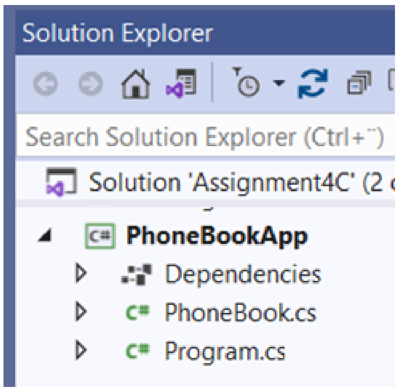
1. Part 1 - PhoneBook App

Create a new project, select Console App, change the solution and the project name.



1.1 Print the contents of the arrays to the console window

All you need to do is to run a for-statement from index 0 to the arrays number of elements -1. Print out a text formatted med name and phone number using Console.WriteLine. Here is how you can format a line of text:

```
string strOut = string.Format("{0,-15} {1,-15}", names[index], phones[index]);
```

1.2 Sort array.

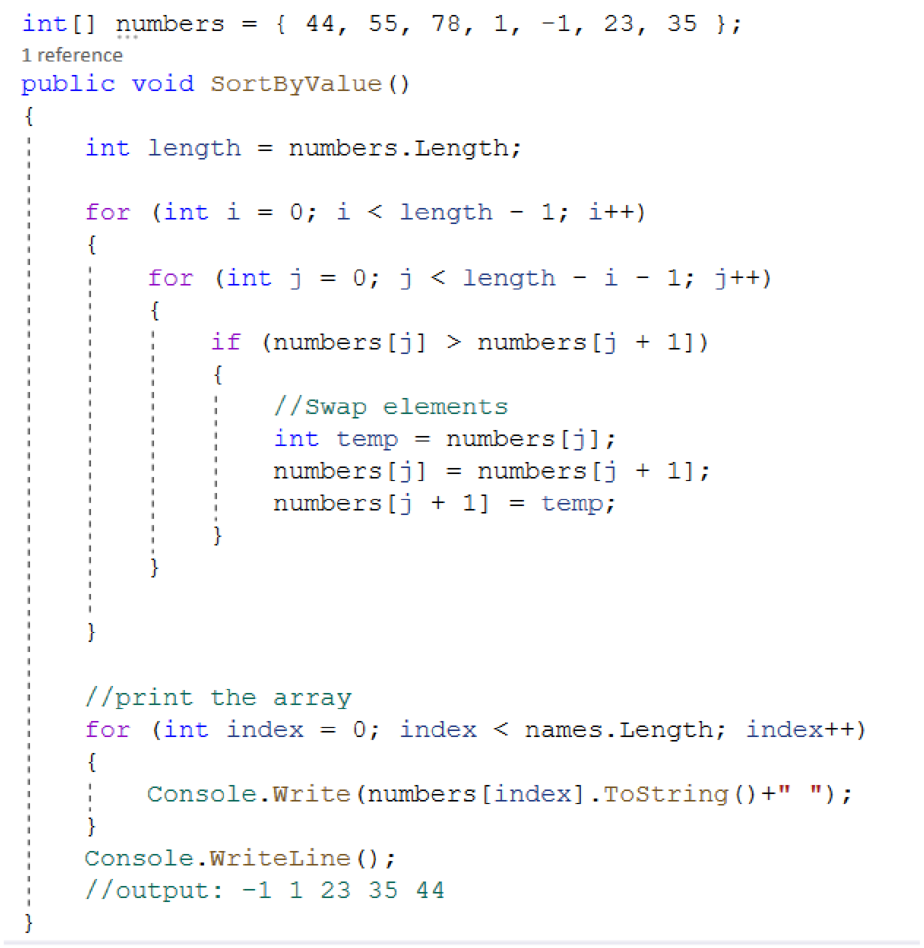
A simple way of sorting an array is using a bubble sort. Using this algorithm, two adjacent element are compared repeatedly and they are swapped if they are not in the intended order. Bubble sort is not an effective sorting algorithm but it works fine with small arrays or when the performance of an application is not important.

See a detailed example on: https://www.programiz.com/dsa/bubble-sort

```
public void SortByName()
{
    int pos, i;
    int length = names.Length;

    for (pos = 0; pos < length - 1; pos++)
    {
        // Last i elements are already in place
        for (i = 0; i < length - pos - 1; i++)
        {
            int result = names[i].CompareTo(names[i + 1]);

            if (result == 1)  // less, then swap the valus
            {
                SwapValues(i);
            }
        }
    }
}
```

```
private void SwapValues(int pos)
{
    string temp = names[pos];
    names[pos] = names[pos + 1];
    names[pos + 1] = temp;

    //Swap also the corresponding phones
    temp = phones[pos];
    phones[pos] = phones[pos + 1];
    phones[pos + 1] = temp;
}
```

Here is another example of bubble sort using an array of integers:

