Query Operators Hands –on Lab

Task 1: Open VS2010 and create a blank solution. Give 'LINQ_ApplicationDevelopment' name to it.

Task 2: Add a class library project in this solution. Name it as 'CS_DataLibrary'. This project will contains classes on which you will be applying query operators. Add following four classes in this project:

```
public class Person
    {
        public int PersonId { get; set; }
        public string PersonName { get; set; }
        public int Age { get; set; }
        public string City { get; set; }
        public string Occupation { get; set; }
        public int Income { get; set; }
    }

    public class PersonCollection : List<Person>
    {
        public PersonCollection()
        {
            Add(new Person() { PersonId = 101, PersonName = "Kishan",
Age = 30, City= "Pune",  Occupation = "Service", Income = 56000 });
            Add(new Person() { PersonId = 102, PersonName = "Vasudev",
Age = 32, City = "Nagpur", Occupation = "Self Employeed", Income =
100000 });
            Add(new Person() { PersonId = 103, PersonName = "Ram", Age
= 37, City = "Mumbai", Occupation = "Service", Income = 56000 });
            Add(new Person() { PersonId = 104, PersonName = "Gopal",
Age = 42, City = "Kolhapur", Occupation = "Self Employeed", Income =
86000 });
            Add(new Person() { PersonId = 105, PersonName = "Madhav",
Age = 41, City = "Kolhapur", Occupation = "Self Employeed", Income =
96000 });
            Add(new Person() { PersonId = 106, PersonName = "Mohan",
Age = 53, City = "Jalgaon", Occupation = "Service", Income = 36000 });
            Add(new Person() { PersonId = 107, PersonName = "Kirshna",
Age = 23, City = "Pune", Occupation = "Service", Income = 66000 });
            Add(new Person() { PersonId = 108, PersonName =
"ParthSarathi", Age = 53, City = "Nagpur", Occupation = "Service",
Income = 76000 });
            Add(new Person() { PersonId = 109, PersonName = "Mukund",
Age = 54, City = "Jalgaon", Occupation = "Self Employeed", Income =
96000 });
            Add(new Person() { PersonId = 1010, PersonName = "Keshav",
Age = 23, City = "Pune", Occupation = "Service", Income = 23000 });
```

```
        }
    }

    public class Employee
    {
        public int EmpNo { get; set; }
        public string EmpName { get; set; }
        public string DeptName { get; set; }
        public int Salary { get; set; }
    }

    public class EmployeeCollection : List<Employee>
    {
        public EmployeeCollection()
        {
            Add(new Employee() { EmpNo = 1001, EmpName =
"YashodaNandan", DeptName = "IT", Salary = 53000 });
            Add(new Employee() { EmpNo = 1002, EmpName =
"DevkiNandan", DeptName = "CTD", Salary = 33000 });
            Add(new Employee() { EmpNo = 1003, EmpName = "RadheShyam",
DeptName = "SYS", Salary = 63000 });
            Add(new Employee() { EmpNo = 1004, EmpName = "Gopal",
DeptName = "HRD", Salary = 13000 });
            Add(new Employee() { EmpNo = 1005, EmpName = "Govind",
DeptName = "SYS", Salary = 93000 });
            Add(new Employee() { EmpNo = 1006, EmpName = "Mohan",
DeptName = "CTD", Salary = 63000 });
            Add(new Employee() { EmpNo = 1007, EmpName = "Madhav",
DeptName = "IT", Salary = 23000 });
            Add(new Employee() { EmpNo = 1008, EmpName = "Milind",
DeptName = "ACCTS", Salary = 53000 });
            Add(new Employee() { EmpNo = 1009, EmpName = "Vasudev",
DeptName = "ACCTS", Salary = 63000 });
            Add(new Employee() { EmpNo = 1010, EmpName = "Shridhar",
DeptName = "IT", Salary = 83000 });
        }
    }
```

Task 3: In the same solution, add a new console application project, name it as 'CS_SimpleLINQ'. In this project add a reference of the data library created in the 'Task 2'. In this project we will understand Lambda Expression and Extension method.

