specifying the desired output stream, which must be derived from **TextWriter.**

// Redirect Console.Out. **Note: Any relevant program may also be considered** 2M

```
using System;
using System.IO;
        class Redirect {
        static void Main() {
        StreamWriter log_out = null;
        try {
        log_out = new StreamWriter("logfile.txt");
        // Redirect standard out to logfile.txt.
        Console.SetOut(log_out);
        Console.WriteLine("This is the start of the log file.");
        for(int i=0; i<10; i++) Console.WriteLine(i);
        ConsolWriteLine("This is the end of the log file.");
                } catch(IOException exc) {
                Console.WriteLine("I/O Error\n" + exc.Message);
                } finally
                {
                if(log_out != null) log_out.Close();
                }}}
```

6.b. Write a C# program to demonstrate the use of File Stream Classes. **6M**

C# defines two types of stream classes. They are,

Byte Stream: It provides a convenient means for handling input and output of byte.

// Write to a file. **Note: Any relevant program may also be considered** **3M**

```
 using System;
 using System.IO;
        class WriteToFile {
        static void Main(string[] args) {
        FileStream fout = null;
        try {
        // Open output file.
        fout = new FileStream("test.txt", FileMode.CreateNew);
        // Write the alphabet to the file.
        for(char c = 'A'; c <= 'Z'; c++)
        fout.WriteByte((byte) c);
        } catch(IOException exc) {
        Console.WriteLine("I/O Error:\n" + exc.Message);
        } finally {
        if(fout != null) fout.Close();
               }
               }
        }
```

2. **Character Stream:** It provides a convenient means for handling input and output of characters. Character stream uses Unicode and therefore can be internationalized. 3M

// Input from the console using ReadLine().**Note: Any relevant program may also be considered**

```
using System;
       class ReadString {
       static void Main() {
       string str;
       Console.WriteLine("Enter some characters.");
       str = Console.ReadLine();
       Console.WriteLine("You entered: " + str);
              }
       }
```

(OR)

7. Explain Delegates with C# programs. **12M**

Delegate is an object that can refer to a method. Therefore, when you create a delegate, you are creating an object that can hold a reference to a method. Furthermore, the method can be called through this reference. In other words, a delegate can invoke the method to which it refers. It is important to understand that the same delegate can be used to call different methods during the runtime of a program by simply changing the method to which the delegate refers. A delegate type is declared using the keyword **delegate**. 4M

The general form of a delegate declaration is shown here:
delegate *ret-type name*(*parameter-list*); 2M
// A simple delegate example. **Note: Any relevant program may also be considered** 6M

```
using System;
// Declare a delegate type.
        delegate string StrMod(string str);
        class DelegateTest {
        // Replaces spaces with hyphens.
        static string ReplaceSpaces(string s) {
        Console.WriteLine("Replacing spaces with hyphens.");
        return s.Replace(' ', '-');
        }
        // Remove spaces.
        static string RemoveSpaces(string s) {
        string temp = "";
        int i;
        Console.WriteLine("Removing spaces.");
        for(i=0; i < s.Length; i++)
        if(s[i] != ' ') temp += s[i];
        return temp;
        }
        // Reverse a string.
        static string Reverse(string s) {
        string temp = "";
        int i, j;
        Console.WriteLine("Reversing string.");
        for(j=0, i=s.Length-1; i >= 0; i--, j++)
        temp += s[i];
        return temp;
        }
static void Main() {
// Construct a delegate.
StrMod strOp = new StrMod(ReplaceSpaces);
string str;
// Call methods through the delegate.
str = strOp("This is a test.");
Console.WriteLine("Resulting string: " + str);
Console.WriteLine();
strOp = new StrMod(RemoveSpaces);
str = strOp("This is a test.");
Console.WriteLine("Resulting string: " + str);
Console.WriteLine();
strOp = new StrMod(Reverse);
str = strOp("This is a test.");
Console.WriteLine("Resulting string: " + str);
        }
}
```

UNIT-IV

8. a. Describe generic method and Explain with C# program. 6M

It is possible to declare a generic method that uses one or more type parameters of its own. Furthermore, it is possible to create a generic method that is enclosed within a non-generic class.
2M