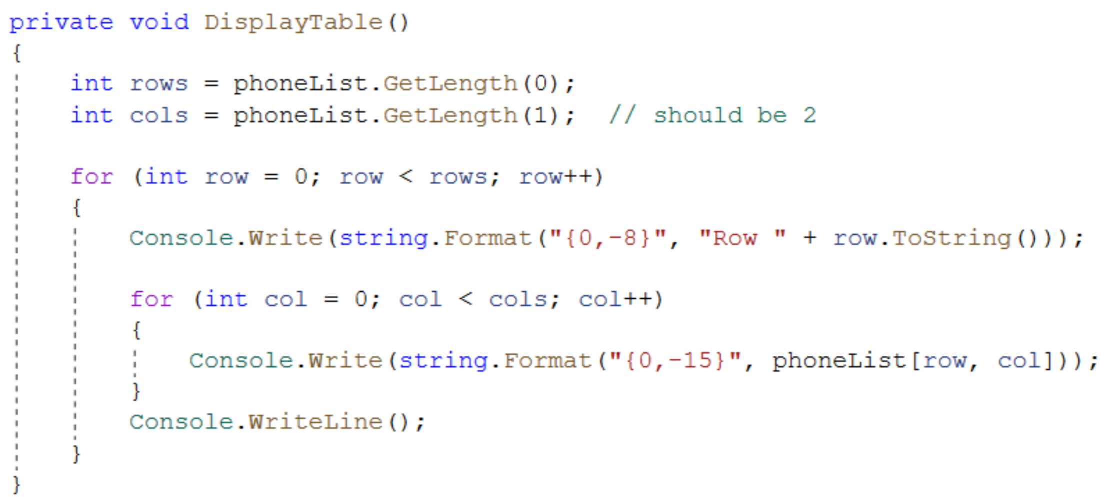


```
int[] numbers = { 44, 55, 78, 1, -1, 23, 35 };
1 reference
public void SortByValue()
{
    int length = numbers.Length;

    for (int i = 0; i < length - 1; i++)
    {
        for (int j = 0; j < length - i - 1; j++)
        {
            if (numbers[j] > numbers[j + 1])
            {
                //Swap elements
                int temp = numbers[j];
                numbers[j] = numbers[j + 1];
                numbers[j + 1] = temp;
            }
        }

    }

    //print the array
    for (int index = 0; index < names.Length; index++)
    {
        Console.Write(numbers[index].ToString()+" ");
    }
    Console.WriteLine();
    //output: -1 1 23 35 44
}
```

Test the above method by calling it from the Main method in the Program.cs.

1.3 Print the two dimensional array

The number of elements in a row and a column can be determined udinh the array's GetLenth (row nr). What value does the Length property of an array result in? Try it using phoneList.Length.

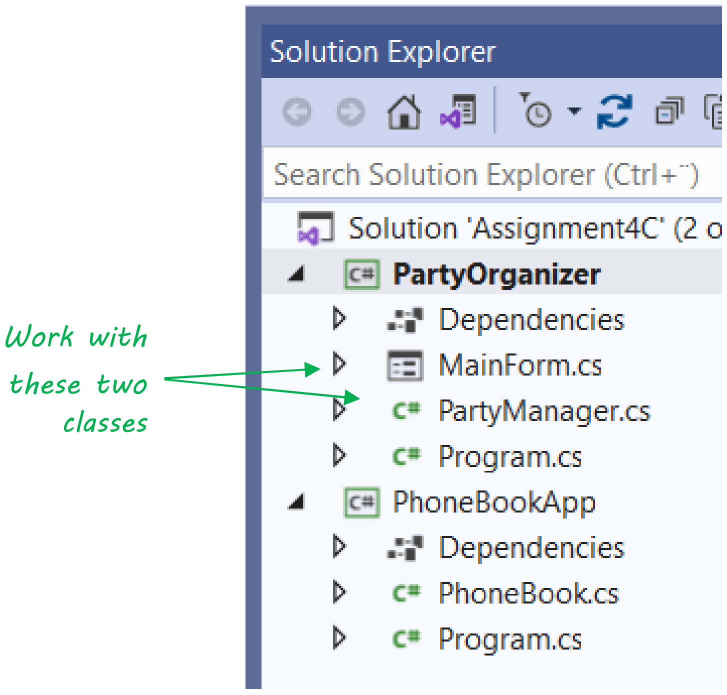


```
private void DisplayTable()
{
    int rows = phoneList.GetLength(0);
    int cols = phoneList.GetLength(1);  // should be 2

    for (int row = 0; row < rows; row++)
    {
        Console.Write(string.Format("{0,-8}", "Row " + row.ToString()));

        for (int col = 0; col < cols; col++)
        {
            Console.Write(string.Format("{0,-15}", phoneList[row, col]));
        }
        Console.WriteLine();
    }
}
```

2. Part 2 : Party Organizer

2.1 Create a new project

Create a project in the same solution or close VS and create a new project. Select Windows Forms Application (without the .NET Framework) from the template, choose a name for your solution and brows to a project directory. To create a new project in the same solution, right click on the Solution (e.g. Assignment 4C) and select *Add New Project*. Then right click on the project e.g. PartyOrganizer and select *Set as start up project*.



Rename Form1 to **MainForm**. Draw your GUI as in the assignment description or the way you want it. In addition to **MainForm** which is responsible for handling input/output, you will need to create another class for performing the task of party-organization. Let's call this class **PartyManager** and save it in the file PartyManager.cs.

