Winter 2019 – Final Examination

IT602 – Object Oriented Programming

ID: …….. Name: …….

Max Marks: 50 Time: 2½ hours

Instructions:

1. All questions are compulsory.
2. Marks carried by each question are mentioned alongside the question.
3. The answers are to be written on the question paper itself in space provided below every question.
4. Supplementary sheet provided is only for rough work and should NOT be attached to the question paper.

Q1. (2+8)

(a) What is the output of the following function when called from Main()?

```
static void QueryOverInts()
{
    int[] numbers = { 20, 80, 10, 50, 30, 60, 40, 70, 90 };

    var less = numbers.Where(n => n > numbers[2]).Select(n=>n);
    var intFactor = numbers.Select(n => n / numbers.Min()).ToArray();

    numbers[2] = 40;

    foreach (var i in less)
        Console.Write("{0} ", i);
    Console.WriteLine();
    Console.WriteLine("{0}", intFactor.Sum());
}
```

// your answer goes here:

(b) Following is the output of the program which follows.

```
Elective 1 Topper:
Id=201712004, Name=NILESHKUMAR

------Elective 1 Percentage wise list-----------
```

```
NILESHKUMAR - 77.5
DHRUV - 75
SHASHANK - 68.75
KISHAN - 67.5
RASHMI - 66.25
DHRUVIL - 63.75
HIMANSHU - 63.75
ARCHAN - 58.75
YASH - 50
SAHIL - 26.25
Name of students in both subjects:
ARCHAN
SAHIL
NILESHKUMAR
DHRUV
RASHMI
Students in only elective1:
DHRUVIL
HIMANSHU
KISHAN
YASH
SHASHANK
Total Students: 15
```

