CIT Coursework Documentation

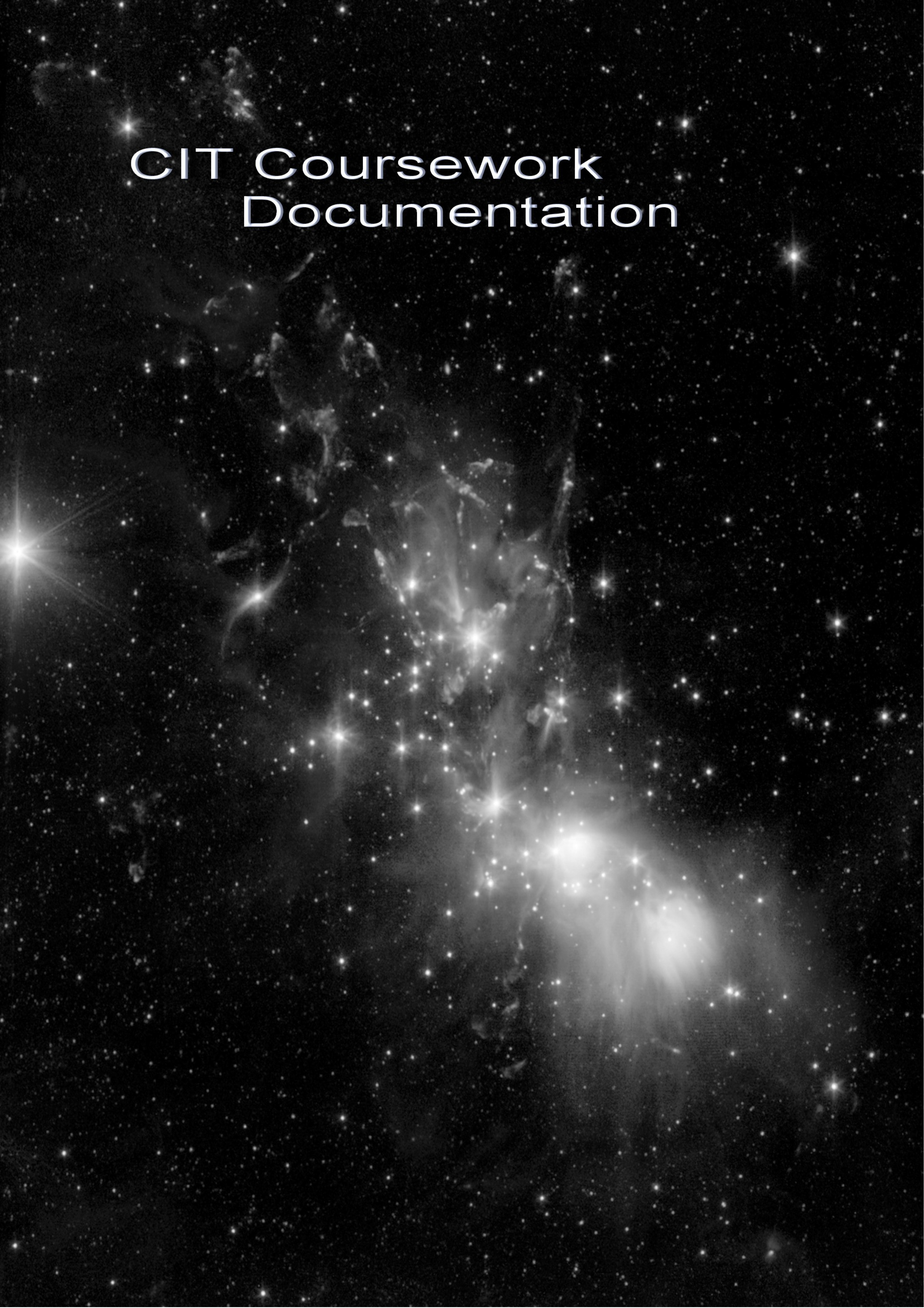


Table of Content

...

Acknowledgement

MSDN2 Library
http://msdn2.microsoft.com/en-us/library/ms123401(en-us,MSDN.10).aspx

Wikipedia
http://en.wikipedia.org/wiki/Main_Page

Chapter 1 : Problem Definition and Project Objective

This documentation is a structured elaboration on my CIT coursework project and the corresponding computer program. In other words, it provides information on the program development life cycle of my project, and the algorithms implemented by my computer program. In this chapter, I would give an overview to the problem, and describe the objective of this project.

1.1 Description of the Problem

I am required to write a computer program for the school annual dinner registration. During the data collection stage, personal information of the participants has to be input into the program, for example name of participant, year of graduation, sex, age, employment, number of seats required, etc. Moreover, the program should validate all input data and have functions to amend the input data.

At the end of the registration, the program is required to generates a seating plan of the anniversary dinner in a text file. I should define the seat allocation rules clearly and any other system parameters such as table sizes. And consider at least two allocation rules at the same time to generate a seating plan.

1.2 Definition of the Problem

I define the problem as to write a computer program to do the following tasks :

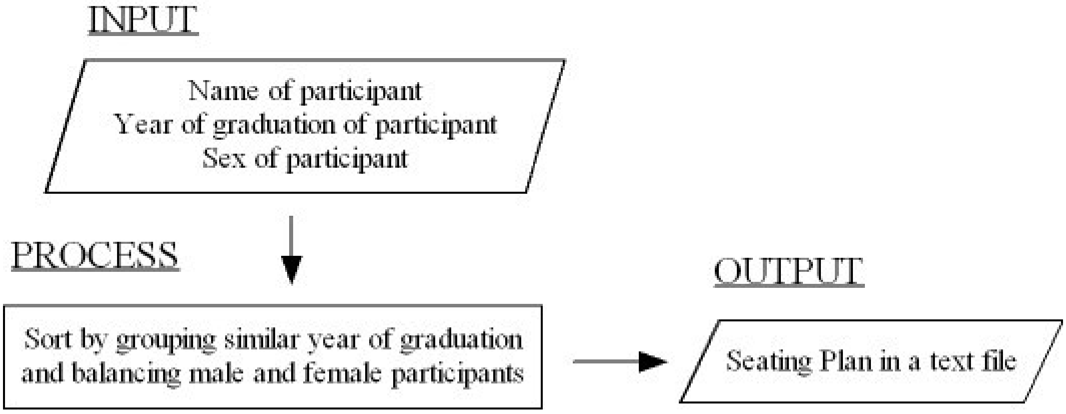
1. Retrieve name, year of graduation and sex data of participants
2. Validate name, year of graduation and sex data of participants
3. Allocate participants to seats under user-defined table size, in addition consider the following two allocation rules at the same time : - Grouping participants of similar year of graduation in each table - Balancing male and female participants in each table
4. Generate a seating plan in a text file

- The name of participant is a string or characters that is unique and not null.
- The year of graduation of participant is a whole number ranging from 1000 to 3000, which means a specified year from 1000 AD to 3000 AD. This year range is enough for most usage.
- The sex of participant is a character 'm' or 'f' , or a word 'male' or 'female', indicating male and female sexuality respectively.
- A seating plan is a sorted list of information. It indicates the exact participants sit in each particular table. And it allocate seats to participants following the two defined allocation rules. The seating plan is in the form of a text file.