2.2 The PartyManager class

The purpose of this class is to handle all data and tasks about parties and other events.

Fields: **guestList**: An array of strings for storing names of guests.

costPerPerson: A double (or decimal) to store cost per guest.

feePerPerson: A double (or decimal) to store amount to be paid by each person.

Constructor: Write a constructor with one parameter, a whole number that can be used to dimension the array (max number of guests). Each event usually has a maximum number of guests which can be 5, 20 or 1000. The array should have space (elements) to the size. The size is a input from the user.

Properties: Write a property with read and write access to each instance variable (except the **guestsList**). DO NOT write a property for this array as it is an object and arrays are passed by reference. Giving access to an array is risky as changes in the other object will affect the original array. Write a method to returns one element a time (see the code on the next page).

Methods: Write methods for adding a new guest (guest name) in the array and another method for deleting an existing name (marking the element as null or empty) at a given position in the array. Before adding, format the name so the last name comes first (LUNDSTRÖM, Nisse).

Write a methods that returns the value saved in the array in a given position (index).

Write a method that finds the first vacant place in the array.

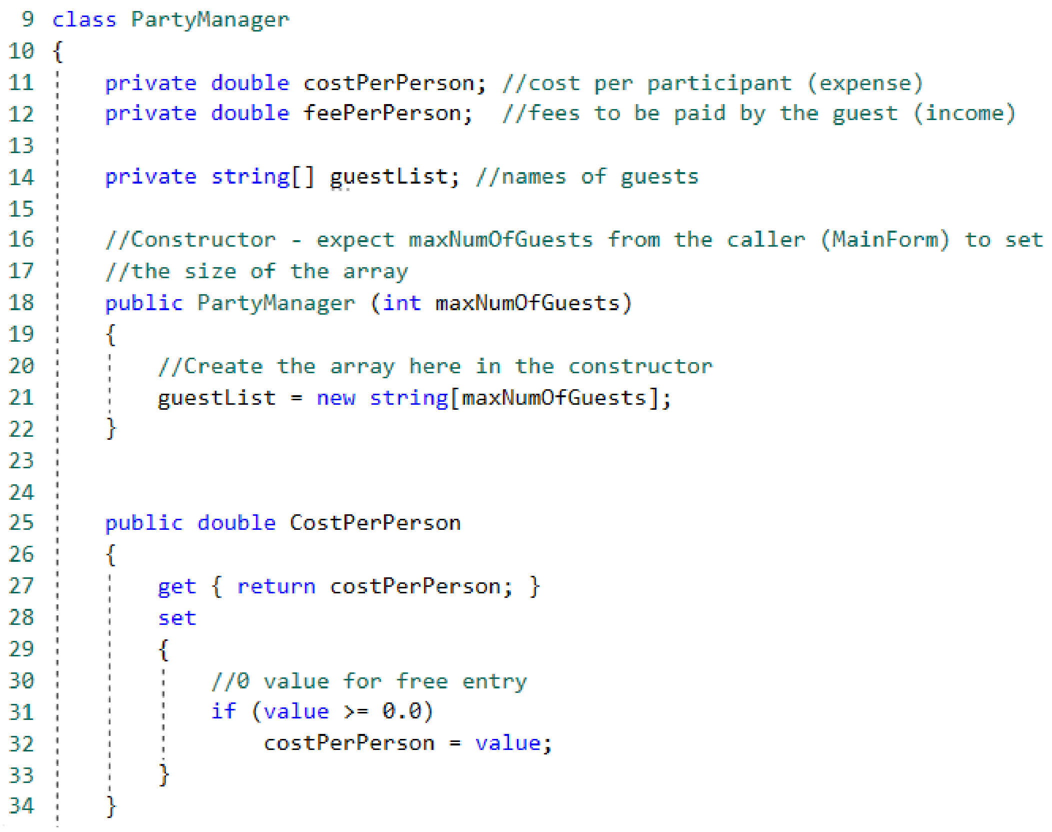
Write a method that returns an array of string containing names of the registered guests.

Write a method that checks a given index so it is not out of range (as explained in the assignment document).

Write other methods if necessary.

Fields, constructor and property

The following code snippet shows a part of the class.



```
class PartyManager
{
    private double costPerPerson; //cost per participant (expense)
    private double feePerPerson;  //fees to be paid by the guest (income)

    private string[] guestList; //names of guests

    //Constructor - expect maxNumOfGuests from the caller (MainForm) to set
    //the size of the array
    public PartyManager (int maxNumOfGuests)
    {
        //Create the array here in the constructor
        guestList = new string[maxNumOfGuests];
    }


    public double CostPerPerson
    {
        get { return costPerPerson; }
        set
        {
            //0 value for free entry
            if (value >= 0.0)
                costPerPerson = value;
        }
    }
```

The constructor is called by the MainForm passing a value that is to be used to set the capacity of the array. The array is created in the constructor (Line 21)

- The array is declared on line 14.
- The constructor begins on line 18.
- Notice how the array is created in the constructor on line 21.
- A property connected to the instance variable costPerPerson.
- Write a property for the feePerPerson in the same way.