Task 4: Write the following code in the Main Method:

```
static void Main(string[] args)
        {

            var PersonList = new PersonCollection();

            //Defining the Function Pointer for Lambda Expression

            Func<Person, bool> valueFilter = delegate(Person p) {
return p.PersonName.StartsWith("M"); };


            var oQueryFunctionPointer = PersonList.Where(valueFilter)
                                    .Select(p => new Person()
                                    {
                                        PersonId = p.PersonId,
                                        PersonName = p.PersonName,
                                        Age = p.Age,
                                        City = p.City,
                                        Occupation = p.Occupation,
                                        Income = p.Income
                                    }
                                            );



            Console.WriteLine("Result using Query Methods");
            Console.WriteLine();

Console.WriteLine("PersonId\tPersonName\tAge\tCity\t\tOccupation\tInco
me");

            foreach (var item in oQueryFunctionPointer)
            {
                Console.WriteLine(item.PersonId + "\t\t" +
item.PersonName + "\t\t" + item.Age + "\t" + item.City + "\t" +
item.Occupation + "\t" + item.Income);
            }
            Console.WriteLine();
            Console.WriteLine();

            //Simple LINQ to Object Code using Query Mathods and
Lambda Expression

            var oQueryResult = PersonList.Where(p =>
```

```
p.PersonName.StartsWith("M"))
                                    .Select(p => new Person()
                                                 {

PersonId=p.PersonId,

PersonName=p.PersonName,
                                                     Age=p.Age,
                                                     City=p.City,

Occupation=p.Occupation,
                                                     Income=p.Income
                                                 }
                                          );


            Console.WriteLine("Result using Query Methods");
            Console.WriteLine();

Console.WriteLine("PersonId\tPersonName\tAge\tCity\t\tOccupation\tInco
me");

            foreach (var item in oQueryResult)
            {
                Console.WriteLine(item.PersonId + "\t\t" +
item.PersonName + "\t\t" +
item.Age+"\t"+item.City+"\t"+item.Occupation+"\t"+item.Income);
            }

            Console.WriteLine();
            Console.WriteLine();
            Console.WriteLine("Using LINQ to Objects (OLINQ)");
            Console.WriteLine();
            Console.WriteLine();

            var QueryOLinq = from p in PersonList
                             where p.PersonName.StartsWith("M")
                             select new Person()
                              {
                                  PersonId = p.PersonId,
                                  PersonName = p.PersonName,
                                  Age = p.Age,
                                  City = p.City,
                                  Occupation = p.Occupation,
                                  Income = p.Income
```

```
                                };

            Console.WriteLine("Result using Query OLINQ");
            Console.WriteLine();

Console.WriteLine("PersonId\tPersonName\tAge\tCity\t\tOccupation\tInco
me");

            foreach (var item in QueryOLinq)
            {
                Console.WriteLine(item.PersonId + "\t\t" +
item.PersonName + "\t\t" + item.Age + "\t" + item.City + "\t" +
item.Occupation + "\t" + item.Income);
            }

            Console.ReadLine();
        }
```

Run the project and test the output.

Using Joins

In this exercise we will understand various joins.

Task 1: In the same solution add a new Console project. Name it as 'CS_Joinns'.