Using LINQ, complete the following code so that the output of the program is as shown above. Note that you are not allowed to change code other than at spaces indicated by comments. You can use either of the two LINQ syntaxes (including mixing both).

```
class Program
{
    static void Main(string[] args)
    {
        SemesterResult.Run();
    }
}
class Student
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int InSem1Marks { get; set; }
    public int InSem2Marks { get; set; }
    public int FinalExamMarks { get; set; }
    public double Percent { get; set; }
}

class SemesterResult
{
    private List<Student> _students;
    public static void Run()
    {
        Student[] elective1 = new Student[10];

        elective1[0] = new Student { Id = 201712001, Name = "ARCHAN", InSem1Marks =
9, InSem2Marks = 15, FinalExamMarks = 23 };
        elective1[1] = new Student { Id = 201712002, Name = "DHRUVIL", InSem1Marks =
8, InSem2Marks = 17, FinalExamMarks = 26 };
```

```
            elective1[2] = new Student { Id = 201712003, Name = "SAHIL", InSem1Marks = 4,
InSem2Marks = 14, FinalExamMarks = 3 };
            elective1[3] = new Student { Id = 201712004, Name = "NILESHKUMAR",
InSem1Marks = 11, InSem2Marks = 16, FinalExamMarks = 35 };
            elective1[4] = new Student { Id = 201712005, Name = "DHRUV", InSem1Marks =
10, InSem2Marks = 17, FinalExamMarks = 33 };
            elective1[5] = new Student { Id = 201712006, Name = "HIMANSHU", InSem1Marks =
10, InSem2Marks = 17, FinalExamMarks = 24 };
            elective1[6] = new Student { Id = 201712007, Name = "KISHAN", InSem1Marks =
9, InSem2Marks = 15, FinalExamMarks = 30 };
            elective1[7] = new Student { Id = 201712008, Name = "YASH", InSem1Marks = 7,
InSem2Marks = 16, FinalExamMarks = 17 };
            elective1[8] = new Student { Id = 201712009, Name = "RASHMI", InSem1Marks =
8, InSem2Marks = 18, FinalExamMarks = 27 };
            elective1[9] = new Student { Id = 201712010, Name = "SHASHANK", InSem1Marks =
14, InSem2Marks = 16, FinalExamMarks = 25 };



            SemesterResult e1Result = new SemesterResult(elective1.ToList());

            Console.WriteLine("Elective 1 Topper:");
            var topper = e1Result.HighestMarks();
            Console.Write($"Id={topper.id}, Name={topper.name}\n\n");

            Console.WriteLine("------Elective 1 Percentage wise list-----------");
            var all = e1Result.PercentDescending();
            all.ForEach(one => Console.Write($"{one.Name} - {one.Percent}\n"));



            Student[] elective2 = new Student[10];
            elective2[0] = new Student { Id = 201712001, Name = "ARCHAN", InSem1Marks =
10, InSem2Marks = 14, FinalExamMarks = 27 };
            elective2[1] = new Student { Id = 201712012, Name = "SONAL", InSem1Marks = 6,
InSem2Marks = 15, FinalExamMarks = 29 };
            elective2[2] = new Student { Id = 201712003, Name = "SAHIL", InSem1Marks = 9,
InSem2Marks = 12, FinalExamMarks = 13 };
            elective2[3] = new Student { Id = 201712004, Name = "NILESHKUMAR",
InSem1Marks = 17, InSem2Marks = 15, FinalExamMarks = 33 };
            elective2[4] = new Student { Id = 201712005, Name = "DHRUV", InSem1Marks =
14, InSem2Marks = 13, FinalExamMarks = 34 };
            elective2[5] = new Student { Id = 201712016, Name = "MOHAN", InSem1Marks = 8,
InSem2Marks = 14, FinalExamMarks = 30 };
            elective2[6] = new Student { Id = 201712017, Name = "BRIJESH", InSem1Marks =
14, InSem2Marks = 13, FinalExamMarks = 30 };
            elective2[7] = new Student { Id = 201712018, Name = "SIMON", InSem1Marks =
12, InSem2Marks = 16, FinalExamMarks = 23 };
            elective2[8] = new Student { Id = 201712009, Name = "RASHMI", InSem1Marks =
11, InSem2Marks = 15, FinalExamMarks = 32 };
            elective2[9] = new Student { Id = 201712019, Name = "VINOD", InSem1Marks =
13, InSem2Marks = 11, FinalExamMarks = 28 };

            // prepare list of students who took both electives

// write LINQ code here
```

```
            Console.WriteLine("Name of students in both subjects:");
// the common variable is a list coming as output of the LINQ code you write above
            common.ForEach(one => Console.Write($"{one}\n"));

            // prepare list of students who took only elective 1
// write LINQ code here









            Console.WriteLine("Students in only elective1:");
            onlyElective1.ForEach(one => Console.Write($"{one}\n"));

     // find total number of students who took either elective1 or elective2 or both
// write LINQ code here









            Console.WriteLine($"Total Students: {allStudents}");

        }
        public SemesterResult(List<Student> students)
        {
            _students = students;
        }


        public (int id, string name) HighestMarks()
        {
            // return id and name as tuple of a student who scored
            // highest aggregate marks (InSem1Marks + InSem2Marks + FinalExamMarks)

       // write LINQ queries and the method code below
```

```
        }

        public List<Student> PercentDescending()
        {
// write LINQ query and method code

        }
    }
```