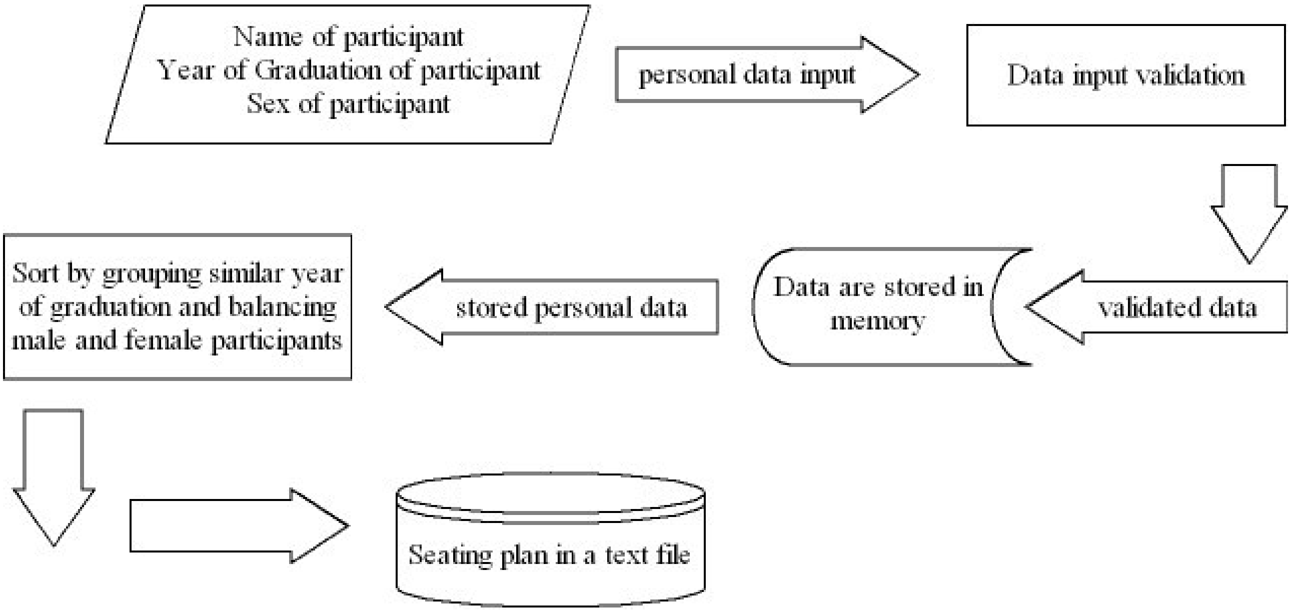
1.3 Division of the Problem

There are multiple solutions to this problem. The yielded solution depends on variable factors such as personal information collected and the seat allocation rule oriented. Moreover it is not easy to determine the best solution among all the possible solutions. At first glance, the algorithm can be very complex. It would be better to break down the problem into manageable pieces.

To simplify the seemingly complex situation, I outline the input, process and output of the program as in below :


INPUT
Name of participant
Year of graduation of participant
Sex of participant
PROCESS
OUTPUT
Sort by grouping similar year of graduation and balancing male and female participants
Seating Plan in a text file

The IPO chart shows that the input are three fields of data. They are name, year of graduation, and sex of a participant. Then the data is passed into the sorting algorithm for processing. At last information is output in the form of a seating plan. It can be seen that the whole problem is all about data manipulation. More importantly, data are the links between the sub-problems. The diagram below shows further division of the problem into sub-problems and the links between them :


Name of participant
Year of Graduation of participant
Sex of participant
personal data input
Data input validation
validated data
Data are stored in memory
stored personal data
Sort by grouping similar year of graduation and balancing male and female participants
Seating plan in a text file

To sum up, the problem can be divided into sub-problems of data input, data validation, data storage, sorting using stored data, creating and appending the seating plan into a text file.

1.4 Key Concepts involved in the Problem

I have identified three key concepts involved in this problem. I think that these concepts deserves my special attention. They are the keys to a accessible, powerful, and distinguished algorithm solution to this problem.

User Interface

A user interface in computer programming is a layer that exists between the user and the computer program. It is a layer where interactions between the user and the computer program take place. A good user interface can increase usability of a computer program. In other words, without a good user interface, no matter how much or how well can the algorithm achieve, the user simply cannot, or at least difficult to access the functionalities provided by the computer program. Therefore, I consider it as the most important key concept involved in this problem.

Data Structure

In computer programming, a data structure is the way the computer program stores data. It is a key concept involved in this problem because as illustrated before, this program is all about data manipulation and data are the links between the sub-problems. The way the data stored would affect how the data is manipulated and how the sub-problems are linked together. This in turn directly affects how the steps are designed to solve the problem.

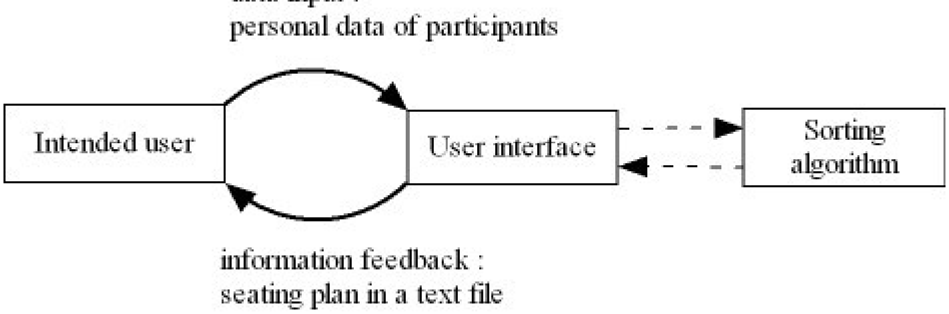
Multiple Solutions

There is no perfect solution to this problem. For a perfect solution, it must be able to give a seating plan that satisfy both the two defined allocation rules. Nevertheless for a set of participants, there may exist an arrangement in which more participants of similar year age can be grouped together, meanwhile there exist another arrangement in which male and female participants are more balanced in each table. Provided that the two arrangements are mutual exclusive, there exist no perfect solution because each arrangement best satisfies one defined allocation rule only. By realizing this fact, the pursuit of a perfect solution is unwise. Instead I should approach the problem by providing multiple solutions, each best satisfies one defined allocation rule.

1.5 Objective of the Project

I write this computer program for the school annual dinner registration. It is written for users who have the source document, containing complete raw data of all participants attending the school annual dinner. A complete record of a participant includes the name, year of graduation, and sex for each of them. The raw data are collected in registration.

The only goal of this computer program is to satisfy the intended users. As viewed from the perspective of the intended users, the processing in the algorithm is of little importance. Their main concerns are how the raw data are being input, what information are being output, and in what form the information feedback are presented to them.



Regarding the concerns of the intended users, they may require the program to provide some features or functions for them. The following points would identify and explain these expected requirements from them.

Easy to Learn

The user interface should display concepts, words and phrases familiar to users. The user interface design should be simple and well-organized. So that at first contact users can immediately know how to use the program and access functionalities, even with little or no instruction given. Furthermore users can quickly familiarize with the user interface and well recognize the user interface each time he/she uses the program.

Efficient in Control

The user interface should be designed in a way that users can input participants' data and make commands to the program with minimal keystrokes or mouse clicks. So that users can control the program in a smoother manner and with less effort.

Easy to Use

The program should display concise instruction whenever needed so that inexperienced users can have a better control over the program. Also the program should display system status so that users are informed about what is going on and know what to do next.

Effective in Error Handling

The program should be carefully designed to prevent a problem from occurring in the first place. Error messages should be expressed in plain language, precisely indicate the problem, and constructively suggest a solution. Furthermore there should be a confirmation before users commit any critical actions.

Stable Execution

The program should reduce run-time error as much as possible. So that during execution, it will not stop responding to users and the operating system. Furthermore it saves users' time if program can perform fast. Sometimes a program works very slowly when processing a significant amount of data.

Reliable Output

The program should generate an organized seating plan that is easy to read, and then save it in a location that is easy to find. More importantly during processing there must not be any data corruption.

In this project, what I want to do is to devise an algorithm to solve the problems as defined at the beginning of this chapter. In the process, I would pay extra attention to satisfy the possible requirements from the intended users.

In this project, the computer program is directed to a simple, easy-to-use application. I am not expecting my program to be very powerful. I only hope that every intended users would find it comfortable to work along with my program.

Chapter 2 : Selection of IT Tools and Approaches

To solve a problem by computer, a programming language is needed to tell a computer what to do stepwise. The programming language chosen directly affects the design and implementation of an algorithm. In this chapter, I would comment on the programming language that I used, as well as the alternative ways to approach the problem.

2.1 The Programming Language

Visual Basic (VB) has been the most popular high-level programming language. This programming language makes programming much easier and is well-known to amateur programmers.

In this project I used **Visual Basic 2005 Express Edition** in development process. Visual Basic 2005 is the latest version in the Microsoft Visual Basic.NET family. It is integrated with the Visual Studio 2005 development environment.