Task 2: Write the following code in it:

```
    public class Person
    {
        int _id;
        int _idRole;
        string _lastName;
        string _firstName;
        public int ID
        {
            get { return _id; }
            set { _id = value; }
        }
        public int IDRole
        {
            get { return _idRole; }
            set { _idRole = value; }
        }
        public string LastName
        {
            get { return _lastName; }
```

```
            set { _lastName = value; }
        }
        public string FirstName
        {
            get { return _firstName; }
            set { _firstName = value; }
        }
    }


    public class Role
    {
        int _id;
        string _roleDescription;
        public int ID
        {
            get { return _id; }
            set { _id = value; }
        }
        public string RoleDescription
        {
            get { return _roleDescription; }
            set { _roleDescription = value; }
        }
    }


    public class Salary
    {
        int _idPerson;
        int _year;
        double _salary;

        public int IDPerson
        {
            get { return _idPerson; }
            set { _idPerson = value; }
        }
        public int Year
        {
            get { return _year; }
            set { _year = value; }
        }
        public double SalaryYear
        {
            get { return _salary; }
            set { _salary = value; }
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
```

```
            #region Data Storage
            List<Person> lstPerson = new List<Person> {
                                                        new Person
                                                        {
                                                            ID = 1,
                                                            IDRole = 1,
                                                            LastName = "Sabnis",
                                                            FirstName = "Mahesh"
                                                        },
                                                        new Person
                                                        {
                                                            ID = 2,
                                                            IDRole = 2,
                                                            LastName = "Pandit",
                                                            FirstName = "Ajay"
                                                        },
                                                        new Person
                                                        {
                                                            ID = 3,
                                                            IDRole = 2,
                                                            LastName = "Patil",
                                                            FirstName = "Sanjay"
                                                        },
                                                        new Person
                                                        {
                                                            ID = 4,
                                                            IDRole = 3,
                                                            LastName = "Puranik",
                                                            FirstName = "Rajesh"
                                                        },
                                                        new Person
                                                        {
                                                             ID=5,
                                                             IDRole=2,
                                                             LastName="Deshpande",
                                                             FirstName="Amol"
                                                        }
                                                    };
            List<Role> lstRoles = new List<Role>