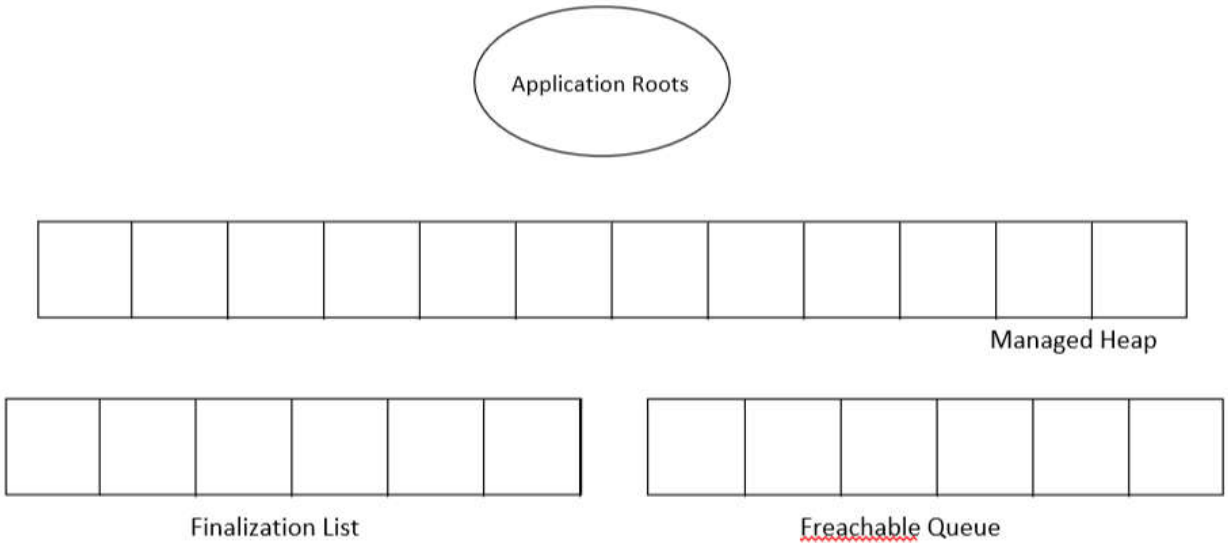
Q2. (10)

This question is about Garbage Collection mechanism of CLR as discussed in the lectures during the course. You need to complete the diagrams (fill up the objects names in various data structures) so that it reflects the state of CLR and the application accurately as far as GC is concerned.

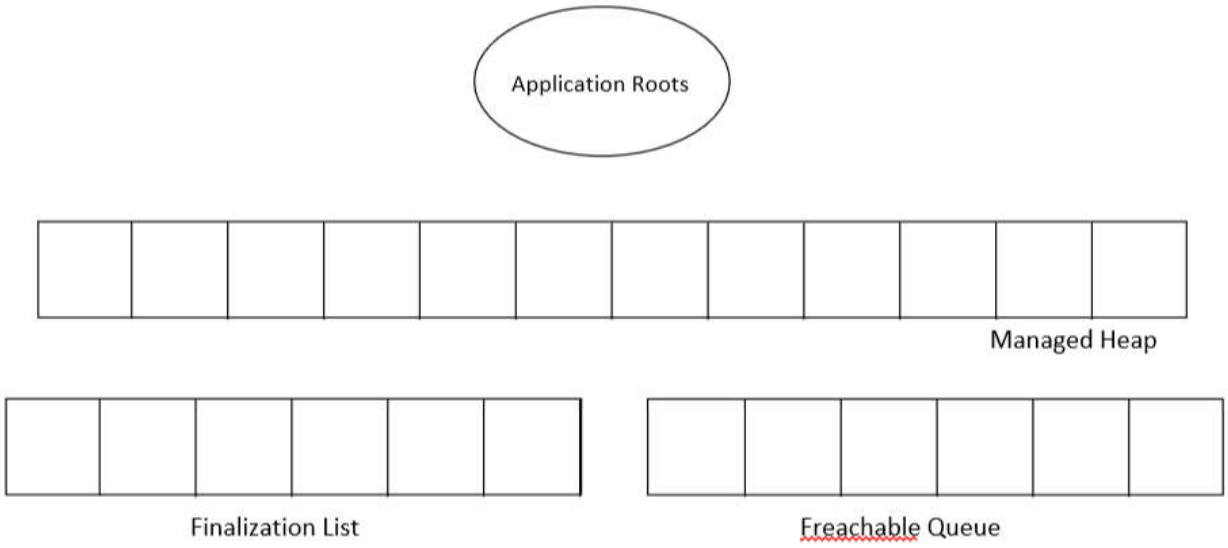
The application has objects named A, B, ..., K in the managed heap, as the GC mechanism executes through various states as mentioned below.