Counting number of elements with values (guests):

It is also important to keep track of the number of guests saved in the array. Each time, a new person is added, the number should increment by one and each time an existing element is removed (name cleared), the number should decrement by one. You may use an extra field, **numOfGuests,** in the class but make sure you always keep the value updated. A better alternative is to write a method that counts the number. Loop through the array and count the number of elements that are not empty (or null), as in the code below:

```
private int NumOfGuests ( )
{
    int numGuests = 0;

    for (int i = 0; i < guestList.Length; i++)
    {
        if (!string.IsNullOrEmpty ( guestList[i] ))
        {
            numGuests++;
        }
    }
    return numGuests;
}
```

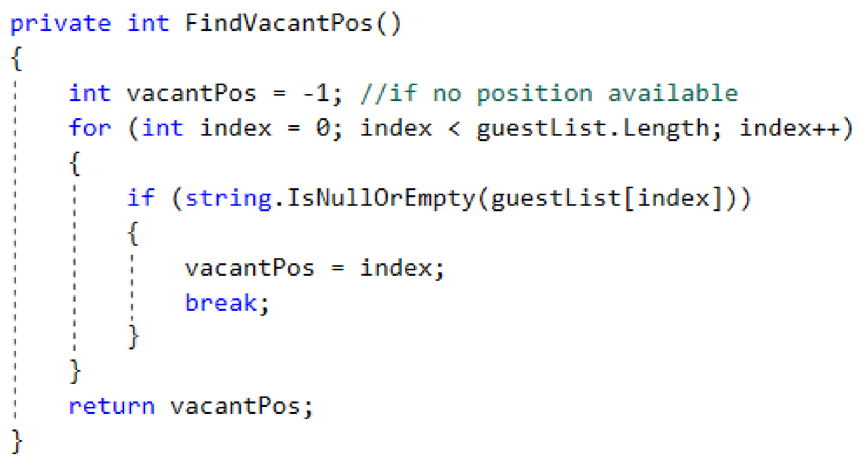
Note:

In this assignment, we are using an array of strings, but if you happen to have an array of other types, for example doubles, then you should apply other criteria to mark empty elements (using -1,0, or a very big number, or a small number). The types int and double have a defined max and min value, int.MaxValue, int MinValue, which can be used. An array of bool values can be set to false to mark empty elements. Remember by empty element, we mean an unused element in the array.

Finding the first empty position:

The method returns -1 signaling that no vacant positon is found; **the array is full**. The caller of this method must check the return value so it is not -1.

Algorithms:



```
private int FindVacantPos()
{
    int vacantPos = -1; //if no position available
    for (int index = 0; index < guestList.Length; index++)
    {
        if (string.IsNullOrEmpty(guestList[index]))
        {
            vacantPos = index;
            break;
        }
    }
    return vacantPos;
}
```

- Start a loop through the array from 0 to Length -1
- At a position **index**:
    - If the element is null or empty (string), return the **index** to the element and exit the loop
- If the loop is done and no empty element is found, return -1

Pay a special attention to the code on line 58 (above figure), where the loop is exited as soon as the first empty element is found. The statement break causes the iteration to exit in advance.

Note: The property **guestList.Length** returns the number of elements that the array is created for. It is the capacity of the array. The index to the last element in the array becomes **guestList.Length -1** and therefore the condition index < guestList.Length is used. Using a condition **with index '<='** throws an "index out of range" exception and causes an abnormal termination of the application.

Add new guest:

To save a new guest in the array, you need to get the first name and the last name from the **MainForm**. The algorithm for adding a new person is:

- Find the first vacant position in the array; call the **FindVacnatPos** method.
- If the position number is not -1,
    - Put together the first name and the last name and save the resulting string in that position.

- Else exit the method and return false.

```
public bool AddNewGuest(string firstName, string lastName)
{
    bool ok = true;
    int vacantPos = FindVacantPos();

    if (vacantPos != -1)
    {
        guestList[vacantPos] = FullName(firstName, lastName);
    }
    else
        ok = false;

    return ok;
}

private string FullName(string firstName, string lastName)
{
    return lastName.ToUpper() + ", " + firstName;
}
```

Deleting a guest from the array

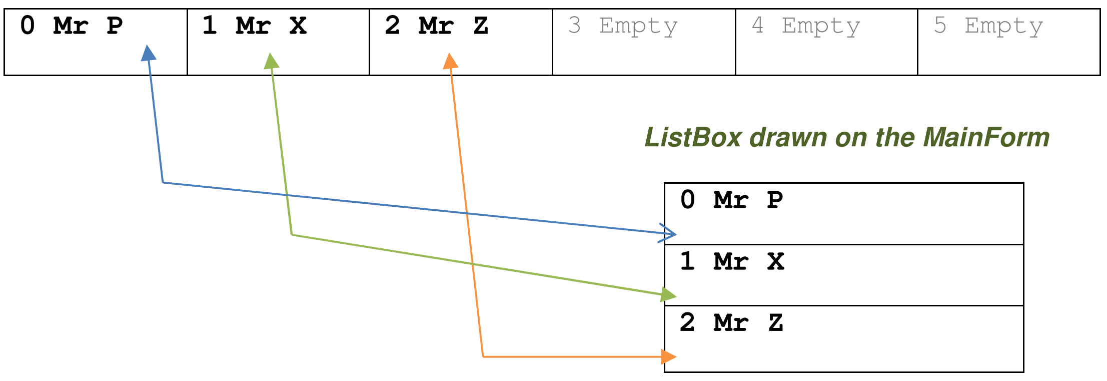
To "delete" an item:

guestList [index] = string.Empty;

