14CS IT 306

Hall Ticket Number:

II/IV B.Tech (Regular/Supplementary) DEGREE EXAMINATION

November, 2017 — **Common for CSE & IT**

Third Semester — **Object Oriented Programming**

Time: Three Hours — **Maximum :** 60 Marks

Answer Question No.1 compulsorily. (1X12 = 12 Marks)

Answer ONE question from each unit. (4X12=48 Marks)

1. Answer all questions (1X12=12 Marks)
   a) What is Encapsulation?
   b) What is Polymorphism?
   c) Differentiate value types and reference types.
   d) List various operators that cannot be overloaded.
   e) Write the uses of base keyword.
   f) Define Exception.
   g) Define Anonymous Methods.
   h) Define delegate.
   i) What is event?
   j) Difference between while and do-while.
   k) List pre-processor directives in C #.
   l) Define Collection.

UNIT I

2. a) Explain C# value types and program control statements with examples. 6M
   b) Write a program to display the Fibonacci series up to given number N in C# . 6M

(OR)

3. a) Explain the three pillars of object-oriented programming in C#? 6M
   b) Define Array. Explain the syntax to declare, initialize and access elements from the following array types with an example.
   i. Single Dimensional Array. ii. Jagged Array 6M

UNIT II

4. a) Explain inheritance in C# with an example. 6M
   b) Explain interface concept with an example. 6M

(OR)

5. a) Write a short note on enumeration and explain its usage with an example. 6M
   b) Explain overloading operators in C# with an example to overload operators +, -, *, > and < 6M

UNIT III

6. a) Write a short notes on a) Console I/O b) Stream class 6M
   b) Write a C # Program to demonstrate the use of FileStream Classes. 6M

(OR)

7. a) Write a C# program to compute and display sum, difference, and multiplication of twonumbers by writing appropriate methods which could be called through multicast delegate methodof programming. 12M

UNIT IV

8. a) What is namespace? Explain the purpose of namespace with an example. 6M
   b) Write a c# program using a generic class with two type parameters. 6M

(OR)

9. a) Describe properties and methods of ArrayList Class with example program. 6M
   b) Explain the C# pre-processor directives with examples. 6M

1. Answer all questions (1X12=12 Marks)

a) **What is 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.

b) **What is Polymorphism?**
Polymorphism (from Greek, meaning "many forms") is the quality that allows one interface to access a general class of actions.

c) **Differentiate value types and reference types.**
Value Type holds the data within its own memory allocation.
Reference Type contains a pointer to another memory location that holds the real data

d) **List various operators that cannot be overloaded.**
&&, ||, [], (T)x(Typecast operator), +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=, as, checked, unchecked, default, delegate, is, new, sizeof, typeof
Give full marks for any four operators.

e) **Write the uses of super keyword.**
1. The base keyword is used to access members of the base class from within a derived class.
2. Specify which base-class constructor should be called when creating instances of the derived class.

f) **Define Exception.**
An exception is an error that occurs at runtime.

g) **Define Anonymous Methods.**
An anonymous method is an unnamed block of code that is associated with a specific delegate instance. An anonymous method is created by following the keyword delegate with a block of code.

h) **Define delegate.**
A delegate is an object that can refer to a method.

i) **What is event?**
An event is, essentially, an automatic notification that some action has occurred in object.

j) **Difference between while and do-while.**
While is an entry controlled loop, means test condition is checked before entering the loop body
Do-While is an exit controlled loop, means test condition is checked after executing the loop body

k) **List pre-processor directives in C #.**
#define, #undef, #if, #else, #elif, #endif, #line, #error, #warning, #region, #endregion
Give full marks for any four directives.

l) **Define Collection.**
A collection is a group of objects.

UNIT I

2. a) **Explain C# value types and program control statements with examples. 6M**
Value Types - 2M Program Control Statements - 4M

Value type variables can be assigned a value directly. They are derived from the class **System.ValueType**.