Advantages

The following box list some of the advantages of this programming language :

- Interactive and customizable development environment
- Enable graphically designed windows application
- Easy to learn
- Able to create powerful application
- Simplified debugging
- Easy to deploy
- Visual Basic 2005 Express Edition is free

There is hardly another programming language that can contain all the above advantages as Visual Basic do. The most prominent advantage of Visual Basic is its simple syntax and semantics rules. The programming codes much resembles English language, both in words and grammar. So the program codes are extremely readable. And inexperienced programmers can learn and recall the language very easily. Also Because Visual Basic is so popular, there are many books and online resources that assist Visual Basic programmers. The integrated development environment (IDE) is helpful too. It provides automatic code formatting, code completion, error detection, debugging, and many more other functions. The help system, MSDN Library with information around 300MB, provide a comprehensive reference to the concepts and language in Visual Basic. The .NET Framework makes Windows services more accessible, and provides many pre-built functions that alleviate programmers' work. This can help programmers to concentrate on the algorithm design rather than repetitive routines.

This simple programming language is powerful at the same time. Visual Basic 2005 Express Edition supports programmers to design graphical user interfaces by drag and drop. Also Visual Basic 2005 is built on the .NET Framework. It is a range of concepts and technologies that can make Visual Basic applications much more powerful than before.

Disadvantages

Different programming languages are good at and weak in building different kinds of applications. However for the objective of this project, I can hardly identify any disadvantages of using Visual Basic 2005 Express Edition.

Visual Basic does not suit for really sophisticated programs. And it is a slower programming language. But for this project, which is directed to write a simple, easy-to-use computer program, Visual Basic is definitely the most appropriate programming language to use. Since the computer program would not be sophisticated, and would not involve complicated and lengthy computation.

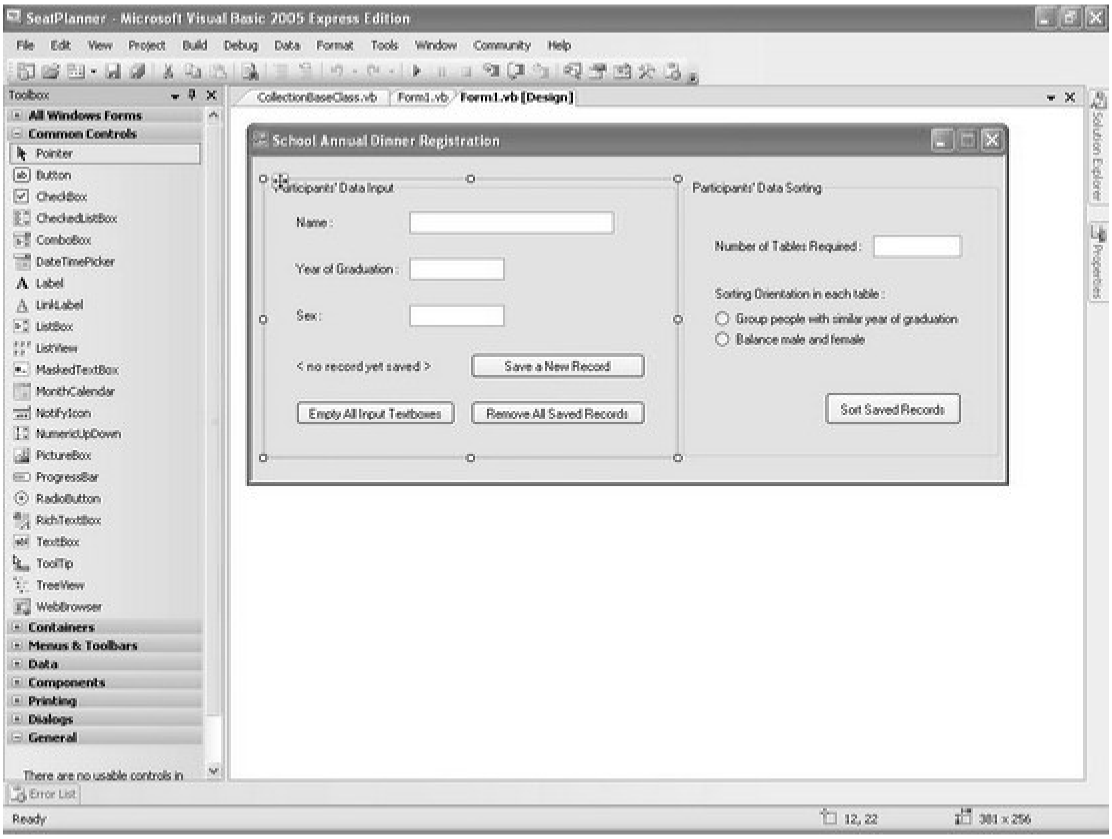
Comparison against other Programming Languages

There are hundreds of programming language in the world. So this I am not intended to compare to all the advantages and disadvantages among them. Instead, I want to highlight the advantages of Visual Basic by comparing it to other languages. In the following, I compare Visual Basic against Pascal and Visual C++ :

	Pascal	Visual Basic	Visual C++
programmers	mostly students	mostly amateurs	mostly professionals
learning difficulty	easy	medium	difficult
programming difficulty	medium	easy	difficult
programming ability	fairly low	powerful	very powerful

All the descriptions and the above comparison show that, Visual Basic is targeted as being a simple but powerful programming language. So I employ Visual Basic 2005 Express Edition as the programming language of this project undoubtedly.

VB 2005 Express Edition – Integrated Development Environment :



2.2 Approaches to the Problem

After choosing the appropriate IT tool, I should consider ways to approach the problem to reach a real algorithm. In the following paragraphs I would compare various approaches to three problems in this project, provided that they are significant and demand careful consideration.

User Interface

In computer programing there are many types of user interfaces. There are three of them which are most commonly used.

Command-line user interface : it is a user interface where the program may displays prompts to users, then users input command string with the computer keyboard, then the program provide output by printing text on the computer screen.

Graphical user interface : it is a user interface where the program accept input such as keystrokes and mouse clicks on the screen, then provide graphical output on the computer screen

Web-based user interface : it is a user interface where web pages are created to accept input and provide output. The data are transmitted via the Internet and viewed by users using web browser.

Conclusion : Although command-line user interface demand less system memory, it is not favourable because it may require the users to remember commands, or manipulate the program by too many keystrokes. Although web-based user interface allows remote access of the program, it is not favourable because the Internet is sometimes unstable and perform slowly. In this case I choose to use graphical user interface because this type of user interface is user-friendly, can be easily created by using Visual Basic 2005 Express Edition.

Sorting Algorithm

A sorting algorithm must be involved in this problem. To select the right one for my program, I would compare three popular sorting algorithms.

Bubble sort : it is a simple sorting algorithm. It works by repeatedly stepping through the list to be sorted, comparing two items at a time, swapping these two items if they are in the wrong order. The pass through the list is repeated until no swaps are needed, which means the list is sorted.

Insertion sort : it is a simple sorting algorithm. It works by repeatedly taking the next item and inserting it into the final data structure in its proper order with respect to items already inserted.

Quick sort : it is an advanced sorting algorithm. It works by employing a divide and conquer strategy to divide a list into sub-lists, and then make use of 'pivots' to sort the list.

Conclusion : Although bubble sort is simple and easy to remember, it is not favourable because it is too inefficient. Although quick sort is efficient, it is not favourable because it is too advanced and difficult for me to implement. In this case I choose to use insertion sort because this type of sorting algorithm is still easy to implement, but is a lot more efficient than bubble sort.

Data Structure

There are two data structures that capture my attention. They are array and collection.

Array : An array is group of related variables, storing values of same data type, and can be accessed by using index

Collection : A collection is a group of objects that can be accessed by using index too, but with pre-built functions like add, insert, remove etc. for data manipulation.

Conclusion : Although collection takes more effort to deal with, I still choose to use it because it is more flexible than array. In conjunction with the use of insertion sort, it can simplify the algorithm. As the insertion actions are already a pre-built function of collections.

Understanding the strength and weakness of various approaches, I have chosen graphical user interface, insertion sort, and collection as the preferred approaches to the problem. In the next chapter, I would bring these ideas into real practice.