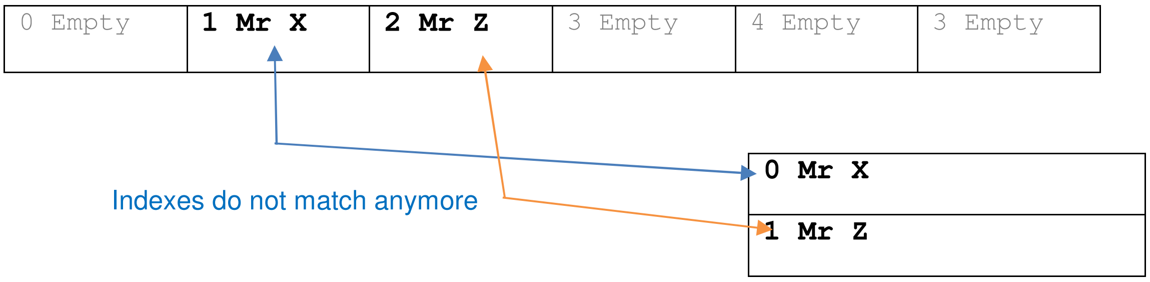
guestList [index] = null; //only string and objects can bet set to null, not int, double, etc.

Then call **UpdateGUI** in the **MainForm** to update the listbox, displaying only registered guests, i.e. array elements not being Empty (or null). See the MainForm class later to learn how the UpdateGUI works. However, here we are creating a problem.

guestList *in the Party class (0, 1, etc. = index)*



The array we are using in this assignment is a static array, (not meaning that is declared as static). Static here implies that our array has a fixed size. The size does not grow and nor it will shrink. If want to delete a guest in a certain position, we simple mark that position being unused. Now delete item 0 and update the listBox. Item number 1 is display as item no 0 in the listBox. Now, let's delete item 0 and update the listBox. Item number 1 in the array is displayed as item no 0 in the listBox, and item no 1 in the array is displayed as item number 1 in the listbox.



If you now try to delete item no 0 from the listBox again, the index is 0 and your code tries to set the element in the position 0 in the array to Empty (or null)! Moreover, if you somewhere in your code try to fetch the data for index 0, your program even crashes, if the element is set to null! If you don't understand this scenario, just try it; run your program, add two guests, then delete the first one, remaining one item. Delete this one too, and see what happens!

Fixes: There are different ways to fix this problem.

1. Map the list box to the non-empty positions in the array, so that the first element that is not empty in the array corresponds to the first index in the listBox – a little but challenging work required!

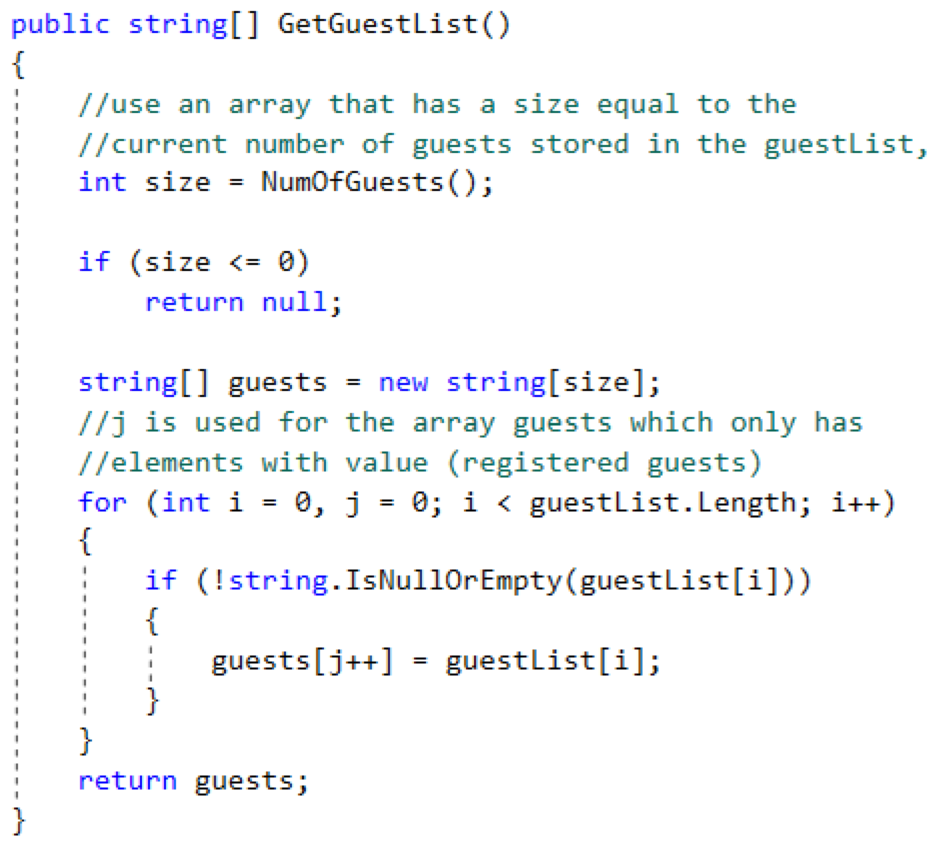
2. Shift the elements on the right side of the element being deleted one step to the left – much easier and a good optimization.

```
public bool DeleteAt(int index)
{
    bool ok = false;
    if (CheckIndex(index))
    {
        guestList[index] = string.Empty;
        MoveElementsOneStepToLeft(index);
        ok = true;
    }
    return ok;
}
private void MoveElementsOneStepToLeft(int index)
{
    for (int i = index + 1; i < guestList.Length; i++)
    {
        guestList[i - 1] = guestList[i];  //move 1 step to left
        guestList[i] = string.Empty;  //empty its place
    }
}
```

