**Code & Module Title**



**CS6004NI Application Development**

**Assessment Weightage & Type**

**30% Individual Coursework**

**Year and Semester**

**2018-19 Autumn**

**Student Name: Rajat Shrestha**

**London Met ID:** **17030954**

**College ID: np01cp4a170021**

**Assignment Due Date: 17th January 2020**

**Assignment Submission Date: 17th January 2020**

*I confirm that I understand my coursework needs to be submitted online via Google Classroom under the relevant module page before the deadline for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a mark of zero will be awarded.*

Table of Contents

[1. Introduction 3](#_Toc30190185)

[1.1. Instructions: 3](#_Toc30190186)

[1.1.1. Adding New Student: 4](#_Toc30190187)

[1.1.2. Sort by name and Registration date: 5](#_Toc30190188)

[1.1.3. Edit or Delete Row: 6](#_Toc30190189)

[1.1.4. Viewing the Total enrolled students’ chart: 7](#_Toc30190190)

[1.1.5. Viewing the Weekly report from a certain date: 8](#_Toc30190191)

[2. Software Architecture 9](#_Toc30190192)

[2.1. Classes from other sources 10](#_Toc30190193)

[3. Classes and Methods: 11](#_Toc30190196)

[3.1. Class Diagram 11](#_Toc30190197)

[3.2. Attributes of Student: 12](#_Toc30190198)

[3.4. Method Descriptions: 13](#_Toc30190199)

[3.4.1. Method description of Student Class 13](#_Toc30190200)

[3.4.2. Method description of StudentForms class 14](#_Toc30190201)

[3.4.3. Method description of Utility Class 16](#_Toc30190202)

[3.4.4. Method description of Static Program class 16](#_Toc30190203)

[4. Report generation 17](#_Toc30190204)

[5. Core Mechanisms 18](#_Toc30190205)

[5.1. Data Structures: 18](#_Toc30190206)

[5.2. Insertion Sort: 19](#_Toc30190208)

[6. Reflection 20](#_Toc30190209)

[References 21](#_Toc30190210)

[Appendix 22](#_Toc30190211)

Table of Figures:

[Figure 1: Main Window with all the interactable components 3](#_Toc30179743)

[Figure 2: Filling in the Form 4](#_Toc30179744)

[Figure 3: Added student at the bottom of the list 4](#_Toc30179745)

[Figure 4: Names Sorted in ascending order 5](#_Toc30179746)

[Figure 5: Names Sorted in descending order (repressing Sort Name button) 5](#_Toc30179747)

[Figure 6: Updating the Student Details 6](#_Toc30179748)

[Figure 7: Updated Student details 6](#_Toc30179749)

[Figure 8: Total Students Enrolled Chart Page 7](#_Toc30179750)

[Figure 9: Weekly report form the selected date 8](#_Toc30179751)

[Figure 10: Software Architecture 9](file:///C:\Users\Rajat\Desktop\AD%2017030954%20Rajat%20Shrestha.docx#_Toc30179752)

[Figure 11: Overall Class Diagram 11](#_Toc30179753)

[Figure 12: Flow-Chart for Report Generation 17](#_Toc30179754)

[Figure 13: Insertion Sort Steps (w3resource, 2019) 19](#_Toc30179755)

Table of Tables:

[Table 1: Description of attributes of Student Class 12](#_Toc30165084)

[Table 2: Method Description of Student Class 13](#_Toc30165085)

[Table 3: Method Description of StudentForm Class 14](#_Toc30165086)

[Table 4: Method Description of Utility Class 16](#_Toc30165087)

[Table 5: Method Description of Static Program Class 16](#_Toc30165088)

# Introduction

This is the report for the first coursework of the Application development module which has the detailed information on the student registration management software which allows users to add, edit and view student details. This report describes and gives instructions of all the features to show the reports on total amount of students enrolled in each course(total/weekly). The major features of this program are:

* Stores the Details of the students
* Displays the student records in the table right after the student is enrolled.
* Can edit the student record if needed
* Can delete the undesired entries
* Displays the weekly report from a given date.
* Displays the total enrollment count of students in a chart.
* Displays and manages the enrollment status of the student.
* Imports the info from a csv file and writes the updated student details.

## Instructions:

The Program when run opens the following window:

