

**Sofia University**  
**Department of Mathematics and Informatics**

**Course :** OO Programming with C#.NET

**Date:** November 2, 2020

**Student Name:**

**Lab No. 5**

**Submit the all C# .NET files developed to solve the problems listed below. Use comments and Modified-Hungarian notation.**

**Problem No. 1**

Use the `class Invoice` provided in the attached file `ex09_03.rar` to create an `array` of `Invoice` objects. Use the **sample data** shown below to initialize the elements of that array. `class Invoice` includes **four properties**- a `PartNumber` (type `int`), a `PartDescription` (type `string`), a `Quantity` of the item being purchased (type `int`) and a `Price` (type `decimal`).

**Write a console application** that performs the following **queries** on the array of `Invoice` objects and **displays the results**:

- a) Use LINQ to **sort** the `Invoice` objects by `PartDescription`.
- b) Use LINQ to **sort** the `Invoice` objects by `Price`.
- c) Use LINQ to **select** the `PartDescription` and `Quantity` and **sort** the results by `Quantity`.
- d) Use LINQ to **select** from each `Invoice` the `PartDescription` and the value of the `Invoice` (i.e., `Quantity * Price`). Name the calculated column `InvoiceTotal`. **Order the results** by `Invoice` value. [*Hint:* Use `let` to store the result of `Quantity * Price` in a **new range variable** `total`.]
- e) Using the results of the LINQ query in *Part d*, select the `InvoiceTotals` in the range \$200 to \$500.
- f) **Group invoices in two groups**- invoices with Unit `price` below \$12 and invoices with unit `price` above of equal to \$12. Display details about invoices in each group **sorted in ascending order of the price**
- g) Group invoices in subgroups, where the outer subgroup is defined by the first letter in the part description and the inner group comprises the invoice details of invoices whose part description starts with the corresponding first letter.

```

C:\WINDOWS\system32\cmd.exe

Groups and subgroups
First letter group: E
Electric sander
83   Electric sander      7   $57.98
First letter group: P
Power saw
24   Power saw           18  $99.99
First letter group: S
Sledge hamme
7    Sledge hamme       11  $21.50
Screwdriver
68   Screwdriver        11  $6.99
68   Screwdriver       122  $6.99
68   Screwdriver       150  $6.99
First letter group: H
Hammer
77   Hammer           76  $11.99
77   Hammer           11  $11.99
77   Hammer           32  $11.99
77   Hammer           55  $11.99
First letter group: L
Lawn mower
39   Lawn mower         3   $79.50
First letter group: J
Jig saw
56   Jig saw            21  $11.00
First letter group: W
Wrench
3    Wrench              34  $7.50
Press any key to continue . . .

```

h)

```

C:\WINDOWS\system32\cmd.exe

Category: Price above $12
68   Screwdriver        106  $6.99
3    Wrench              34   $7.50
56   Jig saw            21   $11.00
77   Hammer           76   $11.99
Category: Price below $12
7    Sledge hammer     11   $21.50
83   Electric sander    7    $57.98
39   Lawn mower         3    $79.50
24   Power saw          18   $99.99

```

- i) **Group invoices in three groups-** invoices with Unit price below \$10 , invoices with unit price between \$10 and \$20 and invoices with unit price above of equal to \$20. **Display details** about invoices in each group **sorted in ascending order of the price**

```

C:\WINDOWS\system32\cmd.exe
Category: Invoices with prices below $10
68  Screwdriver      106  $6.99
3   Wrench            34   $7.50
Category: Invoices with prices between $10 and $20
56  Jig saw           21   $11.00
77  Hammer           76   $11.99
Category: Invoices with prices above $20
7   Sledge hammer    11   $21.50
83  Electric sander   7    $57.98
39  Lawn mower        3    $79.50
24  Power saw         18   $99.99
Press any key to continue . . .

```

**Note:** The LINQ queries (a- g) should be returned by static methods in class Invoice, where each such method takes an array of Invoice instances as an argument. The Console application calls these methods and displays the returned LINQ queries using a generic static method.

- Write also the versions of LINQ making use of methods and Lambda expressions.
- Write versions of generic static method in the Console application making use of the Parallel.For , Parallel.ForEach from TPL and the ForAll() from PLINQ to display the results of the LINQ queries (a- e)
- Compare the time used to execute the LINQ queries with each one of these methods making use of an instance of class Stopwatch.

Sample data

Part number	Part description	Quantity	Price
83	Electric sander	7	57.98
24	Power saw	18	99.99
7	Sledge hammer	11	21.50
77	Hammer	76	11.99
39	Lawn mower	3	79.50
68	Screwdriver	106	6.99
56	Jig saw	21	11.00
3	Wrench	34	7.50

### Problem No. 1b

Consider the following array

```

City[] cities =
{
    new City(){ContinentName = "Europe", CountryName="Germany", CityName="Berlin" },
    new City(){ContinentName = "Europe", CountryName="Germany", CityName="Frankfurt" },
    new City(){ContinentName = "Europe", CountryName="France", CityName="Paris" },
    new City(){ContinentName = "Europe", CountryName="France", CityName="Marseille" },
    new City(){ContinentName = "Asia", CountryName="Japan", CityName="Tokyo" },
}

```

```
new City(){ContinentName ="Asia", CountryName="Korea", CityName="Seul" }
};
```

Write LINQ statements to group the array element by ContinentName and subgroup of CountryName per continent with a list of cities per country.

