**Section 1**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Text.RegularExpressions;

using System.Threading.Tasks;

namespace BitFunConsoleApplication

{

class Program

{

static void Main(string[] args)

{

// Test Case 1:

string outputStr = string.Empty;

String str1 = "The following text<C><B>is centered and in boldface</B></C>";

bool isValid = (isValidHTMLTag(str1));

if (isValid == true)

{

outputStr = "Correctly tagged paragraph";

}

else

{

outputStr = "Expected </C> found </B>";

}

Console.WriteLine(outputStr);

Console.ReadKey();

}

public static Boolean isValidHTMLTag(String str)

{

// Regex to check valid HTML tag.

String regex = "<(\"[^\"]\*\"|'[^']\*'|[^'\">])\*>";

Boolean isMatch = Regex.IsMatch(str, regex);

if (str == null)

{

return false;

}

return isMatch;

}

}

}

**Section 2**

Question 1

ANS:

This code is QuickSort Like Merge Sort, QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

Always pick first element as pivot.

Always pick last element as pivot (implemented below)

Pick a random element as pivot.

Pick median as pivot.

The key process in quickSort is partition(). Target of partitions is, given an array and an element x of array as pivot, put L at its correct position in sorted array and put all smaller elements (smaller than L) before L, and put all greater elements (greater than L) after L. All this should be done in linear time.

Question 2

ANS:

**Section 3**

Question 1

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BitfontainConsoleApplication

{

class Program

{

static void Main(string[] args)

{

string files = string.Empty;

foreach (string file in GetFiles("C:\\temp"))

{

files += file + Environment.NewLine;

Console.WriteLine(file);

}

Console.WriteLine(files);

Console.ReadKey();

}

static IEnumerable<string> GetFiles(string path)

{

Queue<string> queue = new Queue<string>();

queue.Enqueue(path);

while (queue.Count > 0)

{

path = queue.Dequeue();

try

{

foreach (string subDir in Directory.GetDirectories(path))

{

queue.Enqueue(subDir);

}

}

catch (Exception ex)

{

Console.Error.WriteLine(ex);

}

string[] files = null;

try

{

files = Directory.GetFiles(path);

}

catch (Exception ex)

{

Console.Error.WriteLine(ex);

}

if (files != null)

{

for (int i = 0; i < files.Length; i++)

{

yield return files[i];

}

}

}

}

}

}

**Output:**

**C:\temp\printerinst.log**

**C:\temp\vncserver-sservice.log**

**C:\temp\vncserver-sservice.log.bak**

**C:\temp\data\exceldata.xls**

**C:\temp\data\tesd.txt**

Question 2

Design a database to keep track of student’s subjects and marks. Each student will be enrolled in 1 or more courses and will have written 0 or more tests per course.

Student

PK

FK

Enrolled

Enrolled Id

Student Number

Scheduled Id

Test schedule

Studd

PK

FK

Scheduled Id

Student Name

Test date

Test Marks

Course Number

Student Number

Student Name

PK

PK

Courses

Course Number

Course Name

**Section 4**

Question 1

What is shared memory and when will you use it?

ANS:

Shared memory is a memory shared between two or more processes. Shared memory is a method by which program processes can exchange data more quickly than by reading and writing using the regular operating system services.

For example, a client process may have data to pass to a server process that the server process is to modify and return to the client. Ordinarily, this would require the client writing to an output file (using the buffers of the operating system) and the server then reading that file as input from the buffers to its own work space. Using a designated area of shared memory, the data can be made directly accessible to both processes without having to use the system services. To put the data in shared memory, the client gets access to shared memory after checking a semaphore value, writes the data, and then resets the semaphore to signal to the server (which periodically checks shared memory for possible input) that data is waiting. In turn, the server process writes data back to the shared memory area, using the semaphore to indicate that data is ready to be read.

Question 2

What are the restrictions, if any, in C# OOP?

ANS:

**Restrictions of C# OOP:**

* A property cannot be passed via ref or out parameter to a method.
* A property cannot be overloaded. It means that one can only put a single get and set accessor and mutator in a class respectively.
* A property should not alter the state of the underlying variable when the get accessor is called.