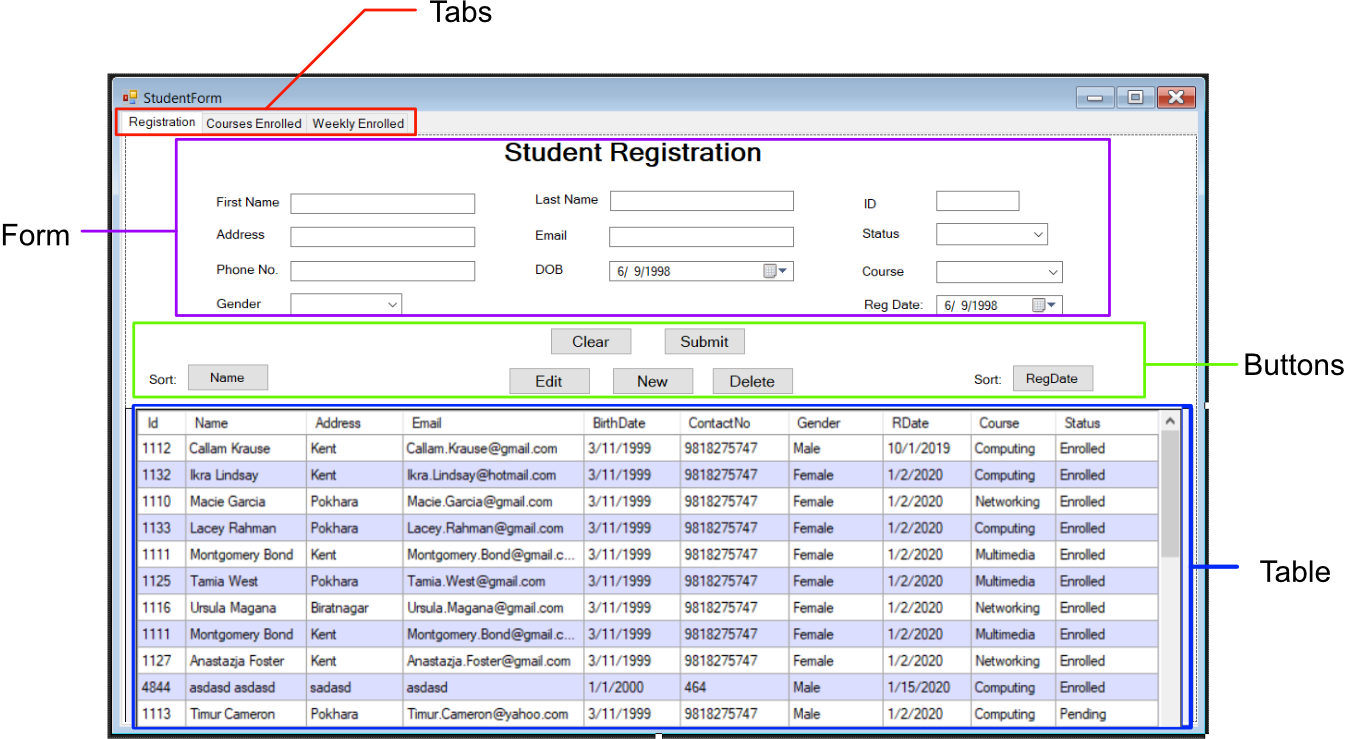


Figure 1: Main Window with all the interactable components

### Adding New Student:

1. Fill the Form on the main window (Registration Tab)
2. Click Submit

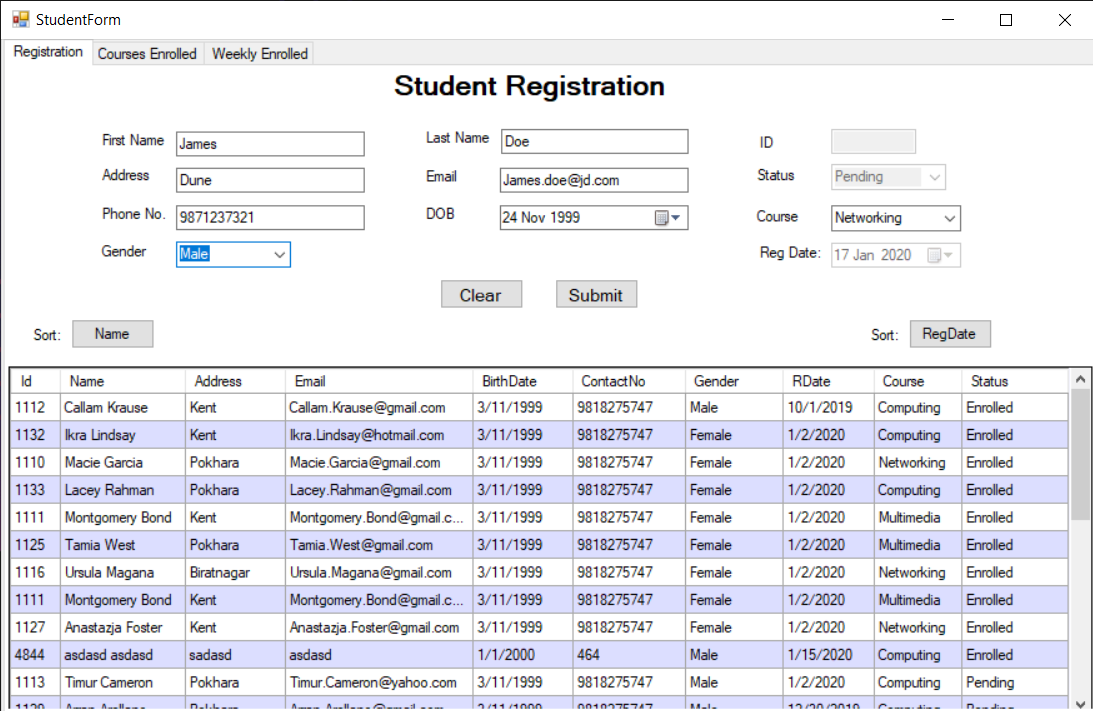


Figure 2: Filling in the Form

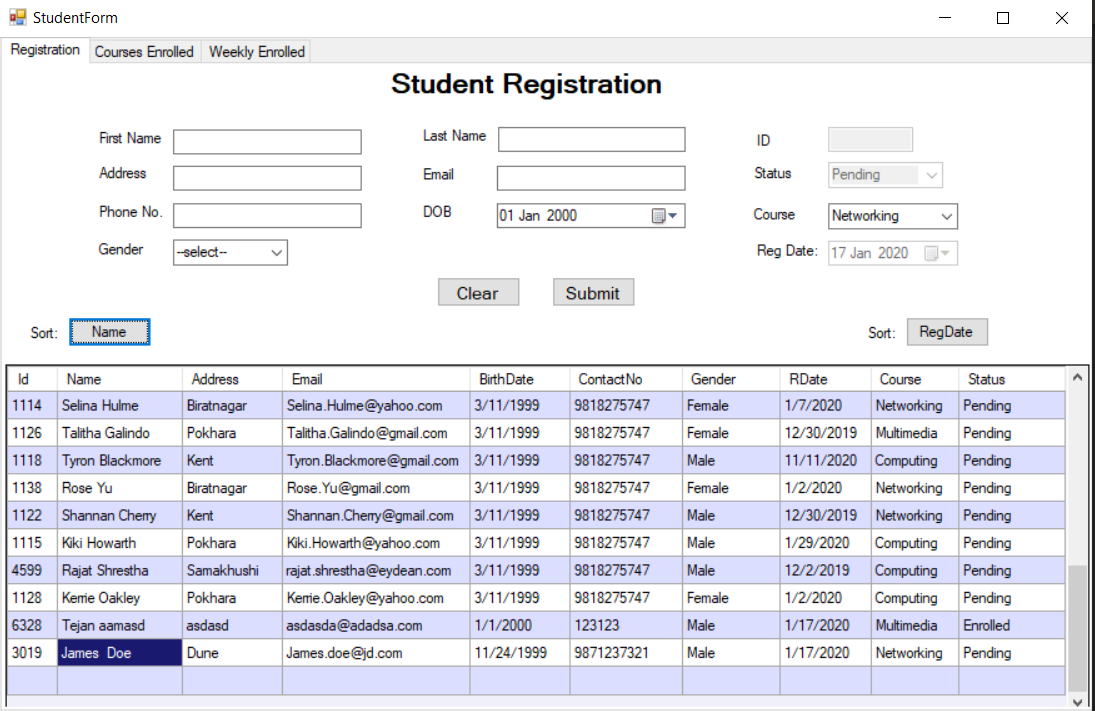


Figure 3: Added student at the bottom of the list

### Sort by name and Registration date:

1. Click the Sort Button(Name or RegDate) for Sorting in Ascending order
2. Press the same button for Descending Order

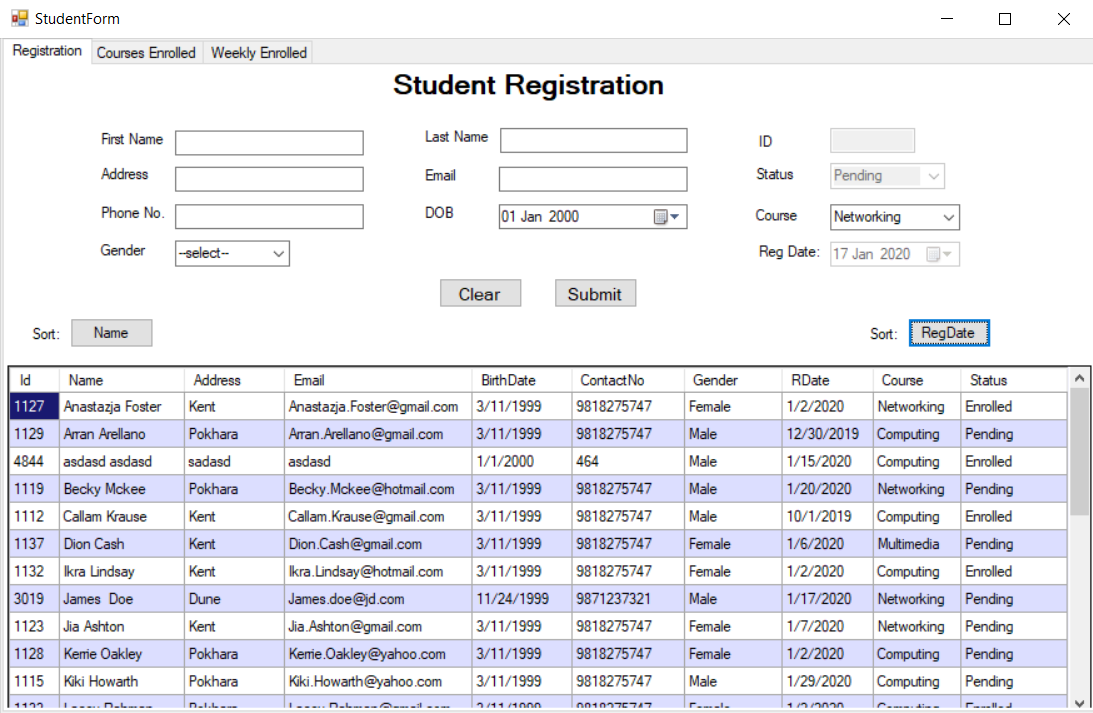


Figure 4: Names Sorted in ascending order

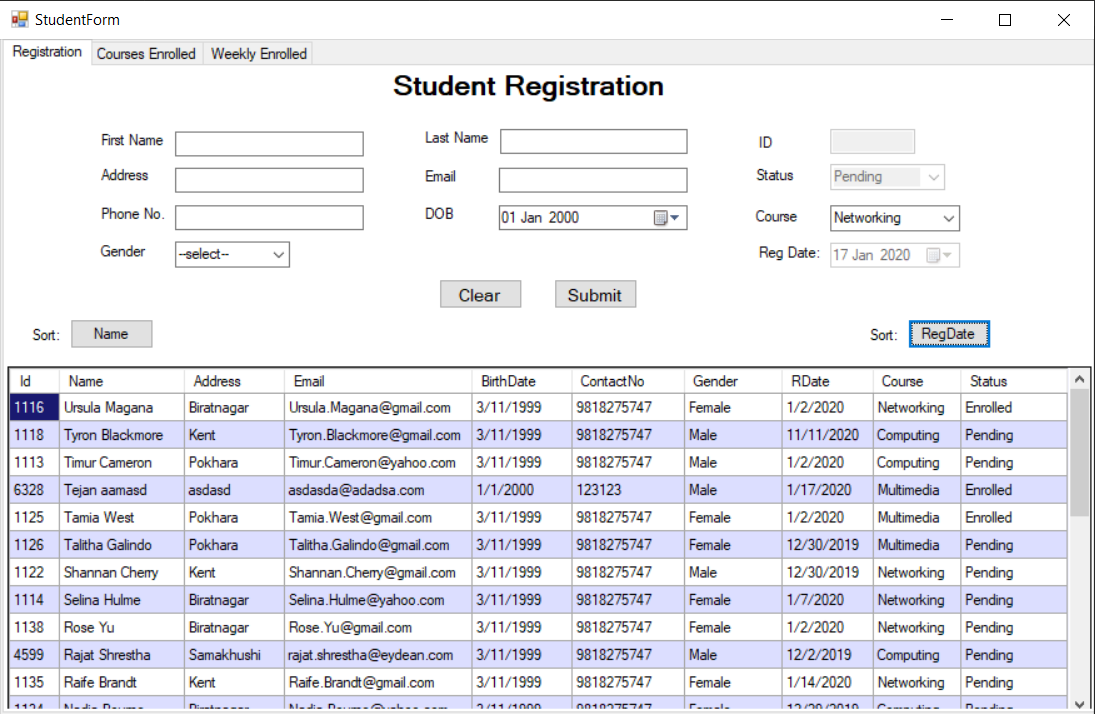


Figure 5: Names Sorted in descending order (repressing Sort Name button)