Returning an array of registered guests:

Algorithm:

- Count the number of registered guests.
- Declare and create a local array of strings with this size.
- Start a loop through the array from 0 to Length -1.
- At a position **index**:
    - If the element is NOT null or empty (string)
        - Save the string from the **guestList** to this local array
        - Increment the index of the local array
- When the loop is complete, return the local array.



```
public string[] GetGuestList()
{
    //use an array that has a size equal to the
    //current number of guests stored in the guestList,
    int size = NumOfGuests();

    if (size <= 0)
        return null;

    string[] guests = new string[size];
    //j is used for the array guests which only has
    //elements with value (registered guests)
    for (int i = 0, j = 0; i < guestList.Length; i++)
    {
        if (!string.IsNullOrEmpty(guestList[i]))
        {
            guests[j++] = guestList[i];
        }
    }
    return guests;
}
```

Notes:

- **NumOfGuests** is the method explained earlier.
- **guestList.Length** has the maximum number of elements **guests.Length** gives the number of guests registered (1 as the first guest is added, 2 when the second guest is added and so on).

Quiz:

Q1: How does **`j++`** work, what happens?

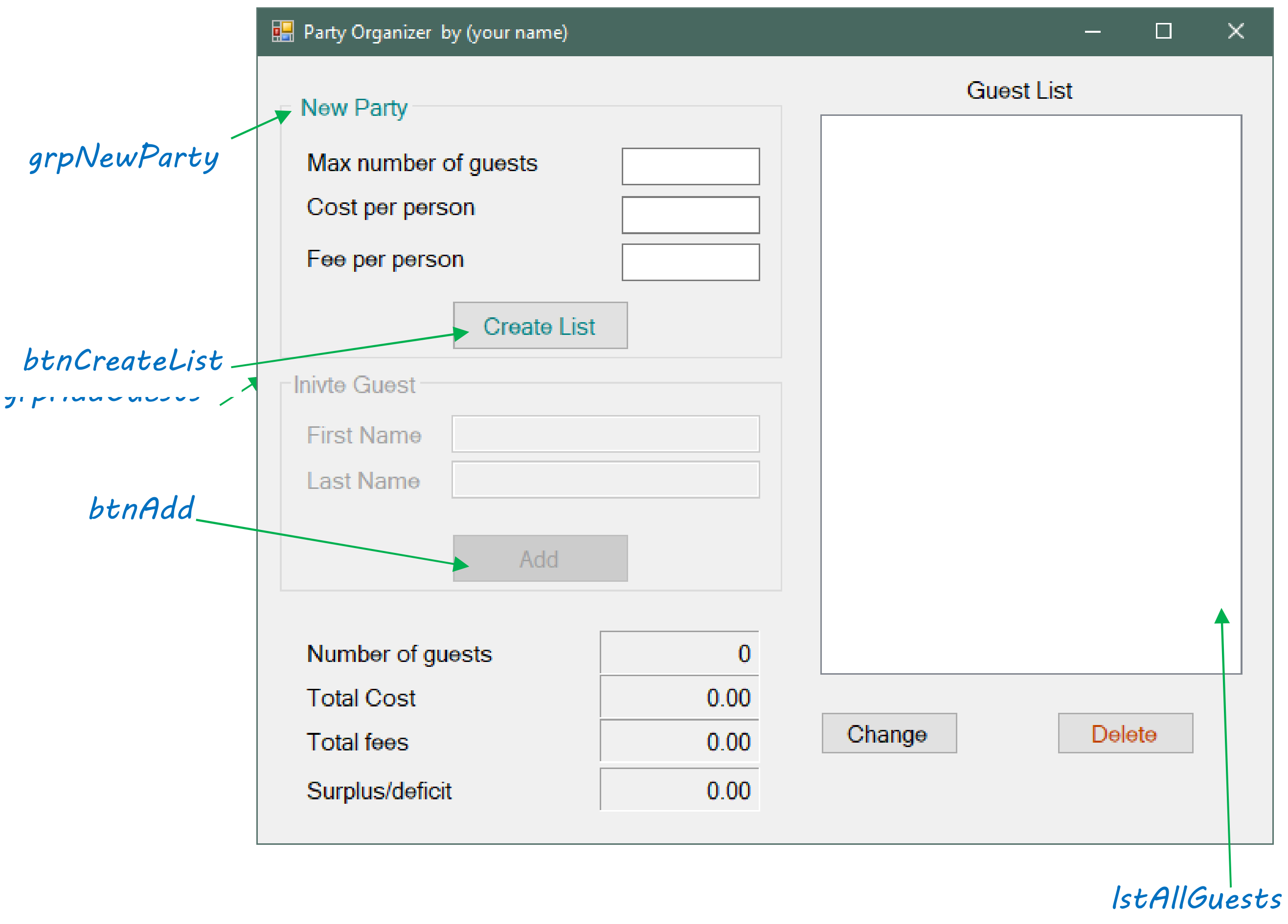
Q2: Could you use **`guests[j]`** if you bring some changes, if yes which changes?