            {
                new Role { ID = 1, RoleDescription = "Manager" },
                new Role { ID = 2, RoleDescription = "Developer" }
            };
            #endregion


            #region Join This is Like Inner Join, Combines two Sequences based upon
Matching Key
            var joinQuery = from per in lstPerson
                            join role in lstRoles on per.IDRole equals role.ID
                            select new
                            {
                                FirstName = per.FirstName,
                                LastName = per.LastName,
                                Responsibility = role.RoleDescription
```

```
                            };


            Console.WriteLine("The Result of Join (Like Inner Join of Sql)");
            Console.WriteLine("First Name \t\t Last Name\t\t Responsibility");
            foreach (var item in joinQuery)
            {
                Console.WriteLine(item.FirstName + "\t\t\t\t" + item.LastName +
"\t\t\t" + item.Responsibility);
            }
            Console.WriteLine("Join Result Ends Here");

            #endregion


            #region GroupJoin, This creates a new Sequence, This is same like Left
Outer Join of Sql

            var groupJoinQuery = from per in lstPerson
                                 join role in lstRoles on per.IDRole equals role.ID
into PerRole
                                 from perrole in PerRole.DefaultIfEmpty()
                                 select new
                                     {
                                         FirstName = per.FirstName,
                                         LastName = per.LastName,
                                         RoleDesc = perrole == null ? "No Role" :
perrole.RoleDescription
                                     };

            Console.WriteLine("The Result of the Group Join (Same as Left Outer Join
of Sql)");
            Console.WriteLine();
            Console.WriteLine("FirstName\t\tLastName\t\tResponsibility");
            Console.WriteLine();
            foreach (var item in groupJoinQuery)
            {
                Console.WriteLine(item.FirstName + "\t\t\t" +
item.LastName+"\t\t\t"+item.RoleDesc);
            }

            #endregion

            Console.Clear();
            #region GruoyBy Query with Standard Type
            Console.WriteLine();
            Console.WriteLine();
            Console.WriteLine("GroupBy on .NET STandard Type");
            var groupByStdType = from typ in typeof(string).GetMethods()
                                 group typ by typ.Name into methodGb
                                 select new { Name = methodGb.Key,
OverloadedMethods=methodGb.Count()};

            foreach (var item in groupByStdType)
            {
```

```
                Console.WriteLine("Name :" + item.Name  + "\tNo. Of Overloades :" +
item.OverloadedMethods);
            }
            Console.WriteLine();
            Console.WriteLine("Ends Here");
            Console.WriteLine();

            Console.WriteLine("GroupBy on Custom Data (Employee Class)");
            Console.WriteLine();
            EmployeeCollection empColl = new EmployeeCollection();
            var deptWiseSumSalary = from emp in empColl
                                    group emp by emp.DeptName into groupDept
                                    select new
                                        {
                                             DeptName = groupDept.Key,
                                             SumSalary = groupDept.Sum(s=>s.Salary)
                                        };
            Console.WriteLine("Department wise Maximum Salary");
            Console.WriteLine("DeptName \t\t\tSalary" );
            Console.WriteLine();
            foreach (var item in deptWiseSumSalary)
            {
                Console.WriteLine(item.DeptName + "\t\t\t\t" + item.SumSalary);
            }

            Console.WriteLine();
            Console.WriteLine("Ends Here");
            Console.WriteLine();
            #endregion

            Console.ReadLine();
        }
    }
```