The value types directly contain data. Some examples are **int, char, and float**, which stores numbers, alphabets, and floating point numbers, respectively. When you declare an **int** type, the system allocates memory to store the value.
The following table lists the available value types in C#

Type	Represents
bool	Boolean value
byte	8-bit unsigned integer
char	16-bit Unicode character
decimal	128-bit precise decimal values with 28-29 significant digits
double	64-bit double-precision floating point type
float	32-bit single-precision floating point type
int	32-bit signed integer type
long	64-bit signed integer type
sbyte	8-bit signed integer type
short	16-bit signed integer type
uint	32-bit unsigned integer type
ulong	64-bit unsigned integer type
ushort	16-bit unsigned integer type

Program Control Statements:

Program Control statements are used to controls the flow of the execution. The flow of control means the order in which a program's statements are executed.
Control statements are classified into three categories, those are

1. Selection statements - if and switch
2. Loop statements - while, do-while, for, foreach
3. Jump statements - break, continue

Selection statements:

Simple if

An **if statement** consists of a boolean expression followed by one or more statements.

syntax

if(condition) statement;

if-else:

An **if statement** can be followed by an optional **else statement**, which executes when the boolean expression is false.

Syntax:

```
if(condition) {
        statement sequence
}
 else {
        statement sequence
}
```

Nested if-else:

You can use one **if** or **else if** statement inside another **if** or **else if**statement(s).

Syntax:

```
if(condition) {
        if(condition) statement;
}
 else {
        statement sequence
}
```

switch statement:

A **switch** statement allows a variable to be tested for equality against a list of values.

Syntax:

```
switch(expression) {
        case constant1: statement sequence break;
        case constant2: statement sequence break;
        case constant3: statement sequence break;
        .
         .
        .
        default: statement sequence break;
}
```

NOTE: Consider any valid example

Loop Statements:

while loop

It repeats a statement or a group of statements while a given condition is true. It tests the condition before executing the loop body.

Syntax:

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

do...while loop:

It is similar to a while statement, except that it tests the condition at the end of the loop body

Syntax:

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

for loop:

It executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.

Syntax:

```
for(initialization; condition; iteration) {
        statement sequence
}
```

foreach loop:

it is used to iterate through arrays or collections

Syntax:

```
foreach(type t in array) {
        statement sequence
}
```

NOTE: Consider any valid example

Jump statements:

break statement:

Terminates the **loop** or **switch** statement and transfers execution to the statement immediately following the loop or switch.

continue statement:

Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

NOTE: Consider any valid example

b) **Write a program to display the Fibonacci series up to given number N in C# .** **6M**

```
using System;
class Fibonacci{
        static void Main(){
                int i, n, t1 = 0, t2 = 1, t3 = 0;
                Console.Write("Enter the value of n: ");
                n = int.Parse(Console.ReadLine());
                Console.WriteLine(t1);
                Console.WriteLine(t2);
                for (i = 0; i <= n; i++)
                {
                        t3 = t1 + t2;
                        Console.WriteLine(t3);
                        t1 = t2;
                        t2 = t3;
                }
        }
}
```

6M

Note: Any relevant program may also be considered

(OR)

3. a) **Explain the three pillars of object-oriented programming in C#?** **6M**

The three Object Oriented Programming features are:

- Encapsulation
- Inheritance
- Polymorphism

2M

Encapsulation:

Encapsulation binds together code and the data it manipulates and keeps them both safe from outside interference and misuse. Encapsulation is a protective container that prevents code and data from being accessed by other code defined outside the container.

2M

Inheritance:

Inheritance is the process by which one object acquires the properties of another object. A type derives from a base type, taking all the base type members fields and functions. Inheritance is most useful when you need to add functionality to an existing type. For example all .NET classes inherit from the System.Object class, so a class can include new functionality as well as use the existing object's class functions and properties as well.

