#1 Explain what is the difference between Skip() and SkipWhile() extension method?

Ans - The skip method skips the number of entries as mentioned in the argument. For eg. listName.Skip(3) will skip the first three entries i.e. the first three entries will not be shown in the output.
The SkipWhile() method will skip the entries till the condition mentioned is true. For eg. SkipWhile(c => c.Age < 10) will skip the entries till the age of those members is less than ten. When the condition becomes false after subsequently proceeding, all the remaining entries will be taken.

#2 Explain what is the difference between Statement Lambda and Expression Lambda?

#3 Explain how you can retrieve a single row with LINQ?

Ans- We can use FirstOrDefault(), LastOrDefault(), SingleOrDefault() for this depending upon our requirement.

#4 What is the difference between the FirstOrDefault() and First() selector methods in the LINQ?

Ans - The First method will query the first result from the inputted list that satisfies the condition mentioned in the First() method (it selects the first member of the list if no condition is mentioned). It gives an exception if no element is there in the list that satisfies the condition. The FirstOrDefault() method doesn't throw that exception. If there is no element present that satisfies that particular condition, it outputs the default value of that particular data type of which we apply this method (0 is default in case of integer, null in case of strings)

#5 What are SelectMany() and Select() in LINQ?

Ans- Select() selects elements or collection of elements from the list upon the condition we write.

#6 What are the Quantifier Operators?

Ans- Quantifier Operators return a boolean value based on the result of the condition mentioned inside the brackets. Examples: All, Any, Contains.

#7 What are the advantages of Deferred Execution?

Ans- Advantages of Deferred Execution are:-

- It gives the result of the latest execution on data.
- We can create custom methods using it.

#8 How will you count the elements of a collection or a list?

Ans- We can use .Count method for this.
Example - listName.Count() gives total number of elements in that list.

#9 How would you use LINQ to join two or more lists or tables together?

Ans- Join in LINQ is similar to inner join in SQL. Taking an example, we have two lists :- list of Candidates (candidateList) and list of CandidateDetails (candidateDetailsList). Candidate contains candidateID, candidateName, candidateAge, candidateSkill.
CandidateDetails contains candidateID and candidateAddress.
We want to apply join and display, candidateName and candidateAddress of common candidates

```
-> var join = candidateList.Join(
                        candidateDetailsList,
                        candidate => candidate.candidateID,
                        candidateDetails => candidateDetails.candidateID,
                        (candidate, candidateDetails) => new
                                    {
                                          Name = candidate.candidateName,
                                Address = candidateDetails.candidateAddress
                                    });
```

#10 What steps would you take to use LINQ to count the number of items in a collection that match a certain condition?

Ans- We can use count for this.
Example - listName.Count(element => condition on that element); gives count of number of elements that satisfy the condition

#11 How can you use LINQ to perform a Union or Intersect operation on two lists?

Ans-
Union:- var unionList = list1.Union(list2, new ListComparer());
var intersectList = list1.Intersect(list2, new ListComparer());
Where list1 and list are lists p=of items of same type and ListComparer class compares the entries in them.

In case of primitive data types List, we can simply do:-
var unionList = list1.Union(list2);
var intersectList = list1.Intersect(list2);

#12 Explain the difference between First, FirstOrDefault, Single and SingleOrDefault in LINQ and when you would use each.

Ans-
First -> This method fetches the first element that satisfies the condition mentioned in it. It returns an exception if the list is empty or if no element matches the condition. We use it to find the first element of the list or first element meeting the condition when we are sure that the list is not empty and one such element does exist.
FirstOrDefault -> This method also fetches the first element that satisfies the condition mentioned in it and if such element is not found it returns the default value of element of that particular data type (0 when integer type list, null when string type list). We use it to find the first element of the list satisfying the condition.
Single -> This method fetches the element that satisfies the condition mentioned in it. It gives an exception if there are more than one element satisfying the given condition and also if no such elements are found. We use it to find the only element meeting the condition.
SingleOrDefault -> This method also fetches the element that satisfies the condition mentioned in it. If there are more than one elements meeting the condition, it also gives an exception.