Run the Application and test the result.

Using Ordering Operators.

In this exercise we will understand how to make use of Ordering operators like OredrBy, OrderByDescending, ThenBy and ThenByDescending. Add the reference of Data library which we have created in the beginning.

Task 1: In this Solution add a new Console Application, name it as 'CS_OrderingOperators'.

Task 2: Write the following code in it:

```
using CS_DataLibrary;

namespace CS_OrderingOperators
```

```
{
    /// <summary>
    /// This APplication will Explain, OrderBy, OrderByDescending,
ThenBy,ThenByDescending
    /// OrderBy and OrderByDescending: It works on Onle one Ordering
Key
    /// ThenBy and ThenByDescending: Concantinates multiple Ordering
Key
    /// </summary>
    class Program
    {
        static void Main(string[] args)
        {

            OrderBy_CLS_Type();

            OrderBy_Employee();

            OrderBy_CLS_Type_Descending();

            OrderBy_Employee_Descending();

            OrderBy_ThenBy_Employees();

            Console.ReadLine();
        }


        static void OrderBy_CLS_Type()
        {
            var orderByCLS = from typ in typeof(string).GetMethods()
                             orderby typ.Name
                             group typ by typ.Name into groupType
                             select new { MethodName= groupType.Key};

            Console.WriteLine("OrderBy Standard CLS Type");
            Console.WriteLine();
            foreach (var item in orderByCLS)
            {
                Console.WriteLine("Name :" + item.MethodName);
            }

            Console.WriteLine("Ends Here");
            Console.WriteLine();
        }
```

```
        static void OrderBy_CLS_Type_Descending()
        {
            var orderByCLS = from typ in typeof(string).GetMethods()
                             orderby typ.Name descending
                             group typ by typ.Name into groupType
                             select new { MethodName = groupType.Key
};

            Console.WriteLine("OrderBy Standard CLS Type");
            Console.WriteLine();
            foreach (var item in orderByCLS)
            {
                Console.WriteLine("Name :" + item.MethodName);
            }

            Console.WriteLine("Ends Here");
            Console.WriteLine();
        }


        static void OrderBy_Employee()
        {
            EmployeeCollection empCol = new EmployeeCollection();

            Console.WriteLine("Displaying All Employees Order By
Name");
            Console.WriteLine();

Console.WriteLine("EmpNo\t\tEmpName\t\t\tDeptName\t\t\tSalary");
            var orderByEmployee = from emp in empCol
                                  orderby emp.EmpName
                                  select emp;
            Console.WriteLine();
            foreach (var item in orderByEmployee)
            {
                Console.WriteLine(item.EmpNo + "\t\t" + item.EmpName +
"\t\t\t" + item.DeptName + "\t\t\t" + item.Salary);
            }
            Console.WriteLine();
            Console.WriteLine("Ends Here");
            Console.WriteLine();
        }

        static void OrderBy_Employee_Descending()
```

```
        {
            EmployeeCollection empCol = new EmployeeCollection();

            Console.WriteLine("Displaying All Employees Order By
Name");
            Console.WriteLine();

Console.WriteLine("EmpNo\t\tEmpName\t\t\tDeptName\t\t\tSalary");
            var orderByEmployee = from emp in empCol
                                  orderby emp.EmpName descending
                                  select emp;
            Console.WriteLine();
            foreach (var item in orderByEmployee)
            {
                Console.WriteLine(item.EmpNo + "\t\t" + item.EmpName +
"\t\t\t" + item.DeptName + "\t\t\t" + item.Salary);
            }
            Console.WriteLine();
            Console.WriteLine("Ends Here");
            Console.WriteLine();
        }

        /// <summary>
        /// This Method Explains OrderBy and ThenBy Operations.
        /// It Makes use of 1->LINQ and 2->Extension Methods for
Demonstration
        /// </summary>
        static void OrderBy_ThenBy_Employees()
        {
            EmployeeCollection empCol = new EmployeeCollection();
            Console.WriteLine("OrderBy and ThenBy using Extension
Methods");
            Console.WriteLine();

            //var orderBy_thenBy_Extension = empCol.OrderBy(Emp =>
Emp.EmpName)
            //                             .ThenBy(Emp => Emp.Salary);

            var orderBy_thenBy_Extension =
empCol.OrderByDescending(Emp => Emp.EmpName)
                                        .ThenByDescending(Emp =>
Emp.Salary);


Console.WriteLine("EmpNo\t\tEmpName\t\t\tDeptName\t\t\tSalary");
```

```
            foreach (var item in orderBy_thenBy_Extension)
            {
                Console.WriteLine(item.EmpNo + "\t\t" + item.EmpName +
"\t\t\t" + item.DeptName + "\t\t\t" + item.Salary);
            }

            Console.WriteLine();
            Console.WriteLine("Ends Here");

            Console.WriteLine("OrderBy and ThenBy using LINQ");
            Console.WriteLine();

            var orderBy_thenBy_LINQ = from Emp in empCol
                                      orderby Emp.EmpName, Emp.Salary
descending
                                      select Emp;


Console.WriteLine("EmpNo\t\tEmpName\t\t\tDeptName\t\t\tSalary");
            foreach (var item in orderBy_thenBy_LINQ)
            {
                Console.WriteLine(item.EmpNo + "\t\t" + item.EmpName +
"\t\t\t" + item.DeptName + "\t\t\t" + item.Salary);
            }
            Console.WriteLine("Ends Here");
            Console.WriteLine();
        }


    }
}
```