### Edit or Delete Row:

* + - 1. Select the row to update
      2. The information appears on the forms
      3. Edit the form
      4. Press Edit Button or to delete press delete
      5. The content is updated

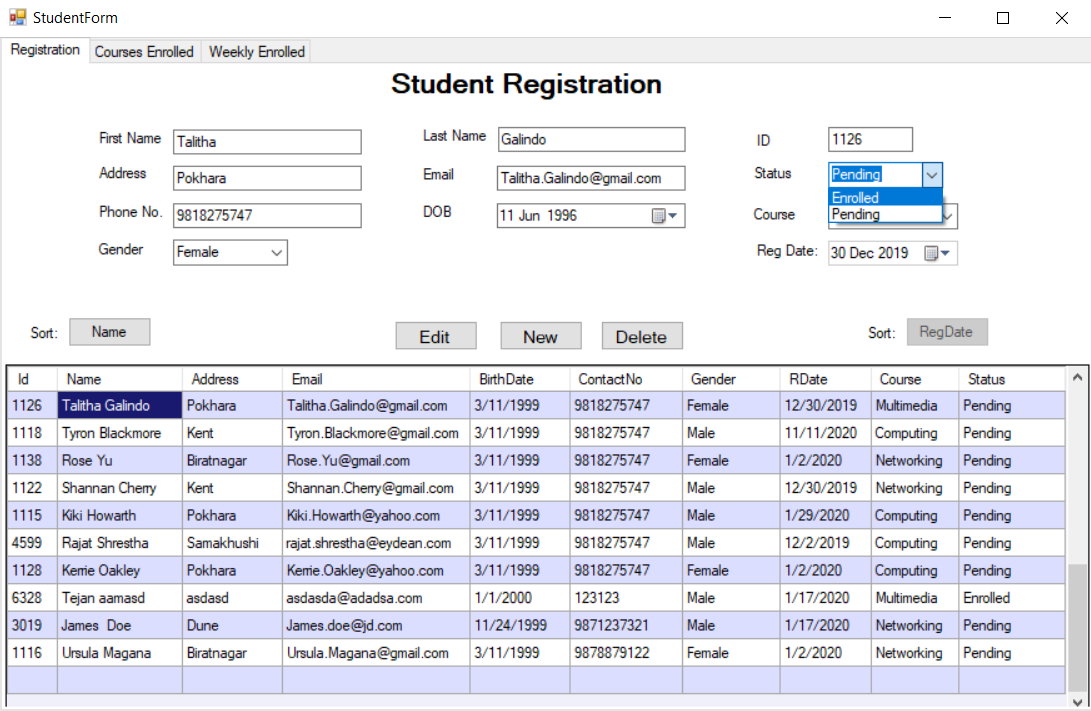


Figure 6: Updating the Student Details

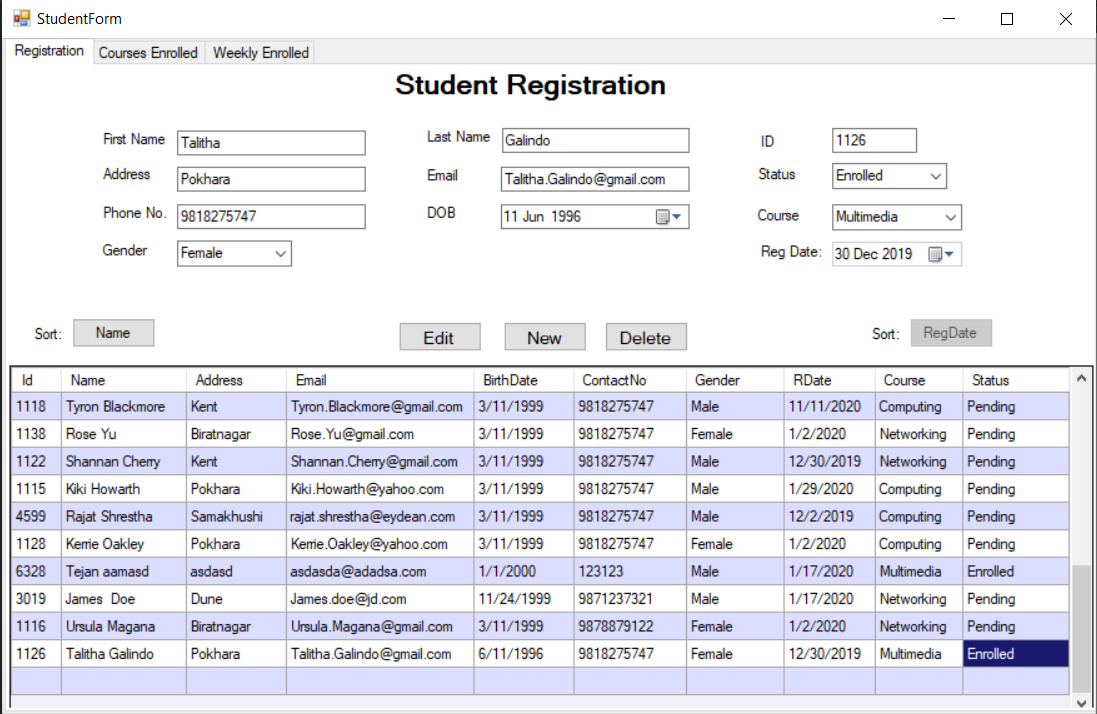


Figure 7: Updated Student details

### Viewing the Total enrolled students’ chart:

* + - 1. Go to the Second tab (Courses Enrolled)
      2. The Chart is generated

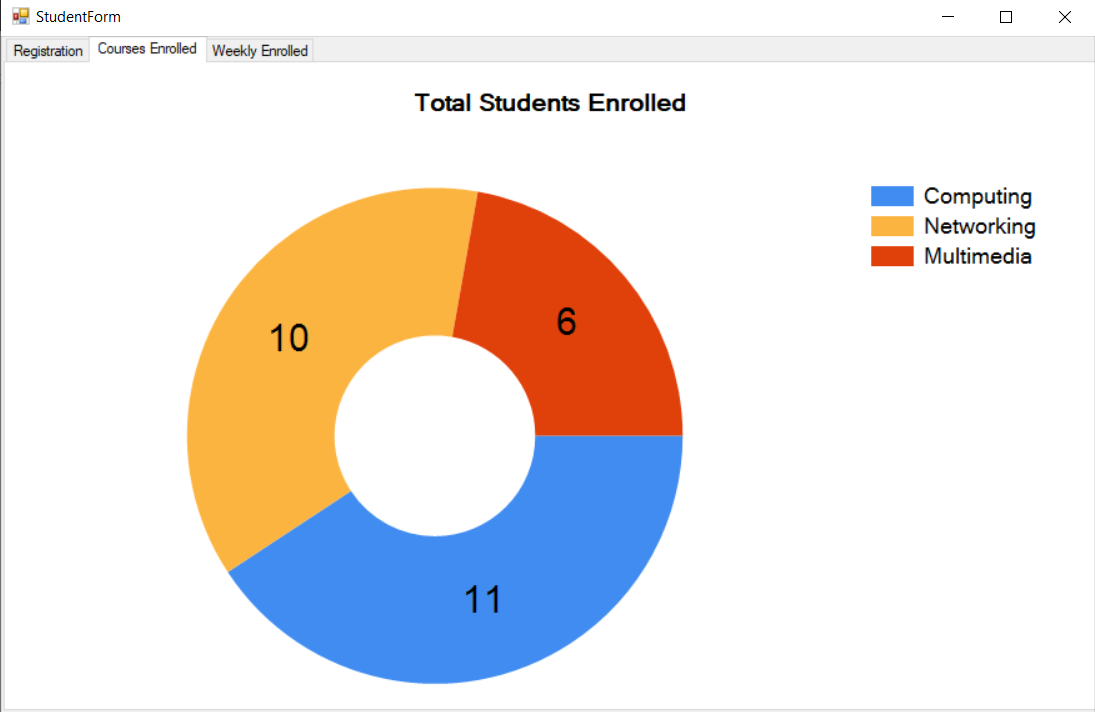
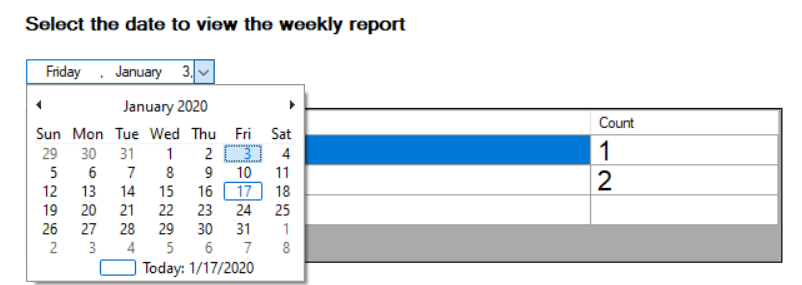


Figure 8: Total Students Enrolled Chart Page

### Viewing the Weekly report from a certain date:

1. Go to the Third tab (Weekly Report)
2. Select the day from when weekly report is to be generated
3. The Weekly report from the selected date is shown with the count of students in each course and their information in the table below in tab-3.