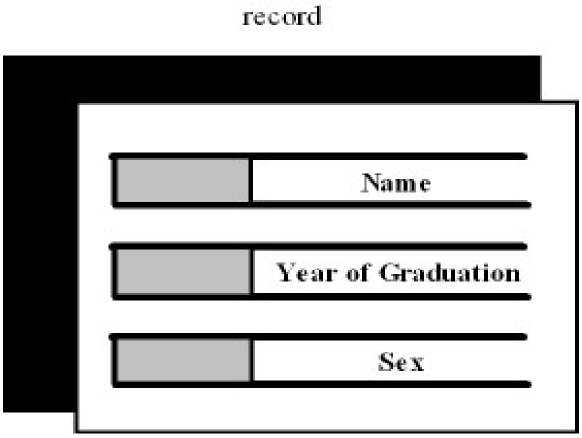
Chapter 3 : Algorithm Design and Implementation

It is recognized that every sub-problems in this project is about data manipulation. Therefore the algorithms must be data dependent. So to implement the algorithms smoothly, the algorithms must be working with organized data. To better organize the data, the concept of a record is introduced. The program also employs an object-oriented data structure.

Definition of a Record

A record is a set of validated data that contains the name, year of graduation, and the sex of one particular participants. So in the program each participant has one record of his/her own.

The concept of record is used throughout the algorithm. It is illustrated as in below :



3.1 Data Structure

In this program, entities, for example, individual records and tables, are represented by objects. An object is an instance of a class. The disadvantage of using objects is that they are quite difficult to be maintained at the beginning stage of program development. But the advantage is that as the algorithm develops , it can aid the reuse of codes and make program development much faster. Also I have implemented a special kind of object, collection objects, to group, organize, and access objects in the program.

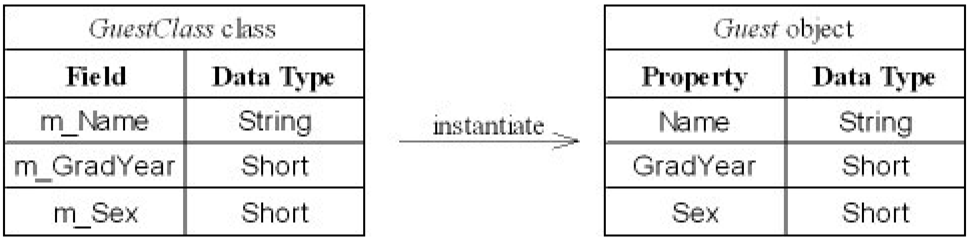
Objects

The GuestClass class and Guest objects

In the program, individaul records are represented by objects named *Guest*, each of them is an instance of the class *GuestClass*. (a 'guest' in the codings eqautes a 'record' in this documentation context)

```
Public Class GuestClass
 Private m_Name As String
 Private m_GradYear As Short
 Private m_Sex As Short

 Public Property Name() As String
  Get
   Return m_Name
  End Get
  Set(ByVal value As String)
   m_Name = value
  End Set
 End Property

 Public Property GradYear() As Short
  Get
   Return m_GradYear
  End Get
  Set(ByVal value As Short)
   m_GradYear = value
  End Set
 End Property

 Public Property Sex() As Short
  Get
   Return m_Sex
  End Get
  Set(ByVal value As Short)
   m_Sex = value
  End Set
 End Property
End Class
```

The record class *GuestClass* have three fields, *m_Name, m_GradYear, m_Sex.* They stores the *Name, GradYear, Sex* property of each *Guest* object.



GuestClass class	
Field	**Data Type**
m_Name	String
m_GradYear	Short
m_Sex	Short

Guest object	
Property	**Data Type**
Name	String
GradYear	Short
Sex	Short

Each *Guest* object, representing a record in the memory, is an instance of the class *GuestClass*. Each *Guest* object encapsulates three common properties : *Name, GradYear, Sex.* It represents that each record contains data of the name of participant, year of graduation of participant and sex of participant of a record respectively.

Collection Classes and Objects

In the program there is a certain number of collection objects. A collection object a an object that manages and groups other objects. Each particular object in a collection object can be accessed by zero-based indices. The advantage of such as implementation is that it favours the use of iterations in the sorting algorithm.Additionally a collection object provides pre-built functions such as insert an element, search an element etc., to make data manipulation be easier.

The CollectionBaseClass class inheritance

In the program, all collection classes defined is derived from the class *CollectionBaseClass*. The *CollectionBaseClass* defines some properties and methods to make data manipulation be easier. Note that it is derived from the class *System.Collections.CollectionBase*, which is a pre-built class in Visual Basic 2005.

```
Public Class CollectionBaseClass
  Inherits System.Collections.CollectionBase

  Default Public Property Item(ByVal index As Short) As Object
    Get
      Return List(index)
    End Get
    Set(ByVal value As Object)
      List.Item(index) = value
    End Set
  End Property

  Public Function Add(ByVal value As Object) As Short
    Return List.Add(value)
  End Function

  Public Sub Insert(ByVal index As Short, ByVal value As Object)
    List.Insert(index, value)
  End Sub

  Public Sub Remove(ByVal value As Object)
    List.Remove(value)
  End Sub
End Class
```

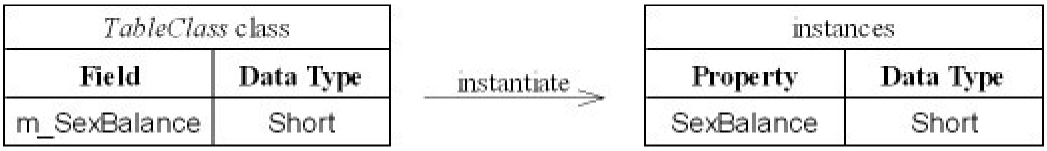
The *CollectionBaseClass* class defines functions that add objects to, insert objects to, and remove objects from a collection object. More importantly it defines a function so that I can use indices to refer to an object in a collection object.

The TableClass class and objects

In the program, tables are represented by instances of the class *TableClass*. The *TableClass* class is derived from the *CollectionBaseClass.* So that each instances of the *TableClass* class inherits all the data manipulation functions defined in the *CollectionBaseClass.*

```
Public Class TableClass
  Inherits SeatPlanner.CollectionBaseClass
  Private m_SexBalance As Short

  Public Property SexBalance() As Short
    Get
      Return m_SexBalance
    End Get
    Set(ByVal value As Short)
      m_SexBalance = value
    End Set
  End Property
End Class
```

The *TableClass* has a field, *m_SexBalance*, to store the *SexBalance* property of each instances of the *TableClass* class.



TableClass class	
Field	**Data Type**
m_SexBalance	Short

instances	
Property	**Data Type**
SexBalance	Short

Each instances of the class *TableClass* represents a table that participants would be allocated to. They stores a property *SexBalance.* This property indicates the difference of the number of male and female in a table.

The AllGuestsClass class and AllGuests object

In the program, the set of all records is represented by an collection object named *AllGuests.* This object is an instance of the class *AllGuestsClass.* And the *AllGuestsClass* is derived from the class *CollectionBaseClass.* So the *AllGuests* object inherits all the data manipulation functions defined in the *CollectionBaseClass.*

```
Public Class AllGuestsClass
  Inherits SeatPlanner.CollectionBaseClass
End Class
```

By putting all *Guest* object into the *AllGuests* object, they are organized and can be accessed by indices, in the form *AllGuests(index)*.