Run the above application.

Using Element Operators.

In this application we will understand Element Operators like First, FirstOrDefault, Last, LastOrDefault, etc.

Task 1: In the Same solution add a new console application; name it as 'CS_ElementOperators'. Add the reference of the data library which we have created in the beginning.

```
using CS_DataLibrary;
```

```
namespace CS_ElementOperators
{

    /// <summary>
    /// This Application explains
First,FirstOrDefault,Las,LastOrDefault,Single,SingleOrDefault,ElementA
t,
    /// ElementAtOrDefault and DefaultIfEmpty
    /// </summary>
    class Program
    {
        static EmployeeCollection EmpCol = new EmployeeCollection();

        static void Main(string[] args)
        {
            //Method_First_Last();

            //Method_FirstOrDefault_LastOrDefault();

           // Method_Single_SingleOrDefault();

            Method_ElementAt_ElementAtDefault();

            Console.ReadLine();
        }

        /// <summary>
        /// This Method will return First and Last Record from the
Employee Collection
        /// </summary>
        static void Method_First_Last()
        {
            Console.WriteLine("The First Record in Employee
Collection");
            Console.WriteLine();

            var FirstEmployee = EmpCol.First();

            Console.WriteLine("EmpNo   :" + FirstEmployee.EmpNo + " "
+
                              "EmpName :" + FirstEmployee.EmpName + "
" +
                              "DeptName:" + FirstEmployee.DeptName + "
" +
                              "Salary  :" + FirstEmployee.Salary);
```

```
            Console.WriteLine();
            Console.WriteLine("Ends Here");

            Console.WriteLine("The First Record in Employee
Collection");
            Console.WriteLine();

            var LastEmployee = EmpCol.Last();

            Console.WriteLine("EmpNo   :" + LastEmployee.EmpNo + " " +
                              "EmpName :" + LastEmployee.EmpName + " "
+
                              "DeptName:" + LastEmployee.DeptName + "
" +
                              "Salary  :" + LastEmployee.Salary);

            Console.WriteLine();
            Console.WriteLine("Ends Here");

            Console.WriteLine();
        }



        /// <summary>
        /// This Method will return FirstOrDefault and LastOrDefault
Record from the Employee Collection
        /// </summary>
        static void Method_FirstOrDefault_LastOrDefault()
        {
            Console.WriteLine("The FirstOrDefault Record in Employee
Collection");
            Console.WriteLine();

            var FirstEmployee =
EmpCol.FirstOrDefault(Emp=>Emp.Salary>=90000);

            Console.WriteLine("EmpNo   :" + FirstEmployee.EmpNo + " "
+
                              "EmpName :" + FirstEmployee.EmpName + "
" +
                              "DeptName:" + FirstEmployee.DeptName + "
" +
                              "Salary  :" + FirstEmployee.Salary);
```

```
            Console.WriteLine();
            Console.WriteLine("Ends Here");

            Console.WriteLine("The First Record in Employee
Collection");
            Console.WriteLine();

            var LastEmployee =
EmpCol.LastOrDefault(Emp=>Emp.Salary>=400000);

            Console.WriteLine("EmpNo   :" + LastEmployee.EmpNo + " " +
                              "EmpName :" + LastEmployee.EmpName + " "
+
                              "DeptName:" + LastEmployee.DeptName + "
" +
                              "Salary  :" + LastEmployee.Salary);

            Console.WriteLine();
            Console.WriteLine("Ends Here");

            Console.WriteLine();
        }

        /// <summary>
        /// This Method will Explain Single and SingleOrDefault Record
from the   Collection
        /// </summary>
        static void Method_Single_SingleOrDefault()
        {
            int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
            var query1 = numbers.Single(n => n > 8);
            Console.WriteLine("Single = " + query1);


            var query2 = numbers.SingleOrDefault(n => n > 9);
            Console.WriteLine("SingleOrDefault" + query2);
        }

        /// <summary>
        /// This Method Returns Element from the sequence based upon
the Zero Based Index.
        /// </summary>
        static void Method_ElementAt_ElementAtDefault()
        {
```

```
            Console.WriteLine("ElementAt Specific Index");
            Console.WriteLine();

            var EmpAtIndex = EmpCol.ElementAt(3);

            Console.WriteLine("EmpNo   :" + EmpAtIndex.EmpNo + " " +
                                            "EmpName :" +
EmpAtIndex.EmpName + " " +
                                            "DeptName:" +
EmpAtIndex.DeptName + " " +
                                            "Salary  :" +
EmpAtIndex.Salary);



            Console.WriteLine();
            Console.WriteLine("Ends Here");
            Console.WriteLine();

            Console.WriteLine("ElementAtOrDefault Specific Index");
            Console.WriteLine();

            var EmpAtOrDefaultIndex = EmpCol.ElementAtOrDefault(0);

            Console.WriteLine("EmpNo   :" + EmpAtOrDefaultIndex.EmpNo
+ " " +
                                            "EmpName :" +
EmpAtOrDefaultIndex.EmpName + " " +
                                            "DeptName:" +
EmpAtOrDefaultIndex.DeptName + " " +
                                            "Salary  :" +
EmpAtOrDefaultIndex.Salary);



            Console.WriteLine();
            Console.WriteLine("Ends Here");

            int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
            var query = numbers.ElementAtOrDefault(9);
            Console.WriteLine(query);
        }
    }
}
```

Run the application.

Using Set Operators

In this exercise we will understand Set Operators like, Distinct , Intersect, Union etc.