State 1: This is initial state just prior to point in time when GC runs.
Objects in Managed Heap: A, B, …, K. Objects with Roots: B, D, E, F, G
Objects that need finalization: A, C, F, H, K

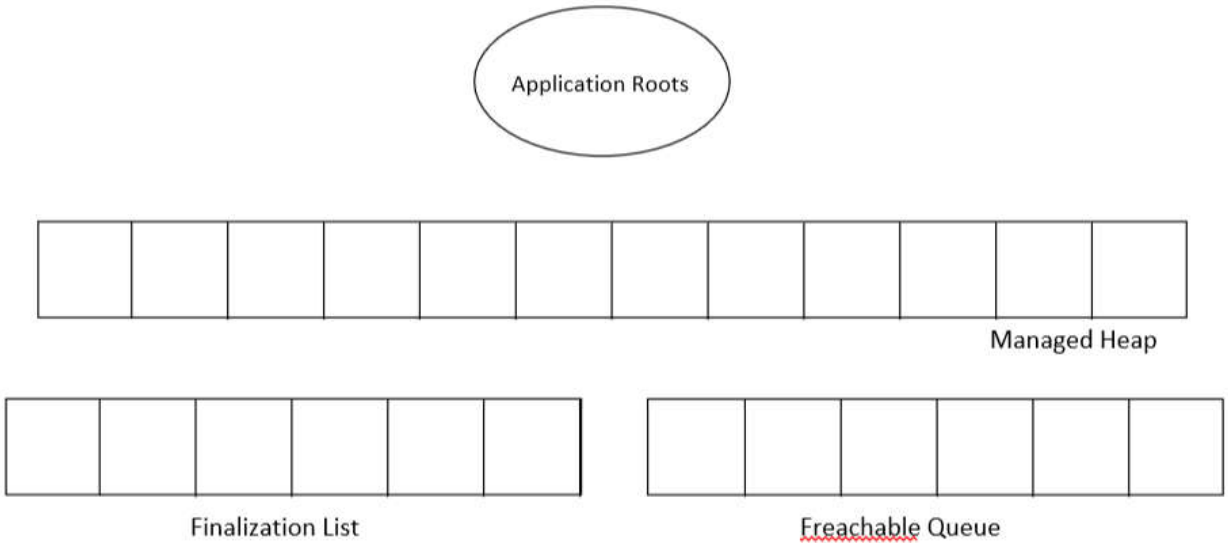
Depict this state in the diagram shown below (draw arrow lines to show app roots)



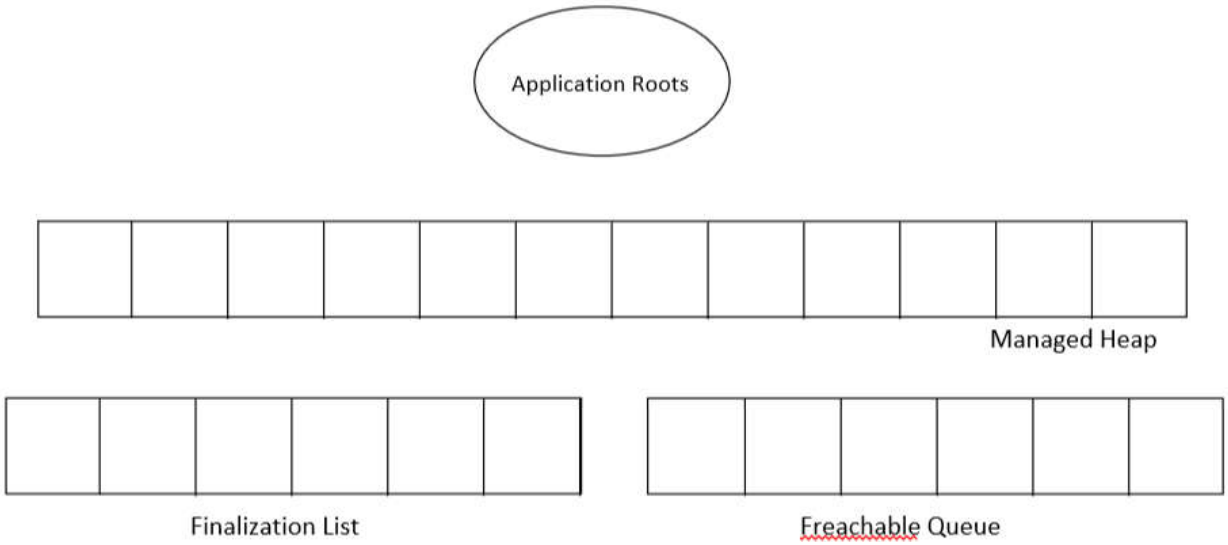
State 2: At this point in time, the GC has run (1st run). The thread responsible for Freachable has not run yet.