The AllTablesClass class and AllTables object

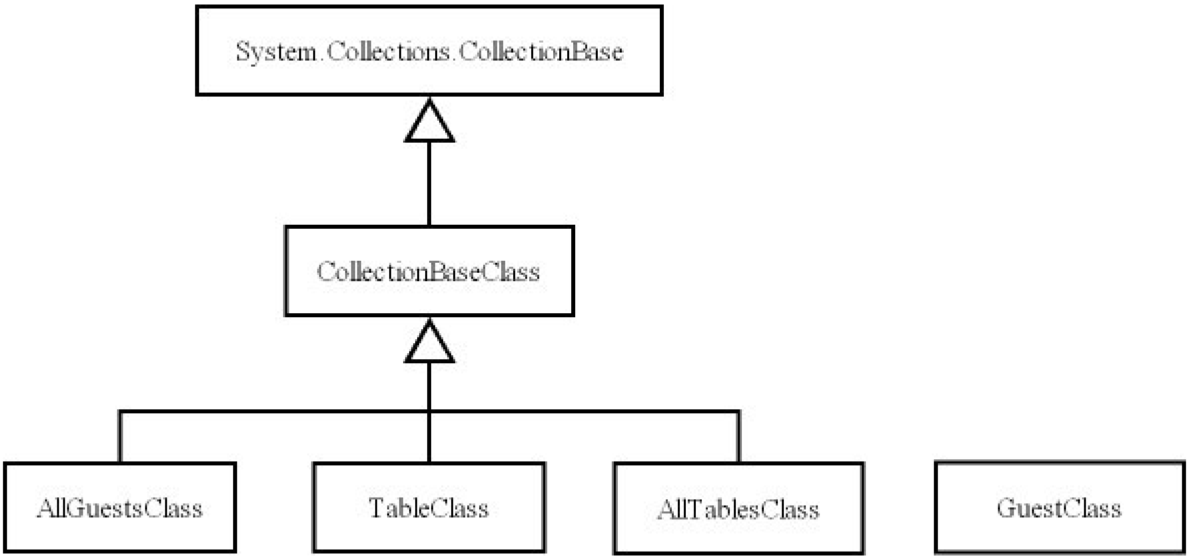
In the program, the set of all tables is represented by an collection object named *AllTables*. This object is an instance of the class *AllTablesClass*. And the *AllTablesClass* is derived from the class *CollectionBaseClass*. So the *AllTables* object inherits all the data manipulation functions defined in the *CollectionBaseClass*.

```
Public Class AllTablesClass
  Inherits SeatPlanner.CollectionBaseClass
End Class
```

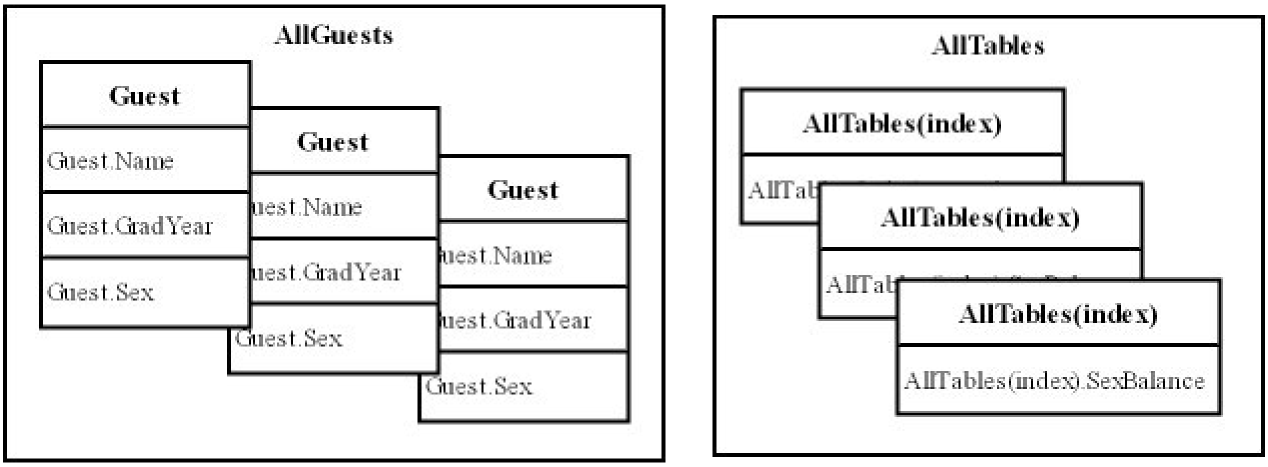
By putting all *Table* object into the *AllTables* object, they are organized and can be accessed by indices, in the form *AllTables(index)*.

Classes and Objects Data Structure

Classes



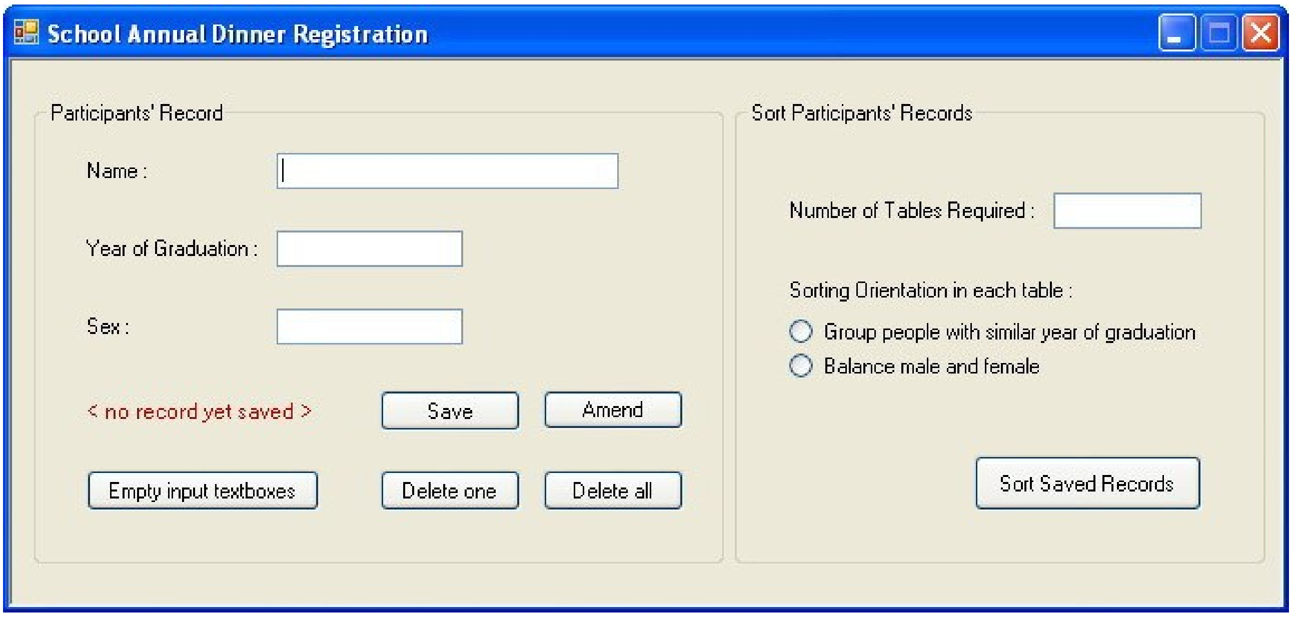
Objects



The algorithm design has a close relationship with the object-oriented data structure. It involves four major parts. They include the user interface, data validation and record saving, sorting, and text-file appending.