Task 1: In the same solution add a new console application, name it as 'CS_SetOperators'. In this project add reference of the data library created in the first task.

Task 2: Write the following code in it:

```
using CS_DataLibrary;

namespace CS_SetOperators
{
    /// <summary>
    /// This Application Explains Set Operators like, Distinct,
Intersect, Union and Except
    /// Distinct:Eliminates duplicates from the sequence
    /// Intersect : returns a sequence made by common elements of two
different sequences
    /// Union : This operator returns a new sequence formed by uniting
the two different sequences.
    /// Except : This operator produces a new sequence composed of the
elements of the first sequence not present in the second sequence.
    /// </summary>

    class Program
    {
        static PersonCollection PerCol = new PersonCollection();
        static EmployeeCollection EmpCol = new EmployeeCollection();

        static void Main(string[] args)
        {
            Method_Distinct_CityNames();

            Method_Intersect();

            Method_Union();

            Method_Except();

            Console.ReadLine();
        }

        /// <summary>
```

```
        /// This Method will return the locations from where Person
are available
        /// </summary>
        static void Method_Distinct_CityNames()
        {
            Console.WriteLine("Select Person Cities");
            Console.WriteLine();

            var distinctCities = (from Per in PerCol
                                  select Per.City).Distinct() ;

            foreach (var item in distinctCities)
            {
                Console.WriteLine(item);
            }

            Console.WriteLine();
            Console.WriteLine("Ends Here");
            Console.WriteLine();
        }

        static void Method_Intersect()
        {
            Console.WriteLine("Intersect");
            int[] Seq1 = { 1, 1, 2, 3, 3 };
            int[] Seq2 = { 1, 3, 3, 4 };

            Seq1.Intersect(Seq2);

            foreach (var item in Seq1.Intersect(Seq2))
            {
                Console.WriteLine(item);
            }


            Console.WriteLine();
            Console.WriteLine("Ends Here");

        }

        static void Method_Union()
        {
            Console.WriteLine("Union");

            int[] Seq1 = { 1, 1, 3, 3 };
```

```
            int[] Seq2 = { 1, 2, 3, 4 };

              Seq1.Union(Seq2);

              foreach (var item in Seq1.Union(Seq2))
            {
                Console.WriteLine(item);
            }

            Console.WriteLine("Ends Here");

        }

        static void Method_Except()
        {
            Console.WriteLine("Except");

            int[] Seq1 = { 1, 2, 3, 4 };
            int[] Seq2 = { 1, 1, 3, 3 };

            Seq1.Except(Seq2);

            foreach (var item in Seq1.Except(Seq2) )
            {
                Console.WriteLine(item);
            }
            Console.WriteLine("Ends Here");
        }
    }
}
```

Using Quntifying Operators

In this exercise we will understand Quantifying operators like, All, Any and Contains etc.

Task 1: In the same solution add a new console application, name it as 'CS_QuantifierOperators'. In the same project add a reference of the data library added in the beginning.

Task 2: Write the following code:

```
using CS_DataLibrary;

namespace CS_QuantifierOperators
{
```

```
    /// <summary>
    /// This Application will Explain Quantifier Operators like, All,
Any and COntains
    /// All :This operator uses the predicate function against the
elements of a
    /// sequence and returns true if all of them satisfy the predicate
condition.
    /// Any: This operator searches a sequence for elements that
satisfy the specified condition.
    /// Contains: This operator looks for a specified type within the
sequence and returns
    /// true when the element is found.
    /// </summary>
    class Program
    {
        static EmployeeCollection EmpCol = new EmployeeCollection();

        static void Main(string[] args)
        {

            Console.WriteLine("All Operator");
            Console.WriteLine();

            Console.WriteLine("Checks Wheather All Employees having
Salary More than 10000");
            var AllEmp = EmpCol.All(Emp => Emp.Salary >= 10000);

            Console.WriteLine(AllEmp);

            Console.WriteLine();
            Console.WriteLine("Ends Here");
            Console.WriteLine();

            Console.WriteLine("Any Operator");
            Console.WriteLine();
            Console.WriteLine("Checks Wheather Any Employees having
Salary Less than 10000");
            var AnyEmp = EmpCol.Any(Emp => Emp.Salary <= 10000);

            Console.WriteLine(AnyEmp);

            Console.WriteLine();
            Console.WriteLine("Ends Here");

            Console.WriteLine();
```

```
            Console.WriteLine("Contains Operator");
            Console.WriteLine();
            Console.WriteLine("Checks Wheather This Employee is
Available");
            Employee EmpToSearch = EmpCol[0];

            var ContainEmp = EmpCol.Contains(EmpToSearch);

            Console.WriteLine(ContainEmp);

            Console.WriteLine();
            Console.WriteLine("Ends Here");
            Console.ReadLine();
        }
    }
}
```