## **Problem No. 2**

**Write a console application that inputs a sentence from the user (assume no punctuation), then determines and displays the *nonduplicate words* in *alphabetical order*.**

Treat **uppercase** and **lowercase** letters the same.

**[Hint:** You can use **string** method **Split** with no arguments, as in **sentence.Split()** , to break a sentence into an **array** of **strings** containing the individual words. By default, **Split** uses **spaces** as delimiters. Use string method **ToLower** in the **select** and **orderby** clauses of your LINQ query to obtain the **lowercase** version of each word.]

## **Problem No. 3a**

**Write a console application that inserts 30 random letters into a `List< char >`. Perform the following queries on the `List` and **display** your results:**

**[Hint: Strings** can be indexed like arrays to access a character at a specific index.]

- Use LINQ to **sort** the `List` in **ascending** order.
- Use LINQ to **sort** the `List` in **descending** order.
- Display** the `List` in ascending order with **duplicates** removed

## **Problem No. 3b**

A strong brand, hot IPO (*Initial Public Offering marks the start of a company's publicly traded life*) and an intense engineering culture help make search advertising company Google a dream employer for a lot of people. Now, a new aptitude test Google is circulating purports to find the best and brightest. Alan Eustace, vice president of engineering and research for Mountain View, Calif.-based Google, published an aptitude test for "uber-geeks."

The 21 GLAT questions include engineering brain-twisters such as, "Consider a function which, for a given whole number  $n$ , returns the number of ones required when writing out all numbers between 0 and  $n$ . For example,  $f(13)=6$ . Notice that  $f(1)=1$ . What is the next largest  $n$  such that  $f(n)=n$ ?"

**Write a console application** to solve this problem.

Hint. Use the ToCharArray() method of convert the string representation of an integer into an array of chars and apply LINQ to solve the problem

#### **Problem No. 4a**

Using a random generator create an array of integers of 100 elements that take random values in the range [20, 50]. **Write a Console application that uses LINQ grouping** to partition the array of numbers by their remainder when divided by 8. Display the total numbers in each group, as well as , the numbers that constitute that group

#### **Problem No. 4b**

Given a List of strings ( "blueberry", "chimpanzee", "abacus", "banana", "apple", "cheese") write a **Console application that uses LINQ grouping** to partition the list of strings in groups of their first letter. Display the total of strings in each group, a title for each group , as well as , the strings that constitute each group

Example:

**Words that start with the letter 'b':**

blueberry banana

**Words that start with the letter 'c':**

chimpanzee cheese

#### **Problem No. 5a**

Run the Task Manager and open the Performance Wizard to view CPU usage on your computer. Run the attached C# project **MultMatricesInTPL**. Experiment increasing the number of columns and rows in the matrices. The larger the matrices, the greater the performance difference between the parallel and sequential versions of the computation. When the matrix is small, the sequential version will run faster because of the overhead in setting up the parallel loop

#### **Problem No. 5b**

Write a Console application to **multiply a matrix by a row** using parallel For. Compare the execution of a sequential and a parallel execution of the task.

#### **Problem No. 5c**

Run the attached project **ReturnAverageTPL**. Modify its code so that the parallel **For** returns the average value of all the sums generated inside the loop.

### Problem No. 6

Given

```
var customers new[] {  
    new { ID = 1,  FirstName = "Sandeep" , LastName = "Ramani" },  
    new { ID = 2,  FirstName = "Dharmik" , LastName = "Chotaliya" },  
    new { ID = 3,  FirstName = "Nisar"    , LastName = "Kalia" } ,  
    new { ID = 4,  FirstName = "Ravi"     , LastName = "Mapara" } ,  
    new { ID = 5,  FirstName = "Hardik"   , LastName = "Mistry" } ,  
    new { ID = 6,  FirstName = "Sandy"    , LastName = "Ramani" },  
    new { ID = 7,  FirstName = "Jigar"    , LastName = "Shah"  },  
    new { ID = 8,  FirstName = "Kaushal"  , LastName = "Parik" } ,  
    new { ID = 9,  FirstName = "Abhishek" , LastName = "Swarnker" } ,  
    new { ID = 10, FirstName = "Sanket"   , LastName = "Patel" } ,  
    new { ID = 11, FirstName = "Dinesh"   , LastName = "Prajapati" },  
    new { ID = 12, FirstName = "Jayesh"   , LastName = "Patel" },  
    new { ID = 13, FirstName = "Nimesh"   , LastName = "Mishra" } ,  
    new { ID = 14, FirstName = "Shiva"    , LastName = "Reddy" } ,  
    new { ID = 15, FirstName = "Jasmin"   , LastName = "Malviya" },  
    new { ID = 16, FirstName = "Hares"    , LastName = "Bhanderi" },  
    new { ID = 17, FirstName = "Ankit"    , LastName = "Ramani" },  
    new { ID = 18, FirstName = "Sanket"   , LastName = "Shah" } ,  
    new { ID = 19, FirstName = "Amit"     , LastName = "Shah" } ,  
    new { ID = 20, FirstName = "Nilesh"   , LastName = "Soni" }    };
```

- Write a PLINQ query to select the customers with IDs in the range between 5 and 15, while preserving the order of the IDs in the output
- Write a PLINQ query to select the customers with distinct LastName
- Write a PLINQ query to select the customer ID, and the FirstName and LastName concatenated by a comma and a space between them

Use `Parallel.ForEach` and `ParallelQuery ForAll` methods to output the results