If you are not sure about your answers, discuss the questions in the forum.

2.3 The MainForm Class

Only and only MainForm should work with the user interface via its controls (the components it contains).

Give your GUI-controls suitable names. You can follow the pattern here, using a three letter prefix. Begin TextBox names with txt, Labels with lbl, buttons with btn and GroupBoxes with grp.



Number of guests 0
Total Cost 0.00
Total fees 0.00
Surplus/deficit 0.00

MainForm declares an object of the PartyManager class on Line 15, but the object is created on this line nor in the constructor (Lines 17 to 21). The reason is that the Party class requires an integer in its constructor (as described earlier).

```
13 public partial class MainForm : Form
14 {
15     PartyManager partyManger; //ref. variable declared, object not created
16
17     public MainForm ( )
18     {
19         InitializeComponent ( );
20         InitializeGUI ( );
21     }
```

Remember the constructor is called when you create an instance of a class (using new). In this application, we let the user decide the size of the array (list of guests) which we will then passed to the PartyManager class' constructor. This is to be done in the method connected to the button Create List.

InitializeGUI – use this method to prepare MainForm:

```
23     private void InitializeGUI()
24     {
25         //Clear input controls
26         ClearInputControls();
27         ClearOutputControls();
28
29         //Disable the dd guest groupbox
30         grpAddGuests.Enabled = false;
31
32         //Enable the create new party groupbox.
33         grpNewParty.Enabled = true;
34     }
```

2.1 Methods called from above (Lines 26-27):

```
private void ClearInputControls()
{
    txtMaxNum.Text = string.Empty;
    txtCost.Text = string.Empty;

    continue with other controls
}
```

```
private void ClearOutputControls()
{
    //Clear output controls
    //can be sett to string.Empty also
    lblNumGuests.Text = "0";
    lblTotalCost.Text = "0.00";
    lblTotalFees.Text = "0.00";
    lblProfit.Text = "0.00";
    lstAllGuests.Items.Clear();
}
```

Double-click on the Create List button and complete the method as in the code snippet below.

```
107  private void btnCreateParty_Click (object sender, EventArgs e)
108  {
109      //The party must have been created successfully before
110      //reading the cost and fee per person! or doing any other work
111
112      bool maxNumOK = CreateParty ();  //Total number of guests
113      if (!maxNumOK)
114          return;  //do no more
115
116      bool amountOK = ReadCostPerPerson () && ReadFeePerPerson ();
117
118      if (maxNumOK && amountOK)
119      {
120          grpAddGuests.Enabled = true;
121          UpdateGUI ( );
122      }
123  }
```

CreateParty( ) creates the object **partyManager** declared as an instance variable at the top of the MainForm class (code on the preceding page). The methods called on Line 78. The code is given below.

```
private bool CreateParty ( )
{
    int maxNumber = 0;
    bool ok = true;

    if (int.TryParse ( txtMaxNum.Text, out maxNumber ) && (maxNumber > 0))
    {
        partyManger = new PartyManager ( maxNumber );
        MessageBox.Show ( $"Party list with space for {maxNumber}  guests created!", "Success" );
    }
    else
    {
        MessageBox.Show ( "Inavlid Total Number. Please try again!", "Error" );
        ok = false;
    }
    return ok;
}
```

ReadCostPerPerson and **ReadFeePerPerson** (Line 116): You should be able to complete these methods by yourself. Refer to your previous assignment.

UpdateGUI( ), called on Line 121, is a method that updates output data on the form. This method can be called whenever some output is recalculated. You should have written the method **GetGuestList**() in the **partyManager** object which returns an array of strings where each string contains data for a registered guest.

```
130 private void UpdateGUI()
131 {
132     lstAllGuests.Items.Clear();
133     string[] guestList = partyManger.GetGuestList();
134      if (guestList != null)
135     {
136         for (int i = 0; i < guestList.Length; i++)
137         {
138             string str = $"{i + 1,4} {guestList[i],-20}";
139             lstAllGuests.Items.Add(str);
140         }
141     }
142    // else
143        // return;  //no return here for the update to continue for the lasted deleted item.
144
145     //The following code must always be carried out
146     //even if guestList is empty!
147     double totalCost = partyManger.CalcTotalCost();
148     lblTotalCost.Text = totalCost.ToString("0.00");
149     continue with the rest
156
```

The loop on line 132 formats every item from the array, adding a counter start from 1 (i+1, i is counted from 0). The $-format is an alternative to string.Format (is easier). The parentheses { } contain the format {value, width} where *width* is the number of chars within witch the value is to be adjusted.