Using Partioning Operators

In this exercise we will understand Partioning Operators like Take, Skip **etc.**

Task 1: In the same solution add a new Console project, name it as 'CS_PartioningOperators'. In this project add reference of the data library we have created in the beginning.

Task 2: Write the following code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;


using CS_DataLibrary;
namespace CS_PartioningOperators
{
    /// <summary>
    /// This Application Explaing Partitioning Operators like Take,
Skip,TakeWhile and SkipWhile
    /// </summary>
    class Program
    {
        static EmployeeCollection EmpCol = new EmployeeCollection();
        static void Main(string[] args)
        {
```

```
            Method_Take();

            Method_Skip();

            Method_TakeWhile_SkipWhile();


            Console.ReadLine();
        }

        /// <summary>
        /// Take: Operators Extracts only specific Number of
Parameters.
        /// This is Generally used when you want to implement
Pagination on the Source
        /// </summary>
        static void Method_Take()
        {
            Console.WriteLine("Take Operator");
            Console.WriteLine("Take only first 3 Employee Records from
the Employee Collection");
            Console.WriteLine();

            var takeFirst_Three = (from Emp in EmpCol
                                   select Emp).Take(3);


Console.WriteLine("EmpNo\t\tEmpName\t\t\tDeptName\tSalary");
            foreach (var item in takeFirst_Three)
            {

Console.WriteLine(item.EmpNo+"\t\t"+item.EmpName+"\t\t"+item.DeptName+
"\t\t"+item.Salary);
            }

            Console.WriteLine();
            Console.WriteLine("Ends Here");
            Console.WriteLine();
        }

        /// <summary>
        /// Skip: Operators Skip paramaters specified form extract
rest of records from Sequence.
        /// This is Generally used when you want to implement
Pagination on the Source
```

```
        /// </summary>
        static void Method_Skip()
        {
            Console.WriteLine("Skip Operator");
            Console.WriteLine("Skip the first 3 Employee Records from
the Employee Collection");
            Console.WriteLine();

            var skipFirst_Three = (from Emp in EmpCol
                                   select Emp).Skip(3);


Console.WriteLine("EmpNo\t\tEmpName\t\t\tDeptName\tSalary");
            foreach (var item in skipFirst_Three)
            {
                Console.WriteLine(item.EmpNo + "\t\t" + item.EmpName +
"\t\t" + item.DeptName + "\t\t" + item.Salary);
            }

            Console.WriteLine();
            Console.WriteLine("Ends Here");
            Console.WriteLine();
        }

        /// <summary>
        ///
        /// </summary>
        static void Method_TakeWhile_SkipWhile()
        {
            int[] numbers = { 9, 3, 5, 4, 2, 6, 7, 1, 8 };
            var query1 = numbers.TakeWhile((n, index) => n >= index);

            foreach (var item in query1)
            {
                Console.WriteLine(item);
            }

            Console.Write("Press Enter key to see the other
elements...");
            Console.ReadLine();
            var query2 = numbers.SkipWhile((n, index) => n >= index);

            foreach (var item in query2)
            {
                Console.WriteLine(item);
```

```
            }
            Console.WriteLine();
        }
    }
}
```