2M

Polymorphism:

Polymorphism is a feature that allows one interface to be used for a general class of action. This concept is often expressed as "one interface, multiple actions". The specific action is determined by the exact nature of circumstances.

b) **Define Array. Explain the syntax to declare, initialize and access elements from the followingarray types with an example.**

i. Single Dimensional Array. **ii. Jagged Array** **6M**

1M

Array:

An array is a data structure that contains several variables of the same type.

An Array has the following properties

- An array can be Single-Dimensional, Multidimensional or Jagged.
- The default value of numeric array elements are set to zero, and reference elements are set to null.
- A jagged array is an array of arrays, and therefore its elements are reference types and are initialized to **null**.
- Arrays are zero indexed: an array with n elements is indexed from 0 to n-1.
- Array elements can be of any type, including an array type.
- Array types are reference types derived from the abstract base type Array.

Single Dimensional Array: **3M**

Syntax:

Declaration:

type[ ] array-name = new type[size];

Initialization:

type[ ] array-name = { val1, val2, val3, ..., valN }; or

int[] nums = new int[] { 99, 10, 100, 18, 78, 23, 63, 9, 87, 49 }; or

int[] nums; nums = new int[] { 99, 10, 100, 18, 78, 23, 63, 9, 87, 49 }; or

int[] nums = new int[10] { 99, 10, 100, 18, 78, 23, 63, 9, 87, 49 };or

int[] nums = new int[10];

nums[0]=1;nums[1]=20];...

Access:

Array-name[index];

Example:

```
using System;
class OneDArray{
        static double Mean(int[] numbers,int noofele){
                int sum=0;
                for(int i=0;i<noofele;i++){
                        sum=sum+numbers[i];
                }
                return ((sum*1.0)/(noofele*1.0));
        }
        static void Main(){
                int[] numbers=new int[10];
                for(int i=0;i<10;i++){
                        Console.Write("numbers["+i+"]=");
                        numbers[i]=int.Parse(Console.ReadLine());
                }
                double mean=Mean(numbers,10);
                Console.WriteLine("Mean of numbers is:"+mean);
        }
}
```

Note: Any relevant program may also be considered

Single Dimensional Array: **2M**

Syntax:

Declaration:

type[ ] [ ] array-name = new type[size][ ];

arrayname[0]=new type[size];

arrayname[1]=new type[size];

arrayname[2]=new type[size]; ...

Initialization:

type[ ] [ ] array-name = new type[size][ ];

arrayname[0]=new type[size]{elements};

Access:

Array-name[index][index];

Example:

```
using System;
class JaggedArray{
        static void Main(){
                int[][] ja = new int[3][];
                ja[0]=new int[3];
                ja[1]=new int[2];
                ja[2]=new int[1];
                for(int r=0;r<3;r++){
                        for(int c=0;c<ja[r].Length;c++)
```

```
                    ja[r][c]=int.Parse(Console.ReadLine());
                }
                for(int r=0;r<ja.Length;r++){
                    for(int c=0;c<ja[r].Length;c++){
                        Console.Write(ja[r][c]);
                    }
                    Console.WriteLine();
                }
            }
        }
```

UNIT II

4. **a) Explain inheritance in C# with an example.** **6M**

Inheritance:

Inheritance is the process by which one object can acquire the properties of another object.

C# supports three types of inheritance

- Single/Simple
- Multilevel
- Hierarchical

Simple/Single inheritance:

One subclasse inherit the features of one superclass (base class).