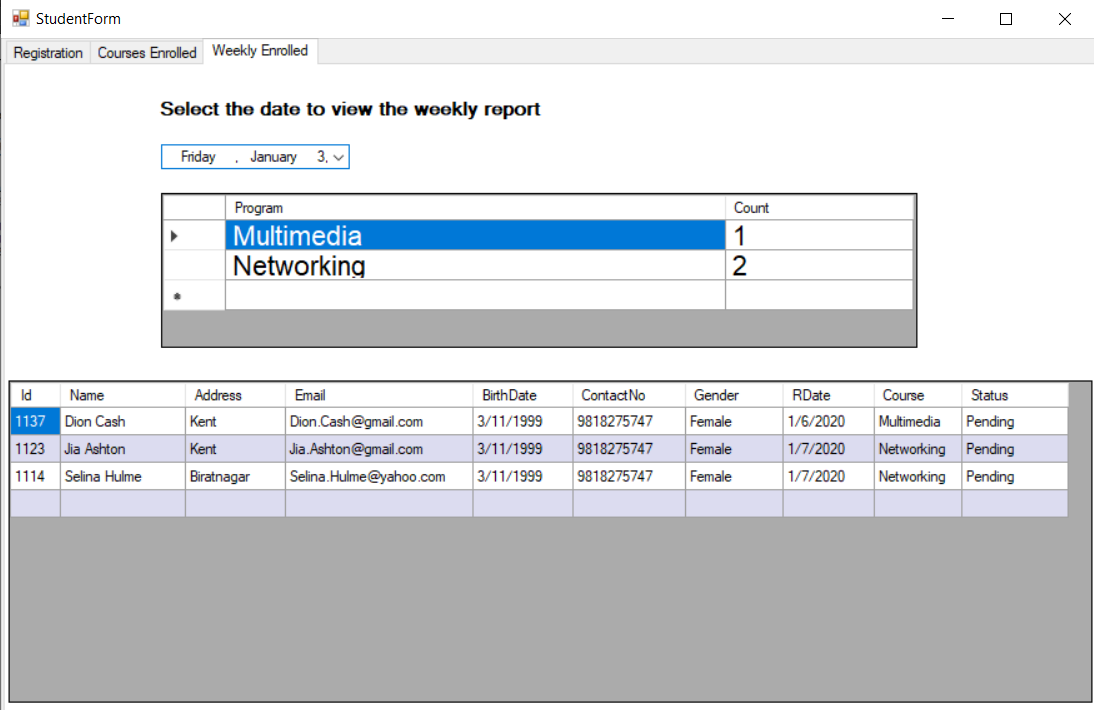


Figure 9: Weekly report form the selected date

# Software Architecture

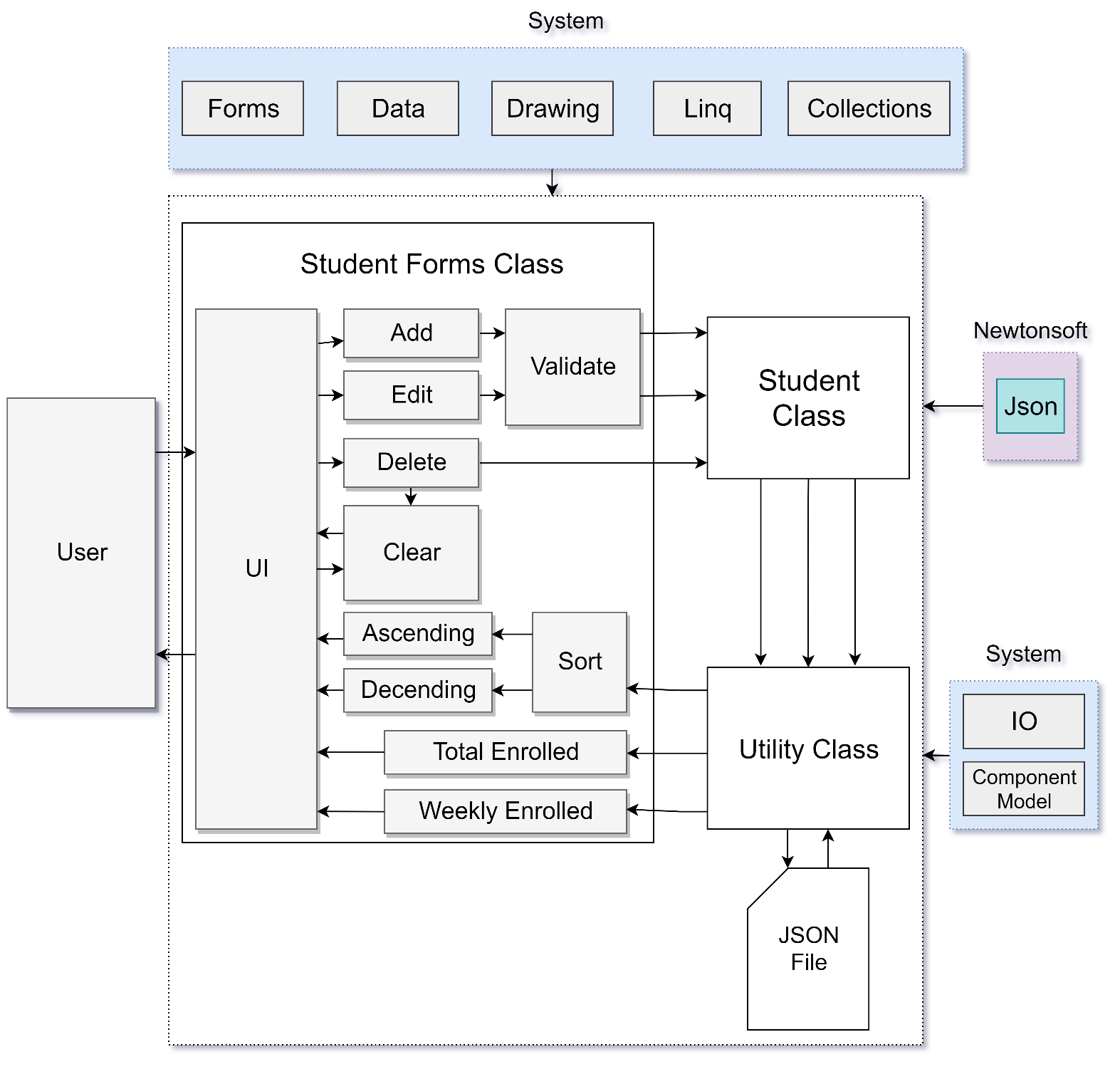
This software is quite simple with only 4 classes built in the CourseWork namespace. The classes import essential system components. The following diagram represents how the overall system function, and the how the system utilized imported modules:

Figure 10: Software Architecture

(The colored Modules are imported i.e. System and Newtonsoft)

## Classes from other sources

Various classes were used in the construction of the Form to make the development easier. The Student form class was directly extended from the Form Class; Most of the System classes were used for storing data in objects, read/write purposes and for basic functionalities; The Newtonsoft.Json package was installed for helping in serialization on the Student class.

### Windows.Forms (Form)

This namespace is used to inherit the form class used in the StudentForm class. Its main purpose is to provide an easy GUI interface for creating form. Windows Forms Applications are the applications which are developed by using Windows Forms (WinForms) and are supported on windows. WinForms limited to Windows Forms Applications and provides different type of UI elements (Microsoft, 2019). In this project form Is expanded to be used as an application for student registration and report preview.

### Other used classes and packages:

These classes were used in various parts of the project which are provided by the system.

* Newtonsoft.Json (JsonConvert)
* System.Collections.Generic (List<T>)
* System.Data (DataTable)
* System.Drawing (Chart)
* System.IO (StreamWriter and StreamReader)
* System.Linq
* System.ComponentModel

(Note: The Newtonsoft.Json package requires to be installed manually.)