State 3: Thread that processes Freachable Queue had completed its responsibility for first 3 objects when it was interrupted by GC thread. Now, GC has run (2nd run) and Freachable Queue thread is blocked.



State 4: Freachable queue thread is unblocked and finishes its work completely. The object K's destructor (written in C#) resurrects this object and calls the following GC function: `ReRegisterForFinalize()`. Objects D and G have lost roots from the application. Now, the GC has run (3rd run).



Q3. (10)

Author a class `OSRes` that manages an OS resource outside of the managed heap using `IntPtr` type to point to the resource and implementing `IDisposable` interface. You can use comments to depict code that does some work (like, // close the handle) instead of writing code. Focus on the structure of the class and the key function calls/mechanisms.

/* your answer goes here */

Q4. (10)

Using knowledge about GC's internal workings and mechanism of Resurrection, design an object pool of expensive Game objects that a Server creates initially on start up. You can think of this server as an online gaming server that needs to quickly respond to game start requests and therefore can't afford to create time + memory expensive Game objects right then.
Here is the server class with depicted use of Game objects.

```
class Server {
  static void Main() {
    // Populate the pool with a bunch of Game objects.
    for (int i = 0; i < 1000; i++)
       new Game();

    // When you need an object, grab one out of the pool.
    Game g = Game.GetObjectFromPool();

    // The server can now use g.
    // :

    // To shut down the application cleanly, shut down the pool
    Game.ShutdownThePool();
  }
}

/* author the Game class here that manages an object pool */
```

Q5. (10)

Observe the code below. The class `AdvCalc` has a list of doubles and few functions that operate on this list. The code in `ProcessAsync()` function is to be made concurrent using `Task`, `async` and `await`. You should carefully identify the sequence dependencies between the actions. Eg. See that `SquarRoot()`depends on list to be first sorted because the `DisplayResults` function assumes `largeList` and `sqrts` lists are having numbers in correspondence. i.e. first number in `largeList` has its square root as first number in `sqrts` list, and so on.

Re-write `Main()` and `ProcessAsync()` functions making necessary changes to introduce maximum concurrency in `ProcessAsync()` using `Task`, `async` and `await`. You should not modify any other code.

```
public class AdvCalc
{
    List<double> largeList = new List<double>();

    public static void Main()
    {
        ProcessAsync();
    }

    public static void ProcessAsync()
    {
        AdvCalc advCalc = new AdvCalc();

        advCalc.PrepareLargeList();
        advCalc.SortInPlace();
        List<double> sqrts = advCalc.SquareRoot();
        double sum = advCalc.SumAll();

         Console.WriteLine("Doing calculations...");

        advCalc.DisplayResults(sqrts, sum);
    }

    private void DisplayResults(List<double> sqrts, double sum)
    {
        Console.WriteLine($"Sum = {sum}");
        for (int i = 0; i < largeList.Count; i++)
            Console.WriteLine($"Square root of {largeList[i]} is {sqrts[i]}");
    }

    private double SumAll()
    {
        return largeList.Sum();
    }

    private List<double> SquareRoot()
    {
        List<double> sq = new List<double>();
        largeList.ForEach(f => sq.Add(Math.Sqrt(f)));
```

```
            return sq;
        }

        private void SortInPlace()
        {
            largeList.Sort();
        }

        private void PrepareLargeList()
        {
            Random r = new Random();
            for (int i = 0; i < 100; i++)
                largeList.Add(r.NextDouble()*1000.0);
        }
    }

/*   Re-write Main() and ProcessAsync() functions here */
```