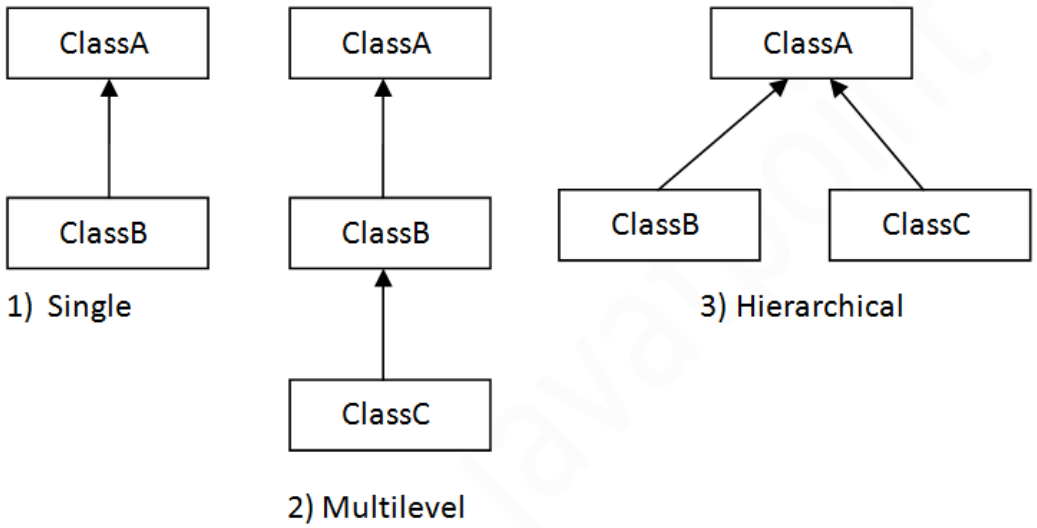
Multilevel:

a subclass is inherited from another subclass.

Hierarchical:

one class serves as a superclass (base class) for more than one sub class.

3M



Example: **Note: Any relevant program may also be considered** **3M**

```
using System;
abstract class BankAccount{
        public String accNo,branch;
        public double ir;
        public decimal bal;
```

```
        public BankAccount(){
                accNo="";
                branch="";
                ir=0.0;
                bal=0.0m;
        }
        public BankAccount(String ano,String br,double i,decimal b){
                accNo=ano;
                branch=br;
                ir=i;
                bal=b;
        }
        abstract public decimal Withdraw(decimal amt);
        public void Deposit(decimal amt){
                bal=bal+amt;
        }
        public virtual String GetType(){
                return "Bank Account";
        }
}
class SBAccount:BankAccount{
        public String accHolderName,accHolderAddress;
        public decimal minBal;
        public SBAccount(String ano,String br,double i,decimal b,String an,String
aa,decimal mb):base(ano,br,i,b){
                accHolderName=an;
                accHolderAddress=aa;
                minBal=mb;
        }

        sealed public override decimal Withdraw(decimal amt){
                if(bal-amt>=minBal){
                        bal-=amt;
                        return bal;
                }
                return -1;
        }
        public virtual String GetType(){
                return "Savings Bank Account";
        }
}
class CBAccount:BankAccount{
        public String accOrgName,accOrgAddress;
        public decimal overdrawLimit;
        public CBAccount(String ano,String br,double i,decimal b,String an,String
aa,decimal ol):base(ano,br,i,b){
```

```
                accOrgName=an;
                accOrgAddress=aa;
                overdrawLimit=ol;
        }
        public override decimal Withdraw(decimal amt){
                if(bal-amt>=overdrawLimit){
                        bal-=amt;
                        return bal;
                }
                return -1;
        }
        public virtual String GetType(){
                return "Current Bank Account";
        }
}
class BankAccountsDemo{
        static void Main(){
                //BankAccount o=new BankAccount();
                //o.Withdraw(1000.0m);
                BankAccount sb=new SBAccount("12345667890","SBI
BEC",4.0,5000.0m,"asdfghjkl","zxcvbnm",500.0m);
                if(sb.Withdraw(500)!=-1)
                        Console.WriteLine("Balance after Withdraw:"+sb.bal);
                else
                        Console.WriteLine("Insuuficiant funds");
                if(sb.Withdraw(6700)!=-1)
                        Console.WriteLine("Balance after Withdraw:"+sb.bal);
                else
                        Console.WriteLine("Insuuficiant funds");
                BankAccount cb=new CBAccount("12345667890","SBI
BEC",4.0,5000.0m,"asdfghjkl","zxcvbnm",500.0m);
                cb.Deposit(1000m);
                if(cb.Withdraw(500)!=-1)
                        Console.WriteLine("Balance after Withdraw:"+cb.bal);
                else
                        Console.WriteLine("Insuuficiant funds");
                if(cb.Withdraw(6700)!=-1)
                        Console.WriteLine("Balance after Withdraw:"+cb.bal);
                else
                        Console.WriteLine("Insuuficiant funds");


        }
}
```