#13 How would you use LINQ to check if a collection contains a specific item?

Ans- We can do this by using where.
```
var result = listName.where(c => c.item = “required item”.
```

#14 How do you use LINQ to retrieve distinct items from a collection and when would you want to do this?

Ans- Using the Distinct method.
```
var result = listName.Distinct()
```
We do this when we need distinct elements from a list.

#15 Explain the difference between Any and All in LINQ and when you would use each.

Ans- Both any and all returns boolean type
listName.All gives true if all of the elements in the list satisfies the specified condition.
We use it to check if all elements meet that given condition.
listName.Any gives true if any one of the elements in the list satisfies the specified condition.
We use it to check if any one of the elements meet that given condition.

#16 what will be the output:

```
                IList<Student> studentList = new List<Student>() {
    new Student() { StudentID = 1, StudentName = "John", Age = 18,
StandardID = 1 } ,
    new Student() { StudentID = 2, StudentName = "Steve",  Age = 21,
StandardID = 1 } ,
    new Student() { StudentID = 3, StudentName = "Bill",  Age = 18,
StandardID = 2 } ,
    new Student() { StudentID = 4, StudentName = "Ram" , Age = 20,
StandardID = 2 } ,
    new Student() { StudentID = 5, StudentName = "Ron" , Age = 21 }
      };

      IList<Standard> standardList = new List<Standard>() {
            new Standard(){ StandardID = 1, StandardName="Standard 1"},
            new Standard(){ StandardID = 2, StandardName="Standard 2"},
            new Standard(){ StandardID = 3, StandardName="Standard 3"}
      };


      a.) var studentNames = studentList.Where(s => s.Age > 18)
                              .Select(s => s)
                              .Where(st => st.StandardID > 0)
                              .Select(s => s.StudentName);

      b.) var teenStudentsName = from s in studentList
                       where s.age > 12 && s.age < 20
                       select new { StudentName = s.StudentName };
```

```
            teenStudentsName.ToList().ForEach(s =>
Console.WriteLine(s.StudentName));


      c.) var studentsGroupByStandard = from s in studentList
                              group s by s.StandardID into sg
                              orderby sg.Key
                                    select new { sg.Key, sg };


                  foreach (var group in studentsGroupByStandard)
                  {
                        Console.WriteLine("StandardID {0}:", group.Key);

                        group.sg.ToList().ForEach(st =>
Console.WriteLine(st.StudentName ));
                  }


      d.) IList<int> intList = new List<int>() { 7, 10, 21, 30, 45, 50, 87
};
                  IList<string> strList = new List<string>() { null, "Two",
"Three", "Four", "Five" };

                  Console.WriteLine("1st Element which is greater than 250
in intList: {0}",

intList.First( i > 250));

                  Console.WriteLine("1st Even Element in intList: {0}",

strList.FirstOrDefault(s => s.Contains("T")));


      e.) IList<string> strList1 = new List<string>() { "Two", "Three",
"Four", "Five" };
                  IList<string> strList2 = new List<string>() { null,
"Two", "Three", "Four", "Five" };

                  Console.WriteLine(strList1.LastOrDefault(s =>
s.Contains("T")));
                  Console.WriteLine(strList2.LastOrDefault(s =>
s.Contains("T")));

      f.) IList<string> strList1 = new List<string>() { "Two", "Three",
"Four", "Five" };
```

```
                IList<string> strList2 = new List<string>() { null,
"Two", "Three", "Four", "Five" };
                IList<string> strList3 = new List<string>();

                Console.WriteLine(strList1.Single(s => s.Contains("T")));
                Console.WriteLine(strList2.SingleOrDefault(s =>
s.Contains("T")));
                Console.WriteLine(strList3.SingleOrDefault(s =>
s.Contains("T")));
```

Ans-

a) Steve
   Ram

b) John
   Bill

c) StandardID 0:
   Ron
   StandardID 1:
   John
   Steve
   StandardID 2:
   Bill
   Ram

d) Run-Time Exception

e) Run-Time Exception (Here it will be null reference exception)

f) Run-Time Exception