Double-click on the **btnAdd** button and complete the method and write code to add a new guest by calling the appropriate method in the **PartyManager** class that adds a new guest. You may have to send the first name and last name as parameter-values to the method.

Hint: It can happen that the use presses the space bar by mistake before or after writing a text. If you would like to clear text from the spaces at the beginning and the end of a string, you can use the string-variable's Trim methods as in the code below.

```
        txtFirstName.Text = txtFirstName.Text.Trim ( );
        txtLastName.Text = txtLastName.Text.Trim ( );
```

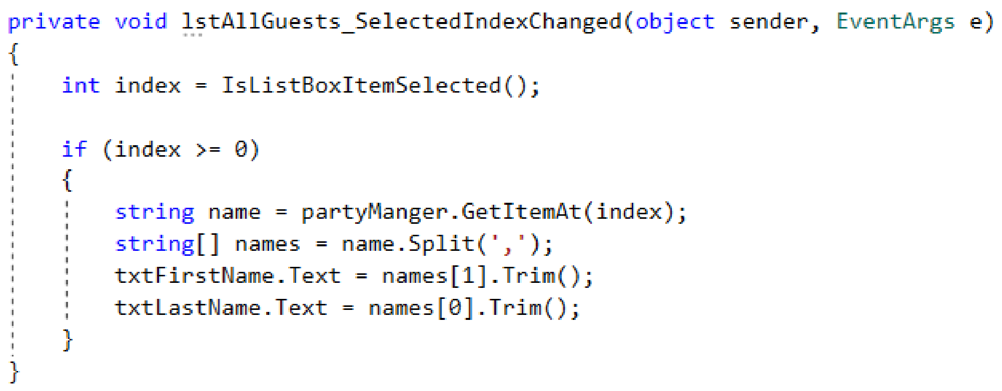
The above method can now be called for both first name and last name or any other string to validate:

```
bool fstNameOK = ValidateText(txtFirstName.Text, "First name must be given!");
bool lstNameOK = ValidateText(txtLastName.Text, "Last name must be given!");
```

Hint: To validate a string so it is not null or empty, you can write a method that can work for any string:

```
private bool ValidateText (string text, string errMessage)
{
    text = text.Trim ( );

    if (string.IsNullOrEmpty ( text ))
    {
        MessageBox.Show ( errMessage, "Error" );
        return false;
    }
    return true;
}
```

Bonus: Double-click on the list box in VS (lstAllGuests) and copy the following code:

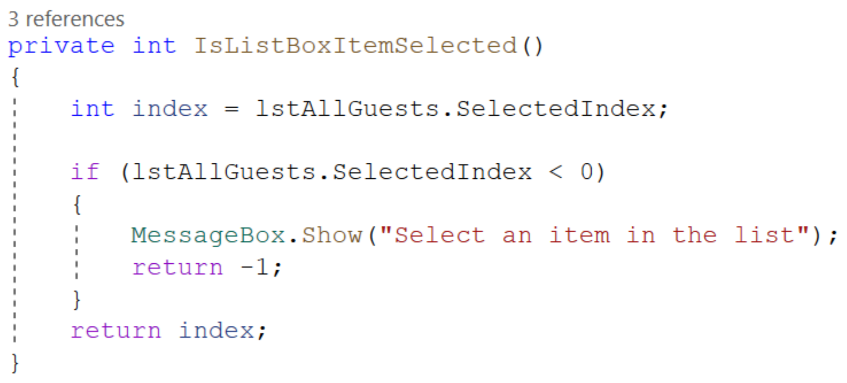


```
private void lstAllGuests_SelectedIndexChanged(object sender, EventArgs e)
{
    int index = IsListBoxItemSelected();

    if (index >= 0)
    {
        string name = partyManger.GetItemAt(index);
        string[] names = name.Split(',');
        txtFirstName.Text = names[1].Trim();
        txtLastName.Text = names[0].Trim();
    }
}
```

Note: You can substitute the following line instead of calling the method IsListBoxItemSelected:

```
int index = lstAllGuests.SelectedIndex;
```

Otherwise: here is the body of the method:



```
3 references
private int IsListBoxItemSelected()
{
    int index = lstAllGuests.SelectedIndex;

    if (lstAllGuests.SelectedIndex < 0)
    {
        MessageBox.Show("Select an item in the list");
        return -1;
    }
    return index;
}
```

An advice to remember from the above: *“Do not write the same code twice. Make a method of the code and call it twice or as many times as needed.”*

Well, if you still face difficulties, don’t hesitate to ask questions in the forum. Nothing is easy unless you learn how to do it!

Good Luck!

Programming is fun. Never give up. Ask for help!

Farid Naisan,
Course Responsible and Instructor