// Example program **Note: Any relevant program may also be considered** 4M

```
using System;
using System.Collections.Generic;
namespace GenericMethodAppl
{
  class Program
  {
    static void Swap<T>(ref T lhs, ref T rhs)
    {
      T temp;
      temp = lhs;
      lhs = rhs;
      rhs = temp;
    }
   }class demo{
```

```
static void Main(string[] args)
{
   int a, b;
   char c, d;
   a = 10;
   b = 20;
   c = 'I';
   d = 'V';
    //display values before swap:
   Console.WriteLine("Int values before calling swap:");
   Console.WriteLine("a = {0}, b = {1}", a, b);
   Console.WriteLine("Char values before calling swap:");
   Console.WriteLine("c = {0}, d = {1}", c, d);
    //call swap
   Swap<int>(ref a, ref b);
   Swap<char>(ref c, ref d);
    //display values after swap:
   Console.WriteLine("Int values after calling swap:");
   Console.WriteLine("a = {0}, b = {1}", a, b);
   Console.WriteLine("Char values after calling swap:");
   Console.WriteLine("c = {0}, d = {1}", c, d);
    Console.ReadKey();
                 }
                }
            }
```

8. b. Write a C# program using a generic class with two type parameters. 6M

We can declare more than one type parameter in a generic type. To specify two or more type parameters, simply use a comma-separated list. For example, the following **TwoGen** class is a variation of the **Gen** class that has two type parameters: 2M

//Example program **Note: Any relevant program may also be considered** 4M

```
// A simple generic class with two type parameters: T and V.
using System;
        class TwoGen<T, V> {
        T ob1;
        V ob2;
        // Notice that this constructor has parameters of type T and V.
        public TwoGen(T o1, V o2) {
        ob1 = o1;
        ob2 = o2;
        }
    // Show types of T and V.
    public void showTypes() {
    Console.WriteLine("Type of T is " + typeof(T));
    Console.WriteLine("Type of V is " + typeof(V));
    }
    public T getob1() {
    return ob1;
    }
    public V GetObj2() {
    return ob2;
    }
    }
    // Demonstrate two generic type parameters.
    class SimpGen {
        static void Main() {
        TwoGen<int, string> tgObj =
        new TwoGen<int, string>(119, "Alpha Beta Gamma");
        // Show the types.
        tgObj.showTypes();
        // Obtain and show values.
        int v = tgObj.getob1();
        Console.WriteLine("value: " + v);
        string str = tgObj.GetObj2();
        Console.WriteLine("value: " + str);
        }
    }
```

(OR)

9. a. Describe properties and methods of ArrayList class with example program. 6M

Method 2M

1.public virtual void AddRange(ICollection *c*)
 2.public virtual int BinarySearch(object *value*)
3.public virtual int BinarySearch(object *value*,IComparer *comparer*)
 4.public virtual int BinarySearch(int *index*, int *count*, object *value*, IComparer *comparer*)
5.public virtual void CopyTo(Array *array*

Properties: **2M**

Property	Description
Capacity	Gets or sets the number of elements that the ArrayList can contain.
Count	Gets the number of elements actually contained in the ArrayList.
IsFixedSize	Gets a value indicating whether the ArrayList has a fixed size.
IsReadOnly	Gets a value indicating whether the ArrayList is read-only.
Item	Gets or sets the element at the specified index.

// Sort and search an ArrayList. **Note: Any relevant program may also be considered** 2M

```
using System;
using System.Collections;
        class SortSearchDemo {
        static void Main() {
        ArrayList al = new ArrayList();
        al.Add(55);al.Add(43);al.Add(-4);al.Add(88);al.Add(3);al.Add(19);
        Console.Write("Original contents: ");
        foreach(int i in al)
        Console.Write(i + " ");
        Console.WriteLine("\n");
        // Sort
        al.Sort();
        Console.Write("Contents after sorting: ");
        foreach(int i in al)
        Console.Write(i + " ");
        Console.WriteLine("\n");
        Console.WriteLine("Index of 43 is " +
        al.BinarySearch(43));
        } }
```

9. b. Explain Generic Interfaces with C# program. **6M**

Example program: **Note: Any relevant program may also be considered** 6M

```
using System;
public interface IA <T>
{
void Write();
}
class Test<T>:IA <T>
{
   T value;
   public Test(T t)
         {
          value = t;
         }
   public void Write()
          {
                   Console.WriteLine(value);
         }
         }
         class Calculator
         {
         public static bool AreEqual <T>(T value1,T value2)
         {
                   return value1.Equals(value2);
         }
         }
                   class InterfaceDemo
                   {
                            static void Main()
                            {
                   Test<int> test1 = new Test<int>(5);
                   test1.Write();
                   Test<string> test2 = new Test<string>("cat");
                   test2.Write();
                   bool Equal=Calculator.AreEqual<int>(1,1);
                   if(Equal)
                   {
```

```
Console.WriteLine("Equal");
}
else
{
Console.WriteLine("Not Equal");
}
bool Equal1=Calculator.AreEqual<string>("A","B");
if(Equal1)
{
Console.WriteLine("Equal");
}
else
{
Console.WriteLine("Not Equal");
                    }
           }
}
```

Signature of the HOD.

Signature of the internal Examiner (B.Krishnaiah)

Name of the external Examiner	Name of the college	Dept.	Signature