b) **Explain interface concept with an example.** **6M**

3M

An interface contains only the signatures of methods, properties, events or indexers. A class or struct that implements the interface must implement the members of the interface that are specified in the interface definition.

Defining interface:

```
interface name {
    ret-type method-name1(param-list);
    ret-type method-name2(param-list);
    // ...
    ret-type method-nameN(param-list);
}
```

Implementing interface:

```
class class-name : interface-name {
    // class-body
}
```

Example: Note: Any relevant program may also be considered **3M**

```
public interface ISeries {
    int GetNext(); // return next number in series
    void Reset(); // restart
    void SetStart(int x); // set starting value
}
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;
    }
    public void SetStart(int x) {
        start = x;
        val = start;
    }
}

// Demonstrate the ISeries interface.
using System;
class SeriesDemo {
    static void Main() {
        ByTwos ob = new ByTwos();
```

```
        for(int i=0; i < 2; i++)
            Console.WriteLine("Next value is " + ob.GetNext());
        Console.WriteLine("\nResetting");
        ob.Reset();
        for(int i=0; i < 2; i++)
            Console.WriteLine("Next value is " + ob.GetNext());
        Console.WriteLine("\nStarting at 100");
        ob.SetStart(100);
        for(int i=0; i < 2; i++)
            Console.WriteLine("Next value is " + ob.GetNext());
    }
}
```

(OR)

5. a) **Write a short note on enumeration and explain its usage with an example.** **6M**

2M

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 };

4M

Example: Note: Any relevant program may also be considered

```
using System;
class EnumEx{
        enum grades{O,AP,A,BP,B,C,F};
        static void FindGrade(int marks){
                if(marks>=90)
                        Console.WriteLine((int)grades.O);
                else if(marks>=80)
                        Console.WriteLine(grades.AP);
                else if(marks>=70)
                        Console.WriteLine(grades.A);
                else if(marks>=60)
                        Console.WriteLine(grades.BP);
                else if(marks>=50)
                        Console.WriteLine(grades.B);
                else if(marks>=40)
                        Console.WriteLine(grades.C);
                else{
                        int g=(int)grades.F;
                        Console.WriteLine((grades)3);
                }
        }
        static void Main(){
                FindGrade(90);
                FindGrade(20);
        }
}
```

b) **Explain overloading operators in C# with an example to overload operators +, -, *, > and <** **6M**