Question 3

Give an example, in your own words, of polymorphism and when it should be used.

ANS:

Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.

Like we specified in the previous chapter; Inheritance lets us inherit fields and methods from another class. Polymorphism uses those methods to perform different tasks. This allows us to perform a single action in different ways.

For example, think of a base class called Animal that has a method called animalSound(). Derived classes of Animals could be Pigs, Cats, Dogs, Birds - And they also have their own implementation of an animal sound (the pig oinks, and the cat meows, etc.)

class Animal // Base class (parent)

{

public virtual void animalSound()

{

Console.WriteLine("The animal makes a sound");

}

}

class Pig : Animal // Derived class (child)

{

public override void animalSound()

{

Console.WriteLine("The pig says: wee wee");

}

}

class Dog : Animal // Derived class (child)

{

public override void animalSound()

{

Console.WriteLine("The dog says: bow wow");

}

}

class Program

{

static void Main(string[] args)

{

Animal myAnimal = new Animal(); // Create a Animal object

Animal myPig = new Pig(); // Create a Pig object

Animal myDog = new Dog(); // Create a Dog object

myAnimal.animalSound();

myPig.animalSound();

myDog.animalSound();

}

}

**Output:**

The animal makes a sound

The pig says: wee wee

The dog says: bow wow

The output from the example above was probably not what you expected (if do not use virtual, override). That is because the base class method overrides the derived class method, when they share the same name.

C# provides an option to override the base class method, by adding the virtual keyword to the method inside the base class, and by using the override keyword for each derived class methods

**Question 4**

Please provide a practical example of interface implementation in C#, describing exactly what benefits you would achieve through your design.

ANS:

An interface is a completely "abstract class", which can only contain abstract methods and properties (with empty bodies.

It is considered good practice to start with the letter "I" at the beginning of an interface, as it makes it easier for yourself and others to remember that it is an interface and not a class.

By default, members of an interface are abstract and public.

To access the interface methods, the interface must be implemented to inherited by another class. To implement an interface, use the : symbol inheritance. The body of the interface method is provided by the "implement" class. Note that you do not have to use the override keyword when implementing an interface.

interface IAnimal

{

void animalSound(); // interface method (does not have a body)

}

class Pig : IAnimal

{

public void animalSound()

{

Console.WriteLine("The pig says: wee wee");

}

}

class Program

{

static void Main(string[] args)

{

Pig myPig = new Pig();

myPig.animalSound();

}

}

**Benefits:**

1) To achieve security - hide certain details and only show the important details of an object (interface).

2) C# does not support "multiple inheritance" (a class can only inherit from one base class). However, it can be achieved with interfaces, because the class can implement multiple interfaces. Note: To implement multiple interfaces, separate them with a comma.

**Question 5**

In your experience what aspects of object orientated development has yielded the most

maintainable, versatile, flexible solution. Provide examples where possible.

ANS:

Object-oriented programming (OOP) can be valuable when developing and maintaining software programs. Object orientated development has yielded the most maintainable, versatile, flexible solution. Some explanations are as following:-

**1. Improved software-development productivity**: Object-oriented programming is modular, as it provides separation of duties in object-based program development. It is also extensible, as objects can be extended to include new attributes and behaviors. Objects can also be reused within an across applications. Because of these three factors – modularity, extensibility, and reusability – object-oriented programming provides improved software-development productivity over traditional procedure-based programming techniques.

**2. Improved software maintainability**: For the reasons mentioned above, objectoriented software is also easier to maintain. Since the design is modular, part of the system can be updated in case of issues without a need to make large-scale changes.

**3. Faster development: Reuse enables faster development**. Object-oriented programming languages come with rich libraries of objects, and code developed during projects is also reusable in future projects.

**4. Lower cost of development: The reuse of software also lowers the cost of development**. Typically, more effort is put into the object-oriented analysis and design, which lowers the overall cost of development.

**5. Higher-quality software**: Faster development of software and lower cost of development allows more time and resources to be used in the verification of the software. Although quality is dependent upon the experience of the teams, object oriented programming tends to result in higher-quality software.