Section 4:

Question 1:

Shared memory is a method by which a program method can exchange data more quickly than by reading and writing using the regular operating system services. Generally, when a client process passes data to a server process ,the server process modifies and returns 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 workspace. 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:

In OOP, we can scope and restrict access to an object's properties by encapsulating it with various property types, to achieve security and abstraction. There are several access modifiers like , public, private , protected, internal etc. to give restrictions on accessing classes and properties.

In Using public as an access modifier,the level/visibility for classes, fields, methods and properties can be set accessible for all classes. To achieve more , we can use , protected
Modifier, which is accessible from the same class and classes, which inherits this class.
private is accessible only from the same class. Internal classes are accessible within the same assembly. We have also two combined modifiers : protected internal and private protected

Question 3:

Polymorphism, in terms of programming, means reusing a single code multiple times. Polymorphism has several advantages, like, It’s great for reusing the same code. It allows us to do a single action in different ways. We could also use a single variable to store multiple data types.

I’m using a simple example to demonstrate the benefit of using it.

```
class Human  // Base class (parent)
{
  public virtual void selfaTalk()
  {
    Console.WriteLine("Hello from Human");
  }
}

class Man: Human  // Derived class (child)
{
  public override void selfaTalk()
  {
    Console.WriteLine("Hello from  Type Man");
  }
}

class Woman: Human   // Derived class (child)
{
  public override void selfaTalk()
  {
    Console.WriteLine("Hello from Type Man");
  }
}

class Program
{
  static void Main(string[] args)
  {
    Human justHuman= new Human();  // Create a Human object
    Human man = new Man();  // Create a Man object
    Human women= new Woman();  // Create a Woman object

    justHuman.selfaTalk();
    man.selfaTalk();
    women.selfaTalk();
  }
}
```

We could always easily override the base Class method and use it for every inherited class to use as our demand.

Question 4:

I work in ASP.NET Core, and In every solution, we use Interface.

As we know, Interface is like a contract: Inheriting It means, we have to accept all inside the contract itself, by implementing it in our derived class.

In Our program, We have a View, Controller and to access the Data layer, we have a service. We access our data layers from controllers via service classes, Which all are instantiated in our controller via interface by loosely coupling. By using interfaces, we are able to instantiate a group of related classes via a single interface. Also,it's easy to give a structure for a group of classes which are inherited by interface. We could easily get the idea of a class just viewing the interface.

Question 5:

In Object-Oriented Programming ,we can build the programs from standard working modules that communicate with one another, rather than having to start writing the code from scratch which leads to saving of development time and higher productivity. We could reuse the chunk or program as per our need. It gives a proper structure to our program. OOP allows us to break the program into bit-sized problems that can be solved easily. systems can be easily upgraded from small to large systems.We are able to redundant our code by reuse the same piece of code over and over.