# Classes and Methods:

## Class Diagram

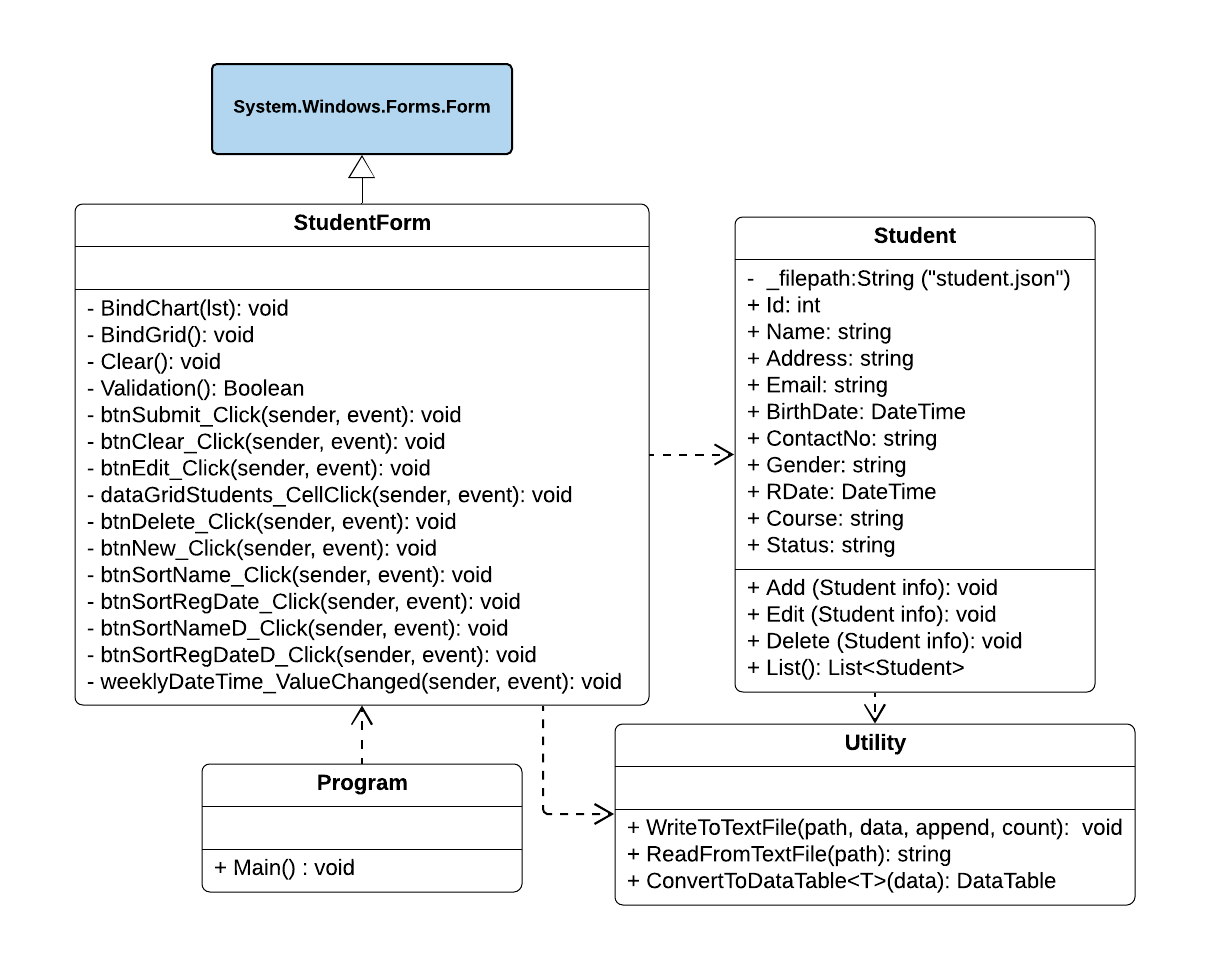
 Class diagram is fundamental diagram type in UML, which is used to represent the static relationship between different components of an application, which is key to understand the overall working of any system. A class is group of objects with common state and behavior and is represented in a UML class diagram by putting Name of the class at the first box, attributes of the class at the middle box and finally the methods of that class at the last box (Pilone & Pitman, 2005).

Figure 11: Overall Class Diagram

## Attributes of Student:

The above class diagram shows the attributes used in the Student class which is used to define a Student Object. The attributes and their significance in detail is as follows:

Table 1: Description of attributes of Student Class

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| \_filepath | Name of the json file to store the Student information in String. |
| Id | Unique ID in int auto generated during the creation. |
| Name | Name of the student consisting of First and Last Name in String. |
| Address | Address of the Student in String |
| Email | E-mail address of the student in String |
| Birthdate | Date of Birth of the Student in DateTime format. |
| ContactNo | Phone Number of the Student in String. |
| Gender | Gender of the Student in String. |
| RDate | Registration Date of the Student. |
| Course | Enrolled Course of the Student in String. |
| Status | Current Enrolment Status of the Student |

*(Note: The other Classes do not have any key attributes, so they are skipped.)*

## Method Descriptions:

A method description contains brief description of statements that are put together to perform an operation in a class (Jenkov, 2015). It helps to make the code modular and reuse the code. The C# program to manage the information regarding students were created with four different classes within the “CourseWork” namespace.

### Method description of Student Class

Table 2: Method Description of Student Class

|  |  |
| --- | --- |
| **Modifiers, Return Type and Method** | **Method description** |
| public void  **Add**(Student info) | The add method is responsible for adding student information received from the from input to the text file. This method also gives random id to each enrolled student. |
| public void  **Edit**(Student info) | The edit option to update details of enrolled students. |
| public void  **Delete**(Student info) | The delete method is to delete the records of the already enrolled student. If the user wants to delete records of student. |
| public List<Student>  **List**() | This method reads data from text file and shows it on the data grid table. |

### Method description of StudentForms class

Table 3: Method Description of StudentForm Class

|  |  |
| --- | --- |
| **Modifiers, Return Type and Method** | **Method description** |
| public  **StudentForm**() | Method which initializes all the components and sets up all the variables required for the components |
| private Boolean  **Validation**() | A method which checks all the entry fields and returns a Boolean to indicate its validation. |
| private void  **BindChart**(List<Student> list) | The function of this method is to add the Student enrolment details in each subject in the chart. The Bind Chart is used to show the count of each course on chart in form of a bar graph. |
| private void  **BindGrid**() | The method which converts the list of students to a DataTable so it can be displayed in the table. This method also initiates the BindChart() method which displays the enrolment chart. |
| private void  **Clear**() | The method in StudentForm table is to clear all the fields in the Form from where all the information of the students is entered. |
| private void  **btnSubmit\_Click**(object sender,  EventArgs e) | This method is for adding new student details from the forms. If the validation returns true, all the input information from the form fields is stored in the JSON file and is displayed on the table. |
| private void  **btnClear\_Click**(object sender,  EventArgs e) | This method invokes Clear function which clears the inputs fields |
| private void  **btnEdit\_Click**(object sender,  EventArgs e) | This method calls the edit method in student class to edit the student records updating the student details whenever user clicks the Edit button. |
| private void  **dataGridStudents\_CellClick**(object  sender, DataGridViewCellEventArgs e) | This method returns the content of the selected row in the DataGrid table for editing or deleting. |
| private void  **btnDelete\_Click**(object sender,  EventArgs e) | The function of this method is this method is to call delete method to delete the record of selected student. Before deleting the program asks for confirmation. |
| private void  **btnNew\_Click**(object sender,  EventArgs e) | The method got activated when the user clicks on New button and then it allows to enter the new record of students. |
| private void  **btnSortName\_Click**(object sender,  EventArgs e) | The method invoked after user presses the sort name option which uses insertion sort to sort all the names in ascending order. |
| private void  **btnSortRegDate\_Click**(object sender, EventArgs e) | Similarly, this method sorts the DataTable according to the registration date in ascending order. The button for descending order is |
| private void  **btnSortNameD\_Click**(object sender,  EventArgs e) | The method to arrange the names in the DataTable descending order. |
| private void  **btnSortRegDateD\_Click**(object sender,  EventArgs e) | The method to arrange the Registration date in the DataTable descending order. |
| private void  **weeklyDateTime\_ValueChanged**(object  sender, EventArgs e) | This method is invoked after the user selects a date in the weekly report calendar. It then queries all the stored data and views only the students who have been enrolled after 7 days of the selected date. |

### Method description of Utility Class

Table 4: Method Description of Utility Class

|  |  |
| --- | --- |
| **Modifiers, Return Type and Methods** | **Method description** |
| public static void  **WriteToTextFile**(string path, string data, bool append = true,  int count = 1) | Method to store the information received from the form in the text file. It uses StreamWriter to write the JSON structured data in the file. |
| public static string  **ReadFromTextFile**(string path) | Method to read the information of students from the text file. The method first checks either or not file exists. If the file exists retrieves the information inside it by using StreamReader to read JSON file. This method returns the read data in string format. |
| public static DataTable  **ConvertToDataTable**<T>(IList<T> data) | This method converts the data and returns it in the DataTable format so that the information of the students can be displayed on the table. |

### Method description of Static Program class

Table 5: Method Description of Static Program Class

|  |  |
| --- | --- |
| **Modifiers, Return Type and Method** | **Method description** |
| static void  **Main**() | The Static method to initiate a new StudentForm which runs the overall program. |

# Report generation

This application has a feature to generate and view all the students enrolled so far in the application. This section describes how these report generation works by providing a brief description and a descriptive diagram (flow-chart).