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

```
using System;
class ThreeDPoint{
        int x,y,z;
        public ThreeDPoint(){
                x=y=z=0;
        }
        public ThreeDPoint(int i,int j,int k){
                x=i;y=j;z=k;
        }
        public void Show(){
                Console.WriteLine("("+x+","+y+","+z+")");
        }
        public static ThreeDPoint operator +(ThreeDPoint p1,ThreeDPoint p2){
                ThreeDPoint p3=new ThreeDPoint();
                p3.x=p1.x+p2.x;
                p3.y=p1.y+p2.y;
                p3.z=p1.z+p2.z;
                return p3;
        }
        public static ThreeDPoint operator -(ThreeDPoint p1,ThreeDPoint p2){
                ThreeDPoint p3=new ThreeDPoint();
                p3.x=p1.x-p2.x;
                p3.y=p1.y-p2.y;
                p3.z=p1.z-p2.z;
                return p3;
        }
        public static ThreeDPoint operator *(ThreeDPoint p1,ThreeDPoint p2){
                ThreeDPoint p3=new ThreeDPoint();
                p3.x=p1.x*p2.x;
                p3.y=p1.y*p2.y;
                p3.z=p1.z*p2.z;
                return p3;
        }
        public static bool operator <(ThreeDPoint p1,ThreeDPoint p2){
                return
Math.Sqrt(p1.x*p1.x+p1.y*p1.y+p1.z*p1.z)<Math.Sqrt(p2.x*p2.x+p2.y*p2.y+p2.z*p2.z);
        }
        public static bool operator >(ThreeDPoint p1,ThreeDPoint p2){
                return
Math.Sqrt(p1.x*p1.x+p1.y*p1.y+p1.z*p1.z)>Math.Sqrt(p2.x*p2.x+p2.y*p2.y+p2.z*p2.z);
        }

}
```

```
class ThreeDPointDemo{
        static void Main(){
                ThreeDPoint p1=new ThreeDPoint();
                Console.WriteLine("Point p1 is:");
                p1.Show();
                ThreeDPoint p2=new ThreeDPoint(10,20,30);
                Console.WriteLine("Point p2 is:");
                p2.Show();

                ThreeDPoint p3=p2+p1;
                Console.WriteLine("p1+p2=");
                p3.Show();
                p3=p2-p1;
                Console.WriteLine("p1-p2=");
                p3.Show();
                p3=p2*p1;
                Console.WriteLine("p1*p2=");
                p3.Show();
                Console.WriteLine("p1<p2"+(p1<p2));
                Console.WriteLine("p1>p2"+(p1>p2));
        }
}
```

UNIT III

6. a) **Write a short notes on a) Console I/O b) Stream class** **6M**

Console I/O is accomplished through the standard streams Console.In, Console.Out, and Console.Error.

Reading Console Input

Console.In is an instance of TextReader, and you can use the methods and properties defined by TextReader to access it. However, you will usually use the methods provided by Console, which automatically read from Console.In. Console defines three input methods. The first two, Read( ) and ReadLine( ), have been available since .NET Framework 1.0. The third, ReadKey( ), was added by .NET Framework 2.0.

3M

Writing Console Output

Console.Out and Console.Error are objects of type TextWriter. Console output is most easily accomplished with Write( ) and WriteLine( ), with which you are already familiar. Versions of these methods exist that output each of the built-in types. Console defines its own versions of Write( ) and WriteLine( ) so they can be called directly on Console, as you have been doing throughout this book. However, you can invoke these (and other) methods on the TextWriter that underlies Console.Out and Console.Error, if you choose.

Stream class: **3M**

The core stream class is System.IO.Stream. Stream represents a byte stream and is a base class for all other stream classes. It is also abstract, which means that you cannot instantiate a Stream

object. Stream defines a set of standard stream operations. Following table shows several commonly used methods defined by Stream. Several of the methods shown in following table will throw an IOException if an I/O error occurs. If an invalid operation is attempted, such as attempting to write to a stream that is read-only, a NotSupportedException is thrown. Other exceptions are possible, depending on the specific method.

Method	Description
void Close( )	Closes the stream.
void Flush( )	Writes the contents of the stream to the physical device.
int ReadByte( )	Returns an integer representation of the next available byte of input. Returns –1 when the end of the file is encountered.
int Read(byte[ ] *buffer*, int *offset*, int *count*)	Attempts to read up to *count* bytes into *buffer* starting at *buffer*[*offset*], returning the number of bytes successfully read.
long Seek(long *offset*, SeekOrigin *origin*)	Sets the current position in the stream to the specified *offset* from the specified *origin*. It returns the new position.
void WriteByte(byte *value*)	Writes a single byte to an output stream.
int Write(byte[ ] *buffer*, int *offset*, int *count*)	Writes a subrange of *count* bytes from the array *buffer*, beginning at *buffer*[*offset*], returning the number of bytes written.