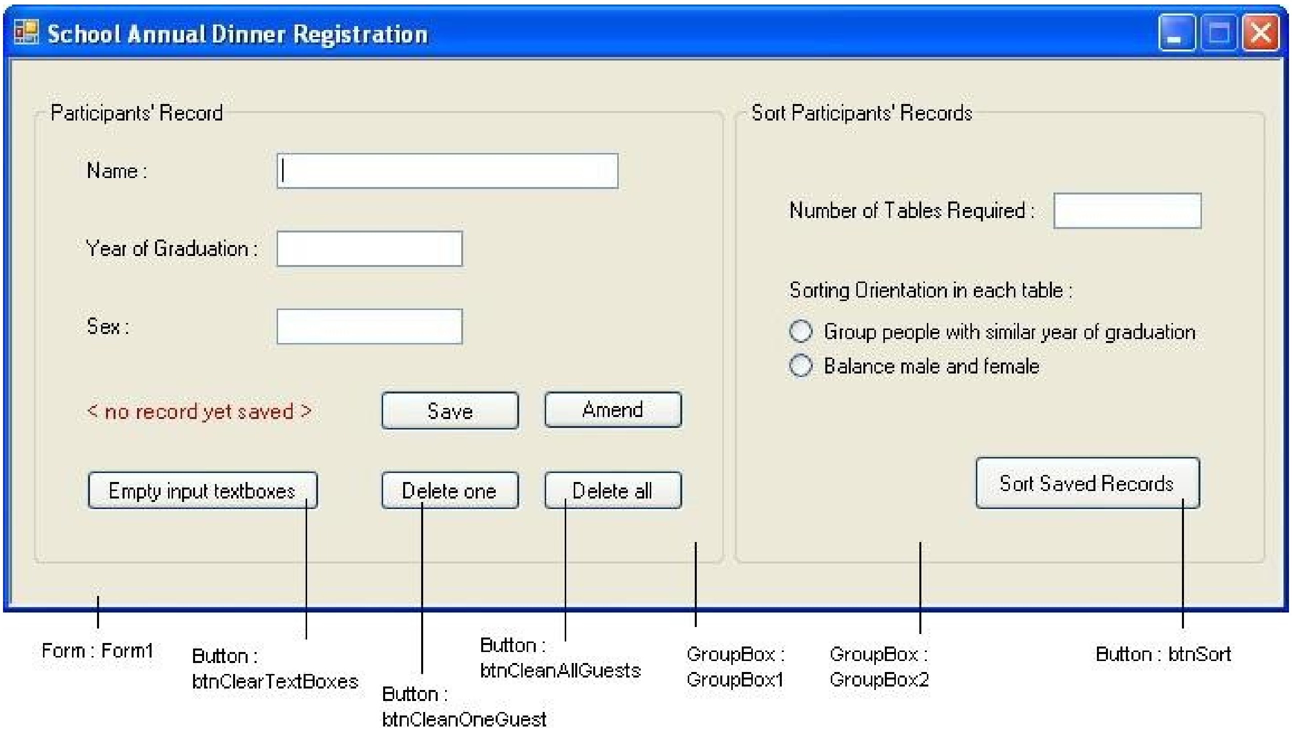
3.2 Design of the User Interface

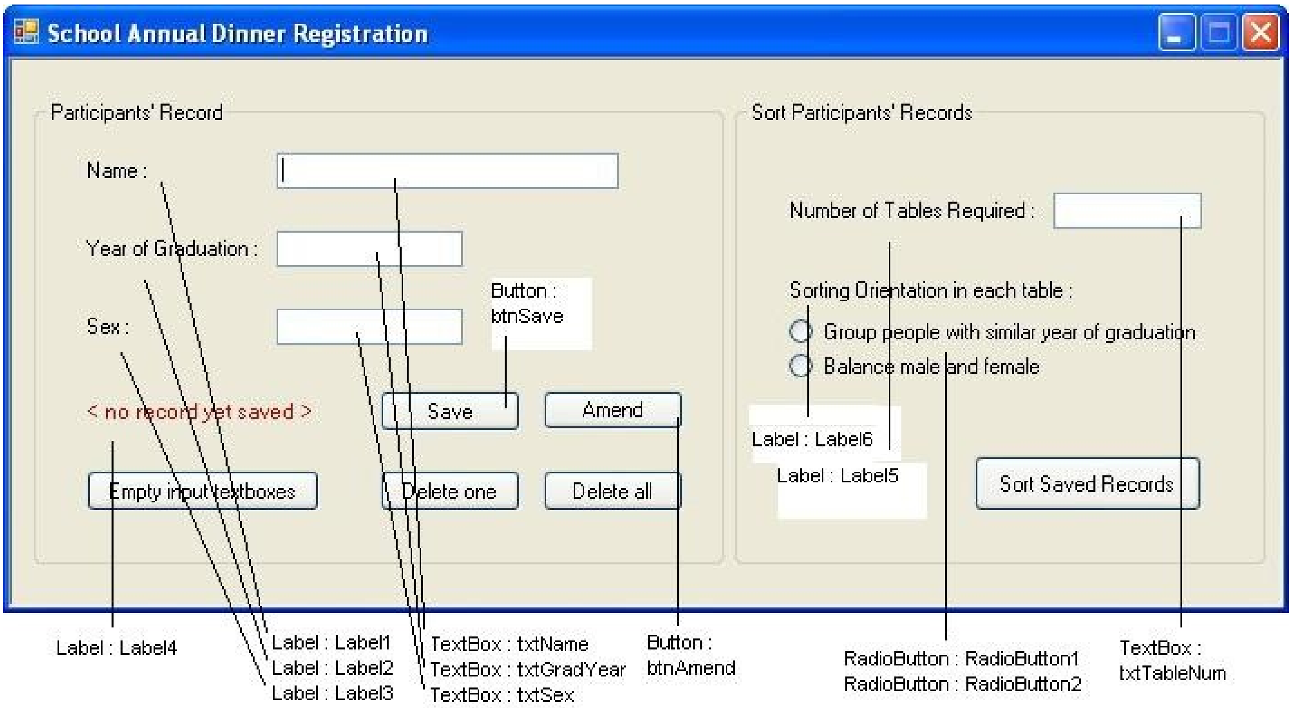
Layout


School Annual Dinner Registration
Participants' Record
Name :
Year of Graduation :
Sex :
< no record yet saved >
Save
Amend
Empty input textboxes
Delete one
Delete all
Sort Participants' Records
Number of Tables Required :
Sorting Orientation in each table :
○ Group people with similar year of graduation
○ Balance male and female
Sort Saved Records

This is the user interface of the program. On the left hand side, the participants record area, there are controls that receive three fields of participants data and provide access to some data manipulation functionalities. On the right hand side, the sort participants records area, there are controls that receive input of the number of tables required, provide options to two sorting orientations, and a command to sort participants records in the computer memory.

The following layouts give information on the controls in the format of :
< type of control > : < name of control in the program >


School Annual Dinner Registration
Participants' Record
Name :
Year of Graduation :
Sex :
< no record yet saved >
Save
Amend
Empty input textboxes
Delete one
Delete all
Sort Participants' Records
Number of Tables Required :
Sorting Orientation in each table :
○ Group people with similar year of graduation
○ Balance male and female
Sort Saved Records
Form : Form1
Button : btnClearTextBoxes
Button : btnCleanOneGuest
Button : btnCleanAllGuests
GroupBox : GroupBox1
GroupBox : GroupBox2
Button : btnSort



Form1

Form1 is a control that contains all other controls of this interface. I have set the MaximizeBox and the FormBorderStyle properties to be Fixed. This prohibit the form to be resized by users and maintain a good look of the interface.

GroupBox1 , GroupBox2

I use these controls to divide the form into two areas so as to better organize the labels, textboxes and buttons of different purposes.

Label1 , Label2 , Label3

They are labels that guide users what fields and where to input the participants records.

txtName , txtGradYear , txtSex

They are textboxes that receive participants data input. I have set the MaxLength properties of *txtName*, *txtGradYear*, and *txtSex* to 50, 4, 6 respectively. So that the number of characters input to the texboxes is limited to those specified maximum length.

Label4

This label displays status of the records in memory under various situations.

btnSave

Whenever this button is clicked, a subroutine is executed to validate the data input, and then create a new record in the computer memory.

btnAmend

Whenever this button is clicked, a subroutine is executed to amend a record in memory, users are required to specify which record to amend.

btnCleanOneGuest, btnCleanAllGuests

Whenever these buttons are clicked, subroutines are executed to delete one specified record or all records in the computer memory respectively.

btnClearTextBoxes

Whenever this button is clicked, a subroutine is called to empty the textboxes *txtName*, *txtGradYear* and *txtSex*.

txtTableNum

This textbox receives users input of the number of tables required. I have set the MaxLength property of it to 4. So that the number of digits input to this textbox is limited to 4.

Label5, Label6

They are labels that guide users where to input the number of tables required, and guide users where to select a sorting orientation respectively.

RadioButton1 , RadioButton2

They are radio buttons that provide two options of sorting orientation, in which users can select only one out of the two.

btnSort

Whenever this button is clicked, a subroutine is executed to sort all records in the computer memory and then save a seating plan in the current directory of the program.

3.3 Design of the Data Validation and Record Saving Algorithm

Whenever users click the “Save” Button (*btnSave)*, the program would first execute the data validation algorithm. If the data are valid, then the program continue to execute the record saving algorithm, to save a valid record in the computer memory.