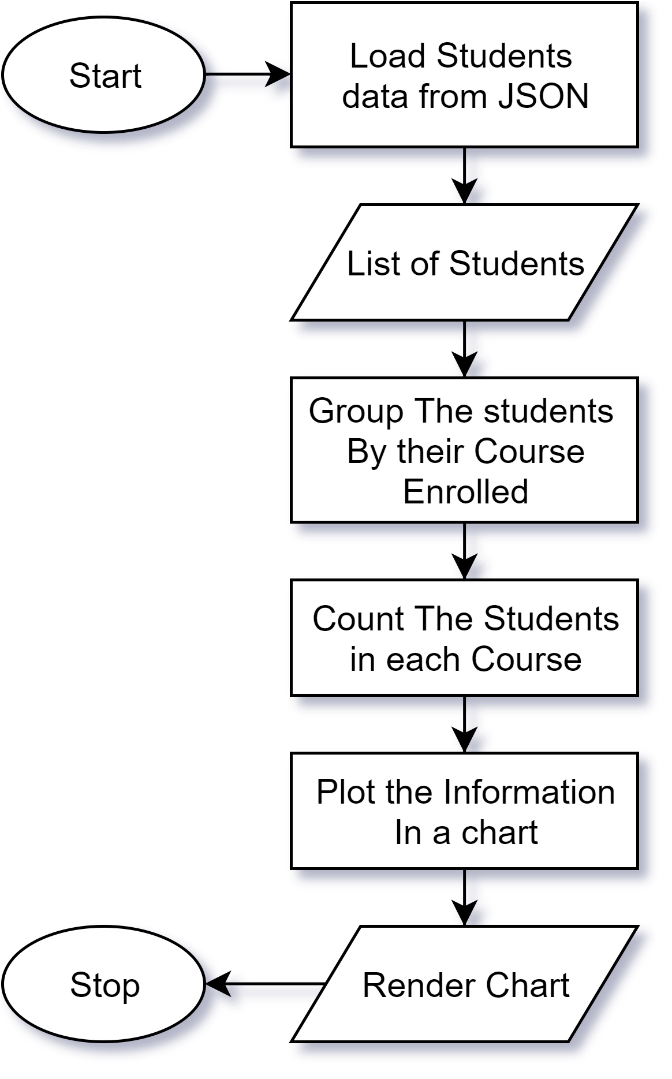


Figure 12: Flow-Chart for Report Generation

# Core Mechanisms

## Data Structures:

A data structure is a specialized class created for organizing, processing, retrieving and storing data efficiently and can be a key aspect in the performance of any software. There are many simple to complex data structures which are used for various purposes and are arranged to work effectively in any software. Each data structure contains information about the data values, relationships between the data and functions that can be applied to the data (Storer, 2012). Some examples of data structures are: Arrays, Stacks, Queues, LinkedList, trees. Among all the available data structures, list data structure is used throughout he projects for storing data and for various operations.

### List:

In C#, List is a generic collection which is used to store the elements or objects in the form of a list, and it is defined under System.Collections.Generic namespace. It provides the same functionality as ArrayList, but a list is a generic whereas ArrayList is a non-generic collection which means the size of a list is dynamic and can grow gradually as needed (MSDN, 2019). A List class has 3 constructors which are used to create a list and the constructors are as follows:

* List<T>(): For initiating empty list
* List<T>(Enumerable): List with specified object type.
* List<T>(Int32): For creating a list with fixed capacity

Example:

List<string> my\_list1 = new List<string>() { “Kathmandu”, “See Sharp” };

## Insertion Sort:

Insertion sort is a simple linear sort based on the idea that one element from the input elements is consumed in each iteration to find its correct position. It iterates the input elements one by one by growing the sorted array at each iteration then repeatedly compares the current element with the largest value in the sorted array. If the current element is greater, the element is kept on its place, else it finds its correct position in the sorted array and moves it to that position. Finally, the sort is complete after all the elements are correctly sorted. Insertion sort has the time complexity of O(n2) which is not the best, but it works well on small to medium projects and has lesser memory consumption than the other more efficient sorts like Merge sort (Mahmoud, 2011). The following are some of the important characteristics of Insertion Sort:

* Efficient for smaller data sets and partially sorted instances, but very inefficient for large lists.
* It is better than Selection Sort and Bubble Sort algorithms but worse than Merge Sort.
* Its space complexity is less requiring only a single additional memory space.
* It is stable sorting technique, as it does not change the relative order of elements that are equal.

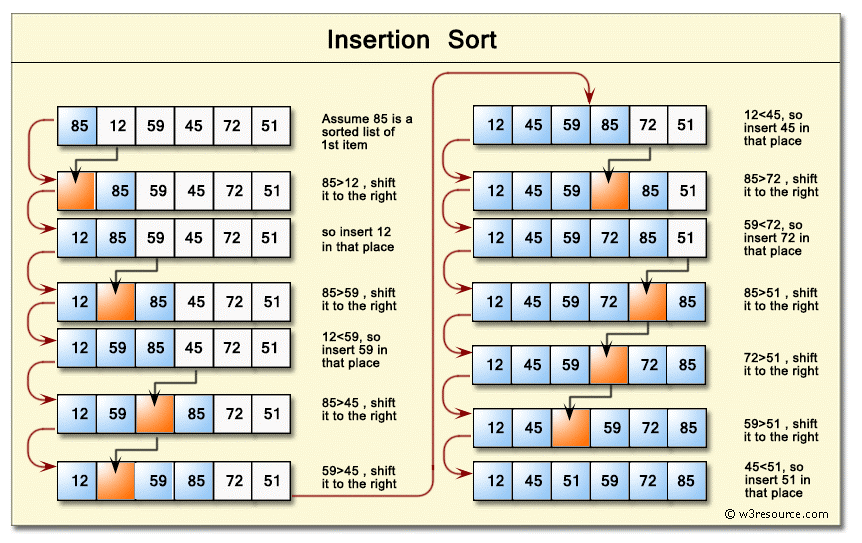


Figure 13: Insertion Sort Steps (w3resource, 2019)

# Reflection

There Was little to no problems faced while developing the windows form in C# using Visual Studio IDE. C# is a modern, simple, object-oriented, and type-safe programming language identical to C++, C and Java (MSDN, 2019). Since, most of the syntax for programming in C# was identical to syntax in Java there was little to no problem and a lot of productivity while coding in C#. The main features and advantages along with the disadvantages are as follows: of using C# with Visual Studio IDE are as follows:

**Advantages:**

1. C# being Object Oriented Programming Language,
2. C# has seamless capability to integrate with Windows,
3. C# is a compiled language,
4. Visual Studio has clean UI and good auto code refactoring for C# development,
5. Easy development due to intuitive design of Visual Studio,
6. Visual Studio has good debugging and other important development tools for C#

**Disadvantages:**

1. Visual Studio is quite Memory Intensive and takes a lot of RAM while running,
2. Some of the features such as hotkey for commenting code and Find/Replace is different in Visual Studio than in other Text Editors and IDEs,
3. Visual Studio is not open source,
4. C# does not work on other platforms (e.g. Linux).

Therefore, C# is a good programming language with a lot of advantages outweighing its disadvantages which allows developers to build powerful windows application easily along with Visual Studio IDE.C# and Visual Studio was quite useful in development of the given project which might have taken more time in other programming language.

# References

Hopping, C., 2018. *What is a graphical user interface?.* [Online]   
Available at: https://www.itpro.co.uk/operating-systems/30248/what-is-a-graphical-user-interface  
[Accessed 18 1 2019].

Jacobson, N. & Thornton, A., 2004. It is Time to Emphasize ArrayLists over Arrays. *inroads – The SIGCSE Bulletin ,* 36(4), pp. 88-92.

Jenkov, J., 2015. *Java Methods.* [Online]   
Available at: http://tutorials.jenkov.com/java/methods.html  
[Accessed 17 Jan 2018].

Mahmoud, H. M., 2011. *Sorting: A Distribution Theory.* First ed. New Jersy: John Wiley & Sons.

Microsoft, 2019. *Form Class.* [Online]   
Available at: https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.form?view=netframework-4.8  
[Accessed 16 January 2020].

MSDN, 2019. *A tour of the C# language.* [Online]   
Available at: https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/  
[Accessed 16 January 2020].

MSDN, 2019. *List<T> Class.* [Online]   
Available at: https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1?view=netframework-4.8  
[Accessed 16 January 2020].

Pilone, D. & Pitman, N., 2005. *UML 2.0 in a Nutshell.* First ed. Sebastapool: O'Reilly Media.

Rouse, M., 2014. *What is object-oriented programming (OOP)? - Definition from WhatIs.com.* [Online]   
Available at: http://searchmicroservices.techtarget.com/definition/object-oriented-programming-OOP  
[Accessed 18 Jan 2018].

Storer, J. A., 2012. *An Introduction to Data Structures and Algorithms.* First ed. Berlin: Springer Science & Business Media.

w3resource, 2019. *Python Data Structures and Algorithms: Insertion sort.* [Online]   
Available at: https://www.w3resource.com/python-exercises/data-structures-and-algorithms/python-search-and-sorting-exercise-6.php  
[Accessed 16 January 2020].