To determine the capabilities of a stream, you will use one or more of Stream's properties. They are shown in Following table. Also shown are the Length and Position properties, which contain the length of the stream and its current position.

Property	Description
bool CanRead	This property is true if the stream can be read. This property is read-only.
bool CanSeek	This property is true if the stream supports position requests. This property is read-only.
bool CanTimeout	This property is true if the stream can time out. This property is read-only.
bool CanWrite	This property is true if the stream can be written. This property is read-only.
long Length	This property contains the length of the stream. This property is read-only.
long Position	This property represents the current position of the stream. This property is read/write.
int ReadTimeout	This property represents the length of time before a time-out will occur for read operations. This property is read/write.
int WriteTimeout	This property represents the length of time before a time-out will occur for write operations. This property is read/write.

b) **Write a C # Program to demonstrate the use of FileStream Classes.** **6M**
Note: Any relavent program may also be considered **6M**

```
using System;
using System.IO;
class FileCopy{
        static void Main(String[] args){
                FileStream fs1,fs2;
```

```
                int ch;
                if(args.Length<2){
                        Console.WriteLine("Error\nUsage:ProgramName File1 File2");
                        return;
                }
                try{
                        fs1=new FileStream(args[0],FileMode.Open,FileAccess.Read);
                        fs2=new FileStream(args[1],FileMode.Create,FileAccess.Write);
                        while((ch=fs1.ReadByte())!=-1){
                                fs2.WriteByte((byte)ch);
                        }
                        fs1.Close();
                        fs2.Close();
                }catch(FileNotFoundException ex){
                        Console.WriteLine("File is Not Existed:\n"+ex);
                }catch(IOException ex){
                        Console.WriteLine("IO Failure:\n"+ex);
                }catch(Exception ex){
                        Console.WriteLine("General Exception:\n"+ex);
                }
        }
}
```

(OR)

7. **Write a C# program to compute and display sum, difference, and multiplication of two numbers by writing appropriate methods which could be called through multicast delegate method of programming.** **12M**

Note: Any relevant program may also be considered **12M**

```
using System;
delegate void ArithmaticOperations(int n1,int n2);
class MulticastDelegateExample
{
        static string result = "";
        public static void Sum(int n1, int n2) {
                result += "sum of" + n1 + "," + n2 + "=" + (n1 + n2);
        }
        public static void Difference(int n1, int n2)
        {
                result += "\nDifference between" + n1 + "," + n2 + "=" + (n1 - n2);
        }
        public static void Product(int n1, int n2)
        {
                result += "\nProduct of" + n1 + "," + n2 + "=" + (n1 * n2);
        }
        static void Main(string[] args)
        {
                ArithmaticOperations ao=Sum;
```

```
            ao += Difference;
            ao += Product;
            ao(10, 20);
            Console.WriteLine(result);
        }
    }
```

UNIT IV

8. a) **What is namespace? Explain the purpose of namespace with an example.** **6M**

2M A namespace defines a declarative region that provides a way to keep one set of names separate from another. In essence, names declared in one namespace will not conflict with the same names declared in another. The namespace used by the .NET Framework library (which is the C# library) is System. This is why you have included using System;

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

```
//Shapes.cs which contains Circle and Rectangle classes under the name space Shapes
using System;
namespace Shapes{
    class Circle{
        int radius;
        public Circle(int r){
            radius=r;
        }
        public void Area(){
            Console.WriteLine("Area is:"+(3.14*radius*radius));
        }
    }
    class Rectangle{
        int length,breadth;
        public Rectangle(int l,int b){
            length=l;
            breadth=b;
        }
        public void Area(){
            Console.WriteLine("Area is:"+(length*breadth));
        }
    }
}

//Shapes1.cs which contains same Circle and Rectangle classes under the name space Shapes1
using System;
namespace Shapes1{
    class Circle{
        int radius;
        public Circle(int r){
            radius=r;
        }
```

```
        public void Area(){
            Console.WriteLine("Area is:"+(3.14*radius*radius));
        }
    }
    class Rectangle{
        int length,breadth;
        public Rectangle(int l,int b){
            length=l;
            breadth=b;
        }
        public void Area(){
            Console.WriteLine("Area is:"+(length*breadth));
        }
    }
}
using Shapes;
class Demo{
    static void Main(){
        Circle c=new Circle(5);
        c.Area();
        Rectangle r=new Rectangle(5,5);
        r.Area();
    }
}
```