Data Validation Rules

The data validation algorithm compares the three fields of participants data input to a set of rules. If any one of them does not satisfy the rules, an error message is displayed to indicate the problem, and instruct users the correct format of data input. This process ensures that the data input can be processed by the sorting algorithm without errors. It can sometimes detect errors on the source document too. The rules are presented as in follow :

	data type in VB2005	value range
name	string	unique and not null string
year of graduation	short	1000 to 3000
sex	string	“m” or “f” (case insensitive)

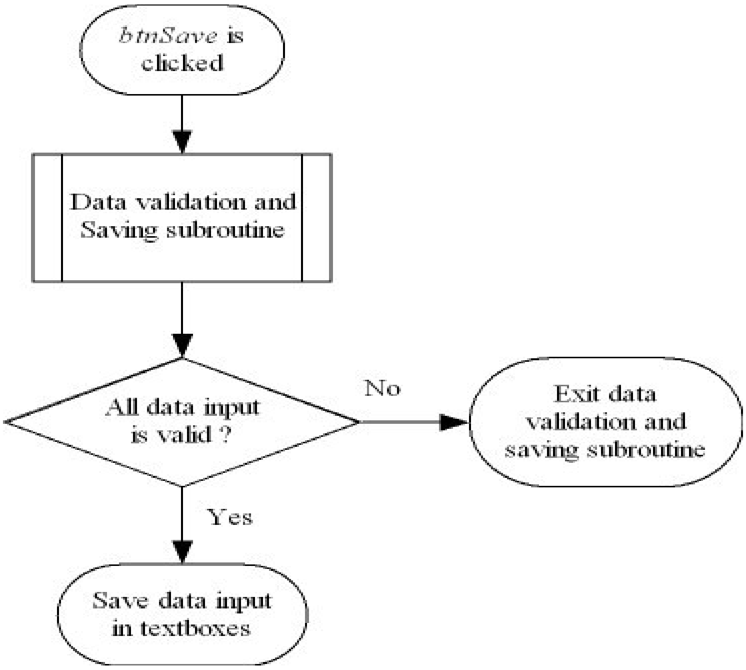
Name : The name of participant is the keyfield. This data field can be any unique strings except a null string.

Year of graduation : The year of graduation data field can be any whole number ranging from 1000 to 3000.

Sex : The sex data field is validated case insensitively. It can be a character 'm' or 'f'. In fact the program implicitly recognize the word 'male' and 'female' too. So that when users really type words instread of characters, the program can also handle the input.

Data Validation Algorithm

The data validation algorithm is embedded in the data validation and saving subroutine. Whenever users want to save a new record of a participant and click the “Save” Button (*btnSave)*, the subroutine would be executed. It always runs before the record saving algorithm.

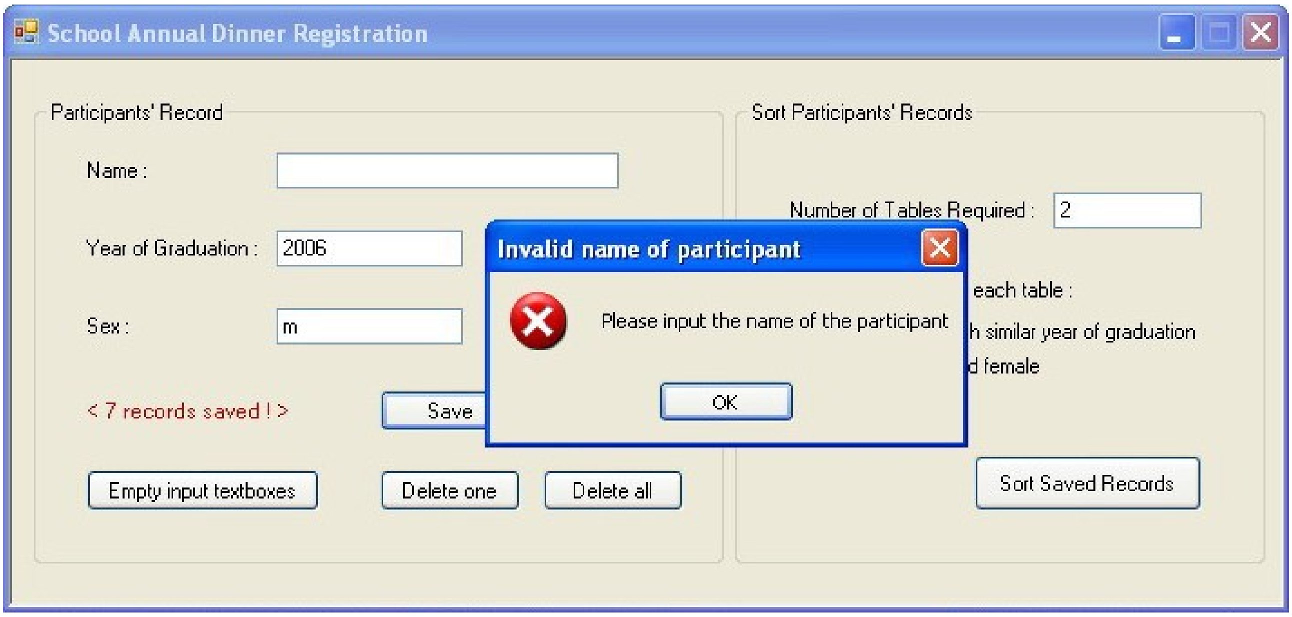

btnSave is clicked
Data validation and Saving subroutine
All data input is valid ?
No
Exit data validation and saving subroutine
Yes
Save data input in textboxes

The data validation algorithm employs two local variables, *temp1* and *temp2*. They are of short data type, meaning that they can store numuric values ranging from -32768 to 32767.

```
'participants data input validation
        Dim temp1, temp2 As Short
        If txtName.Text = "" Then
         MessageBox.Show("Please input the name of the participant", "Invalid name of participant",
      MessageBoxButtons.OK, MessageBoxIcon.Error)
         txtName.Focus()
         Exit Sub
        End If
        For i = 0 To AllGuests.Count - 1
         If txtName.Text = AllGuests(i).Name Then
          MessageBox.Show("There is a record with the same name" & vbCrLf & "Please input another unique
      name of participant", "Invalid name of participant", MessageBoxButtons.OK, MessageBoxIcon.Error)
          txtName.Focus()
          Exit Sub
         End If
        Next
        Try
         temp1 = Short.Parse(txtGradYear.Text)
         If (temp1 < 1000) Or (temp1 > 3000) Then Throw New FormatException
        Catch ex As FormatException
         MessageBox.Show("Please input a whole number ranging from 1000 to 3000", "Invalid year of graduation
      of participant", MessageBoxButtons.OK, MessageBoxIcon.Error)
         txtGradYear.Text = ""
         txtGradYear.Focus()
         Exit Sub
        End Try
        If (txtSex.Text.ToUpper = "M") Or (txtSex.Text.ToUpper = "MALE") Then
         temp2 = 1
        ElseIf (txtSex.Text.ToUpper = "F") Or (txtSex.Text.ToUpper = "FEMALE") Then
         temp2 = -1
        Else
         MessageBox.Show("Please input ""m"" or ""f"" for male or female respectively", "Invalid sex of
      participant", MessageBoxButtons.OK, MessageBoxIcon.Error)
         txtSex.Text = ""
         txtSex.Focus()
         Exit Sub
        End If
```

....

Firstly, the algorithm evaluates if the name of participant textbox is empty. If so, the program displays an error message and terminates the data saving subroutine.