# Appendix

**Program.cs**

using System;

using System.Windows.Forms;

namespace Coursework

{

    static class Program

    {

        [STAThread]

        static void Main()

        {

            Application.EnableVisualStyles();

            Application.SetCompatibleTextRenderingDefault(false);

            Application.Run(new StudentForm());

        }

    }

}

**Utility.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.IO;

namespace Coursework

{

    public static class Utility

    {

        public static void WriteToTextFile(string path, string data,

bool append = true, int count = 1)

        {

            if (!File.Exists(path))

            {

                var file = File.Create(path);

                file.Close();

            }

            using (StreamWriter writer = new StreamWriter(path, append: append))

            {

                if (!append)                 {

                    data = data.Trim().Substring(1, data.Trim().Length - 1);

                    data = data.Trim().Substring(0, data.Trim().Length - 1);

                }

                if (count != 0)

                {

                    data = data + ",";

                }

                writer.WriteLine(data);

            }

        }

        public static string ReadFromTextFile(string path)

        {

            if (File.Exists(path))

            {

                string data;

                using (StreamReader r = new StreamReader(path))

                {

                    data = r.ReadToEnd();

                }

                if (data != "")

                {

                    data = "[" + data + "]";

                }

                return data;

            }

            return null;

        }

        public static DataTable ConvertToDataTable<T>(IList<T> data)

        {

            PropertyDescriptorCollection properties =

                TypeDescriptor.GetProperties(typeof(T));

            DataTable table = new DataTable();

            foreach (PropertyDescriptor prop in properties)

                table.Columns.Add(prop.Name, Nullable

.GetUnderlyingType(prop.PropertyType) ?? prop.PropertyType);

            if (data != null)

            {

                foreach (T item in data)

                {

                    DataRow row = table.NewRow();

                    foreach (PropertyDescriptor prop in properties)

                        row[prop.Name] = prop.GetValue(item) ?? DBNull.Value;

                    table.Rows.Add(row);

                }

            }

            return table;

        }

    }

}

**Student.cs**

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.Linq;

namespace Coursework

{

    class Student

    {

        private string \_filePath = "student.json";

        public  int Id { get; set; }

        public string Name { get; set; }

        public string Address { get; set; }

        public string Email { get; set; }

        public DateTime BirthDate { get; set; }

        public string ContactNo { get; set; }

        public string Gender { get; set; }

        public DateTime RDate { get; set; }

        public string Course { get; set; }

        public string Status { get; set; }

        public void Add(Student info)

        {

            Random r = new Random();

            info.Id = r.Next(1000, 9999);

            string data = JsonConvert.SerializeObject(info, Formatting.None);

            Utility.WriteToTextFile(\_filePath, data);

        }

        public void Edit (Student info)

        {

            List<Student> list = List();

            Student s = list.Where(x => x.Id == info.Id).FirstOrDefault();

            list.Remove(s);

            list.Add(info);

            string data = JsonConvert.SerializeObject(list, Formatting.None);

            Utility.WriteToTextFile(\_filePath, data, false);

        }

        public void Delete (Student info)

        {

            List<Student> list = List();

            Student s = list.Where(x => x.Id == info.Id).FirstOrDefault();

            list.Remove(s);

            string data = JsonConvert.SerializeObject(list, Formatting.None);

            Utility.WriteToTextFile(\_filePath, data, false);

        }

        public List<Student> List()

        {

            string student = Utility.ReadFromTextFile(\_filePath);

            if (student != null)

            {

               List<Student> lst = JsonConvert.DeserializeObject<List<Student>>(student);

                return lst;

            }

            return null;

        }

    }

}

**StudentForm.cs**

using System;

using System.Collections.Generic;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

List<string> my\_list1 = new List<string>() { “Record\_1”, “Record\_2” };

namespace Coursework

{

    public partial class StudentForm : Form

    {

        public StudentForm()

        {

            InitializeComponent();

            BindGrid();

            dtpDOB.Format = DateTimePickerFormat.Custom;

            dtpRDate.Format = DateTimePickerFormat.Custom;

            dtpDOB.CustomFormat = "dd MMM yyyy";

            dtpRDate.CustomFormat = "dd MMM yyyy";

            dtpDOB.Value = DateTime.Parse("1/1/2000");

            dtpRDate.Value = DateTime.Now;

            dtpRDate.Enabled = false;

            txtID.Enabled = false;

            btnSortNameD.Enabled = false;

            btnSortRegDateD.Enabled = false;

            cbGender.SelectedItem = null;

            cbGender.SelectedText = "--select--";

            cbStatus.SelectedItem = "Pending";

            cbStatus.Enabled = false;

            btnEdit.Visible = false;

            btnDelete.Visible = false;

            btnNew.Visible = false;

            btnSortRegDateD.Visible = false;

            lblsort1.Visible = true;

            lblsort2.Visible = true;

            btnSortName.Visible = true;

            btnSortRegDate.Visible = true;

            btnSortNameD.Visible = false;

            dataGridStudents.RowHeadersVisible = false;

            dataGridStudents2.RowHeadersVisible = false;

        }

        private void btnSubmit\_Click(object sender, EventArgs e)

        {

            if (Validation())

            {

                Student obj = new Student();

                string fName = txtFirstName.Text;

                string lName = txtLastName.Text;

                obj.Name = fName + " " + lName;

                obj.Address = txtAddress.Text;

                obj.Email = txtEmail.Text;

                obj.BirthDate = dtpDOB.Value.Date;

                obj.ContactNo = txtPhone.Text;

                obj.Gender = cbGender.SelectedItem.ToString();

                obj.RDate = dtpRDate.Value.Date;

                obj.Course = cbCourse.SelectedItem.ToString();

                obj.Status = cbStatus.SelectedItem.ToString();

                lblsort1.Visible = true;

                lblsort2.Visible = true;

                btnSortName.Visible = true;

                btnSortRegDate.Visible = true;

                obj.Add(obj);

                Clear();

                BindGrid();

            }

        }

        private void btnClear\_Click(object sender, EventArgs e)

        {

            Clear();

        }

        private void btnEdit\_Click(object sender, EventArgs e)

        {

            Student obj = new Student();

            obj.Id = int.Parse(txtID.Text);

            string firstName = txtFirstName.Text;

            string lastName = txtLastName.Text;

            obj.Name = firstName + " " + lastName;

            obj.Address = txtAddress.Text;

            obj.Email = txtEmail.Text;

            obj.BirthDate = dtpDOB.Value.Date;

            obj.ContactNo = txtPhone.Text;

            obj.Gender = cbGender.SelectedItem.ToString();

            obj.RDate = dtpRDate.Value.Date;

            obj.Course = cbCourse.SelectedItem.ToString();

            obj.Status = cbStatus.SelectedItem.ToString();

            obj.Edit(obj);

            BindGrid();

            Clear();

            btnSubmit.Visible = true;

            btnClear.Visible = true;

            txtID.Enabled = false;

            btnEdit.Visible = false;

            btnDelete.Visible = false;

            btnNew.Visible = false;

            dtpRDate.Enabled = true;

            lblsort1.Visible = true;

            lblsort2.Visible = true;

            btnSortName.Visible = true;

            btnSortRegDate.Visible = true;

            cbStatus.Enabled = false;

        }

      private void dataGridStudents\_CellClick(object sender,DataGridViewCellEventArgs e)

        {

            try

            {

                string value = dataGridStudents[0, e.RowIndex].Value.ToString();

                if (String.IsNullOrEmpty(value))

                {

                    btnSubmit.Visible = true;

                    btnClear.Visible = true;

                    btnEdit.Visible = false;

                    btnDelete.Visible = false;

                    Clear();

                }

                else

                {

                    btnSubmit.Visible = false;

                    btnClear.Visible = false;

                    txtID.Enabled = true;

                    btnEdit.Visible = true;

                    btnDelete.Visible = true;

                    btnNew.Visible = true;

                    dtpRDate.Enabled = true;

                    cbStatus.Enabled = true;

                    int indexRow;

                    indexRow = e.RowIndex;

                    DataGridViewRow row = dataGridStudents.Rows[indexRow];

                    List<String> tempName = (row.Cells[1].Value.ToString()).Split()

.ToList();

                    txtFirstName.Text = tempName[0];

                    txtLastName.Text = tempName[1];

                    txtID.Text = row.Cells[0].Value.ToString();

                    txtAddress.Text = row.Cells[2].Value.ToString();

                    txtEmail.Text = row.Cells[3].Value.ToString();

                    dtpDOB.Value = DateTime.Parse(row.Cells[4].Value.ToString());

                    txtPhone.Text = row.Cells[5].Value.ToString();

                    cbGender.Text = row.Cells[6].Value.ToString();

                    dtpRDate.Value = DateTime.Parse(row.Cells[7].Value.ToString());

                    cbCourse.Text = row.Cells[8].Value.ToString();

                }

            }

            catch

            {

                MessageBox.Show("Try using the sort button insted of this!");

            }

        }

        private void btnDelete\_Click(object sender, EventArgs e)

        {

            Student obj = new Student();

            obj.Id = int.Parse(txtID.Text);

            string firstName = txtFirstName.Text;

            string lastName = txtLastName.Text;

            string name = firstName + " " + lastName;

            string word = "Do you want to Delete the Record of " + name + "?";

            string title = "Are you sure that you want to Delete the Record?";

            MessageBoxButtons btn = MessageBoxButtons.OKCancel;

            DialogResult outcome = MessageBox.Show(word, title, btn);

            if (outcome == DialogResult.OK)

            {

                obj.Delete(obj);

                BindGrid();

                Clear();

            }

        }

        private void btnNew\_Click(object sender, EventArgs e)