b) **Write a c# program using a generic class with two type parameters.** **6M**

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

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

```
using System;
using System.Collections;
class ArrayListDemo
{
    static void Print(ArrayList Countries)
    {
        for(int i=0;i<Countries.Count;i++)
        Console.Write(Countries[i]+" ");
        Console.WriteLine();
    }
    static void Main()
    {
        ArrayList Countries=new ArrayList();
        Countries.Add("India");
        Countries.Add("Indonesia");
        Countries.Add("Argentina");
        Countries.Add("FinLand");
        Countries.Add("Kenya");
        Countries.Add("Brazil");
        Countries.Add("Bangladesh");
        Countries.Add("NewZealand");
        Countries.Add("England");
        Countries.Add("Srilanka");
        Countries.Add("Pakistan");
        Print(Countries);
        foreach(String c in Countries)
        Console.Write(c+" ");
```

```
            Console.WriteLine();
            IEnumerator en=Countries.GetEnumerator();
            while(en.MoveNext())
        Console.Write(en.Current+" ");
            Console.WriteLine();
            Countries.RemoveAt(1);
            //Countries.Remove("France");
            Countries.Insert(0,"America");
            for(int i=0;i<Countries.Count;i++)
        Console.Write(Countries[i]+" ");
            Console.WriteLine();
            if(Countries.Contains("Brazil"))
        Console.WriteLine("Brazil is Present");
            int k=Countries.IndexOf("England");
            Console.WriteLine("England is present at "+k);
            Countries.Sort();
            Print(Countries);
            k=Countries.BinarySearch("England");
            Console.WriteLine("England is Present at "+k);
            Countries.Reverse();
            Print(Countries);
            ArrayList clist=Countries.GetRange(2,5);
            Print(clist);
            String[] someCountries=new String[4];
            Countries.CopyTo(2,someCountries,0,4);
            for(int i=0;i<someCountries.Length;i++)
        Console.Write(someCountries[i]+" ");
            Console.WriteLine();
            Countries.RemoveRange(0,5);
            Print(Countries);
        }
}
```

b) **Explain the C# pre-processor directives with examples.** **6M**

he following table lists the preprocessor directives available in C# −

SNO	Preprocessor Directive & Description
1	**#define** It defines a sequence of characters, called symbol.
2	**#undef** It allows you to undefine a symbol.
3	**#if** It allows testing a symbol or symbols to see if they evaluate to true.

4	**#else** It allows to create a compound conditional directive, along with #if.
5	**#elif** It allows creating a compound conditional directive.
6	**#endif** Specifies the end of a conditional directive.
7	**#line** It lets you modify the compiler's line number and (optionally) the file name output for errors and warnings.
8	**#error** It allows generating an error from a specific location in your code.
9	**#warning** It allows generating a level one warning from a specific location in your code.
10	**#region** It lets you specify a block of code that you can expand or collapse when using the outlining feature of the Visual Studio Code Editor.
11	**#endregion** It marks the end of a #region block.

Example: Note: Any relevant program may also be considered

```
#define PI
using System;
namespace PreprocessorDAppl {
  class Program {
    static void Main(string[] args) {
      #if (PI)
        Console.WriteLine("PI is defined");
      #else
        Console.WriteLine("PI is not defined");
      #endif
      Console.ReadKey();
    }
  }
}
```