        {

            Clear();

            btnSubmit.Visible = true;

            btnClear.Visible = true;

            txtID.Enabled = false;

            dtpRDate.Enabled = false;

            btnEdit.Visible = false;

            btnDelete.Visible = false;

            btnNew.Visible = false;

        }

        private void btnSortName\_Click(object sender, EventArgs e)

        {

            Student obj = new Student();

            List<Student> student\_list = obj.List();

            int total\_Students = student\_list.Count();

            int right, left;

            for (right = 1; right < total\_Students; right++)

            {

                Student right\_student = student\_list[right];

                left = right - 1;

                while (left >= 0 && string

.Compare(student\_list[left].Name, right\_student.Name) > 0)

                {

                    student\_list[left + 1] = student\_list[left];

                    left = left - 1;

                }

                student\_list[left + 1] = right\_student;

            }

            DataTable dt = Utility.ConvertToDataTable(student\_list);

            dataGridStudents.DataSource = dt;

            btnSortName.Visible = false;

            btnSortName.Enabled = false;

            btnSortNameD.Visible = true;

            btnSortNameD.Enabled = true;

        }

        private void btnSortRegDate\_Click(object sender, EventArgs e)

        {

            Student obj = new Student();

            List<Student> student\_list = obj.List();

            int total\_Students = student\_list.Count();

            int right, left;

            for (right = 1; right < total\_Students; right++)

            {

                Student right\_student = student\_list[right];

                left = right - 1;

                while (left >= 0 && DateTime

.Compare(student\_list[left].RDate, right\_student.RDate) > 0)

                {

                    student\_list[left + 1] = student\_list[left];

                    left = left - 1;

                }

                student\_list[left + 1] = right\_student;

            }

            DataTable dt = Utility.ConvertToDataTable(student\_list);

            dataGridStudents.DataSource = dt;

            btnSortRegDate.Enabled = false;

            btnSortRegDate.Visible = false;

            btnSortRegDateD.Enabled = true;

            btnSortRegDateD.Visible = true;

        }

        private void btnSortNameD\_Click(object sender, EventArgs e)

        {

            Student obj = new Student();

            List<Student> student\_list = obj.List();

            int total\_Students = student\_list.Count();

            int right, left;

            for (right = 1; right < total\_Students; right++)

            {

                Student right\_student = student\_list[right];

                left = right - 1;

                while (left >= 0 && string

.Compare(student\_list[left].Name, right\_student.Name) < 0)

                {

                    student\_list[left + 1] = student\_list[left];

                    left = left - 1;

                }

                student\_list[left + 1] = right\_student;

            }

            DataTable dt = Utility.ConvertToDataTable(student\_list);

            dataGridStudents.DataSource = dt;

            btnSortName.Visible = true;

            btnSortName.Enabled = true;

            btnSortNameD.Visible = false;

            btnSortNameD.Enabled = false;

        }

        private void btnSortRegDateD\_Click(object sender, EventArgs e)

        {

            Student obj = new Student();

            List<Student> student\_list = obj.List();

            int total\_Students = student\_list.Count();

            int right, left;

            for (right = 1; right < total\_Students; right++)

            {

                Student right\_student = student\_list[right];

                left = right - 1;

                while (left >= 0 && DateTime

.Compare(student\_list[left].RDate, right\_student.RDate) < 0)

                {

                    student\_list[left + 1] = student\_list[left];

                    left = left - 1;

                }

                student\_list[left + 1] = right\_student;

            }

            DataTable dt = Utility.ConvertToDataTable(student\_list);

            dataGridStudents.DataSource = dt;

            btnSortRegDate.Enabled = true;

            btnSortRegDate.Visible = true;

            btnSortRegDateD.Enabled = false;

            btnSortRegDateD.Visible = false;

        }

        private void weeklyDateTime\_ValueChanged(object sender, EventArgs e)

        {

            DateTime start = weeklyDateTime.Value.Date;

            DateTime end = start.AddDays(7);

            Student obj = new Student();

            List<Student> listStudents = obj.List();

            try

            {

                List<Student> stu = listStudents.Where(x => x.RDate >= start)

.Where(x => x.RDate <= end).ToList();

                DataTable tableData = Utility.ConvertToDataTable(stu);

                dataGridStudents2.DataSource = tableData;

                var weeklyEnrolled = listStudents

                    .Where(reg => reg.RDate >= start && reg.RDate <= end)

                    .GroupBy(course => course.Course)

                    .Select(x => new

                    {

                        Program = x.First().Course,

                        Count = x.Count().ToString()

                    }).ToList();

                DataTable dt = Utility.ConvertToDataTable(weeklyEnrolled);

                dgWeekly.DataSource = dt;

                foreach (DataGridViewColumn c in dgWeekly.Columns)

                {

                    c.DefaultCellStyle.Font = new Font("Arial", 22F, GraphicsUnit.Pixel);

                }

                dgWeekly.Columns[0].Width = 400;

                dgWeekly.Columns[1].Width = 150;

                dataGridStudents2.Columns[0].Width = 40;

                dataGridStudents2.Columns[1].Width = 100;

                dataGridStudents2.Columns[2].Width = 80;

                dataGridStudents2.Columns[3].Width = 150;

                dataGridStudents2.Columns[4].Width = 80;

                dataGridStudents2.Columns[5].Width = 90;

                dataGridStudents2.Columns[6].Width = 78;

                dataGridStudents2.Columns[7].Width = 73;

                dataGridStudents2.Columns[8].Width = 70;

                dataGridStudents2.Columns[9].Width = 85;

            }

            catch

            {

                MessageBox.Show("No Data to view!");

            }

        }

        private void BindChart(List<Student> lst)

        {

            try

            {

                var result = lst

               .GroupBy(l => l.Course)

               .Select(cl => new

               {

                   Course = cl.First().Course,

                   count = cl.Count().ToString()

               }).ToList();

                DataTable dt = Utility.ConvertToDataTable(result);

                chart.DataSource = dt;

                chart.Name = "Course";

                chart.Series["Series1"].XValueMember = "Course";

                chart.Series["Series1"].YValueMembers = "Count";

                chart.Series["Series1"].IsValueShownAsLabel = true;

                chart.Series["Series1"].Font = new Font("Arial",30F,GraphicsUnit.Pixel);

            }

            catch

            {

                MessageBox.Show("No Data to view!");

            }

        }

        private void BindGrid()

        {

            Student obj = new Student();

            List<Student> listStudents = obj.List();

            DataTable dt = Utility.ConvertToDataTable(listStudents);

            dataGridStudents.DataSource = dt;

            dataGridStudents.Columns[0].Width = 40;

            dataGridStudents.Columns[1].Width = 100;

            dataGridStudents.Columns[2].Width = 80;

            dataGridStudents.Columns[3].Width = 150;

            dataGridStudents.Columns[4].Width = 80;

            dataGridStudents.Columns[5].Width = 90;

            dataGridStudents.Columns[6].Width = 78;

            dataGridStudents.Columns[7].Width = 73;

            dataGridStudents.Columns[8].Width = 70;

            dataGridStudents.Columns[9].Width = 85;

            BindChart(listStudents);

        }

        private void Clear()

        {

            txtFirstName.Text = String.Empty;

            txtLastName.Text = String.Empty;

            txtAddress.Text = String.Empty;

            txtEmail.Text = String.Empty;

            txtPhone.Text = String.Empty;

            txtID.Text = String.Empty;

            cbGender.SelectedItem = null;

            lblsort1.Visible = true;

            lblsort2.Visible = true;

            btnSortName.Visible = true;

            btnSortName.Enabled = true;

            btnSortRegDate.Visible = true;

            btnSortNameD.Visible = false;

            btnSortNameD.Enabled = false;

            btnSortRegDateD.Enabled = false;

            btnSortRegDateD.Visible = false;

            cbGender.SelectedText = "--select--";

            dtpDOB.Value = DateTime.Parse("1/1/2000");

            dtpRDate.Value = DateTime.Now;

            cbStatus.Enabled = false;

        }

        private Boolean Validation()

        {

            if (txtFirstName.Text != "")

            {

                if (txtLastName.Text != "")

                {

                    if (txtAddress.Text != "")

                    {

                        if (txtEmail.Text != "")

                        {

                            if (txtPhone.Text != "")

                            {

                                if (cbGender.SelectedItem != null)

                                {

                                    if (cbCourse.SelectedItem != null)

                                    {

                                        return true;

                                    }

                                    else

                                    {

                                    MessageBox.Show("Select Gender", "No gender ERROR");

                                    }

                                }

                                else

                                {

                                    MessageBox.Show("Select Gender", "No gender ERROR");

                                }

                            }

                            else

                            {

                            MessageBox.Show("No Phone Number", "No phone number ERROR");

                            }

                        }

                        else

                        {

                           MessageBox.Show("Please enter your Email", "No email ERROR");

                        }

                    }

                    else

                    {

                       MessageBox.Show("Please enter your address", "No address ERROR");

                    }

                }

                else

                {

                    MessageBox.Show("Please enter your last name", "No name ERROR");

                }

            }

            else

            {

                MessageBox.Show("Please enter your first name", "No name ERROR");

            }

            return false;

        }

    }

}