Name :

Year of Graduation : 2006

Sex : m

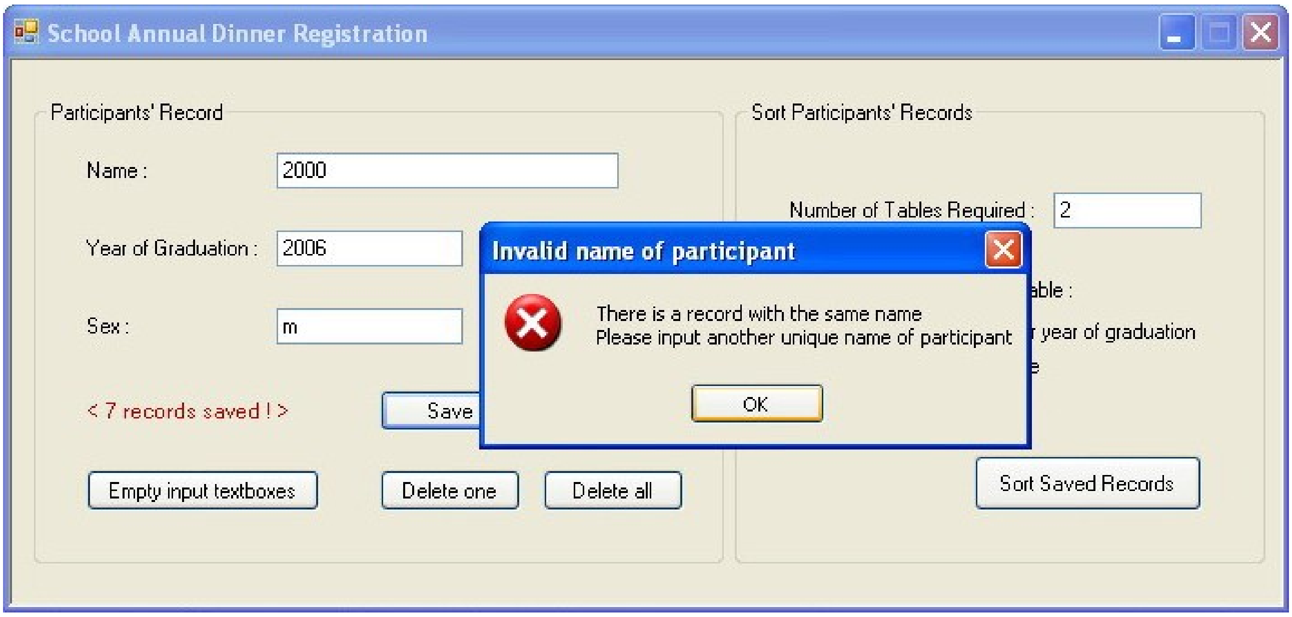
Number of Tables Required : 2

< 7 records saved ! >

Invalid name of participant

Please input the name of the participant

Secondly, the algorithm checks if a same name of participant has been saved into the computer memory. It is because the name of participant is the keyfield, name of participants in each record must be unique. If the input is not unique, the program displays an error message and terminates the data saving subroutine.



Name : 2000

Year of Graduation : 2006

Sex : m

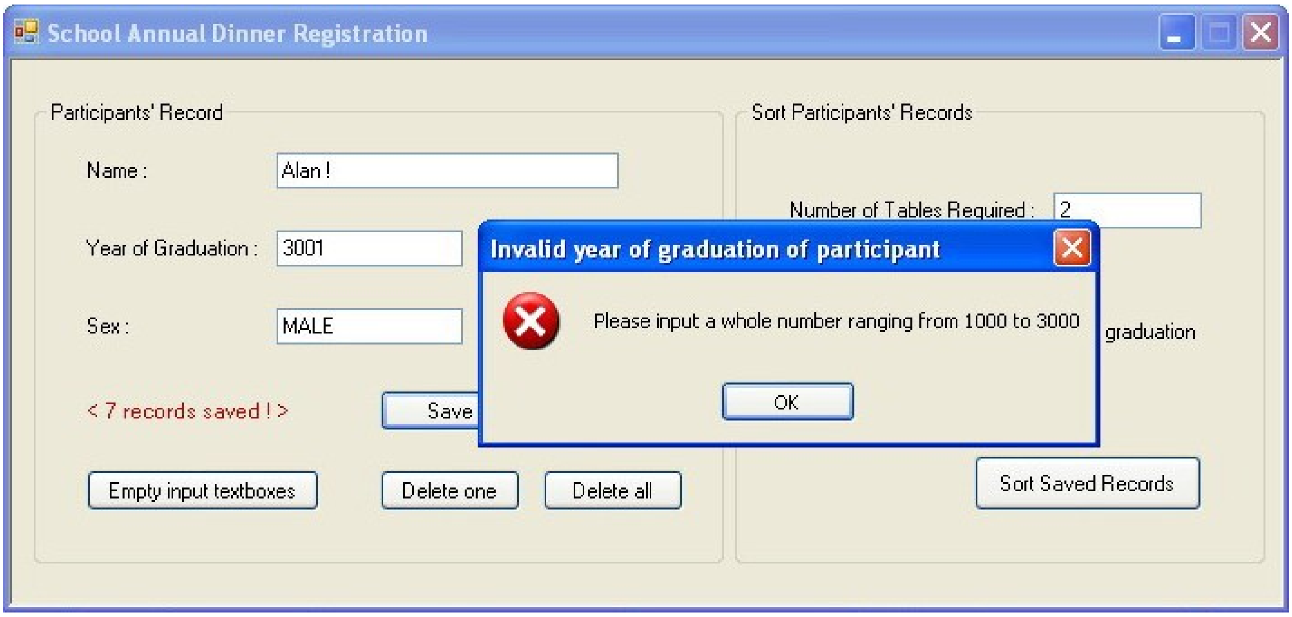
Number of Tables Required : 2

< 7 records saved ! >

Invalid name of participant

There is a record with the same name
Please input another unique name of participant

Thirdly, the algorithm stores the year of graduation into the *temp1*. Then it evaluates if it is out of range or is not compatible with the short data type variable *temp1*. i.e. It checks the data type of the year of graduation. If there is an error, the program displays an error message and terminates the data saving subroutine.



School Annual Dinner Registration

Participants' Record

Name : Alan !

Year of Graduation : 3001

Sex : MALE

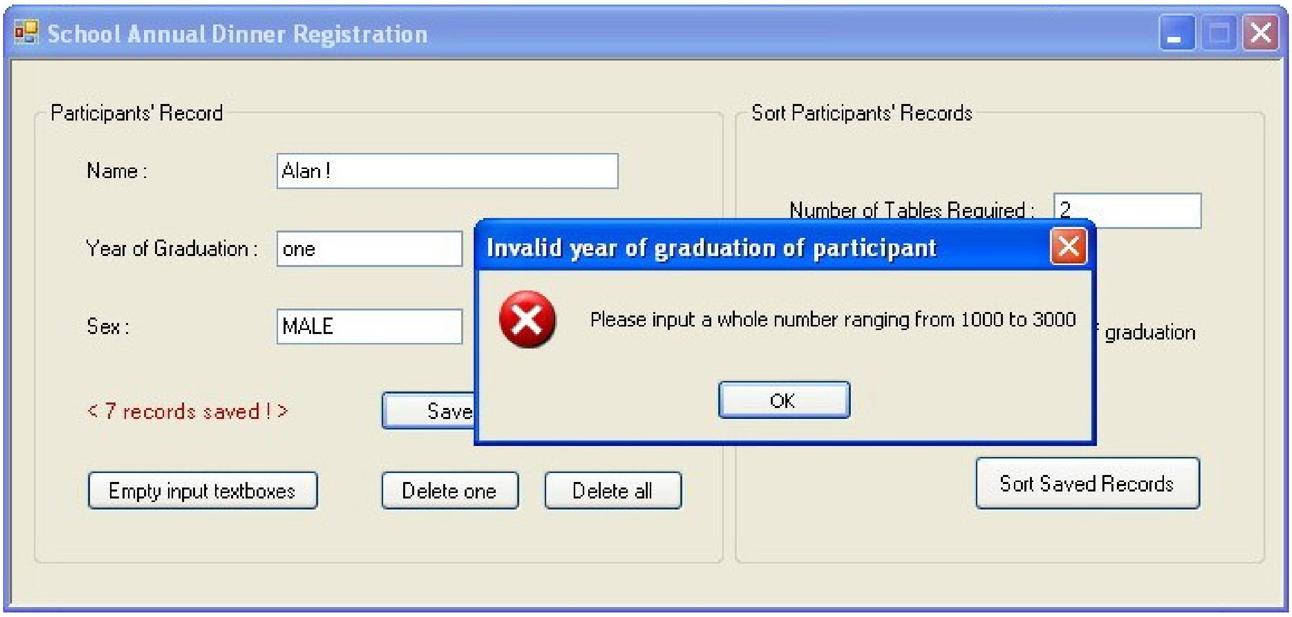
< 7 records saved ! >

Sort Participants' Records

Number of Tables Required : 2

Invalid year of graduation of participant

Please input a whole number ranging from 1000 to 3000



School Annual Dinner Registration

Participants' Record

Name : Alan !

Year of Graduation : one

Sex : MALE

< 7 records saved ! >

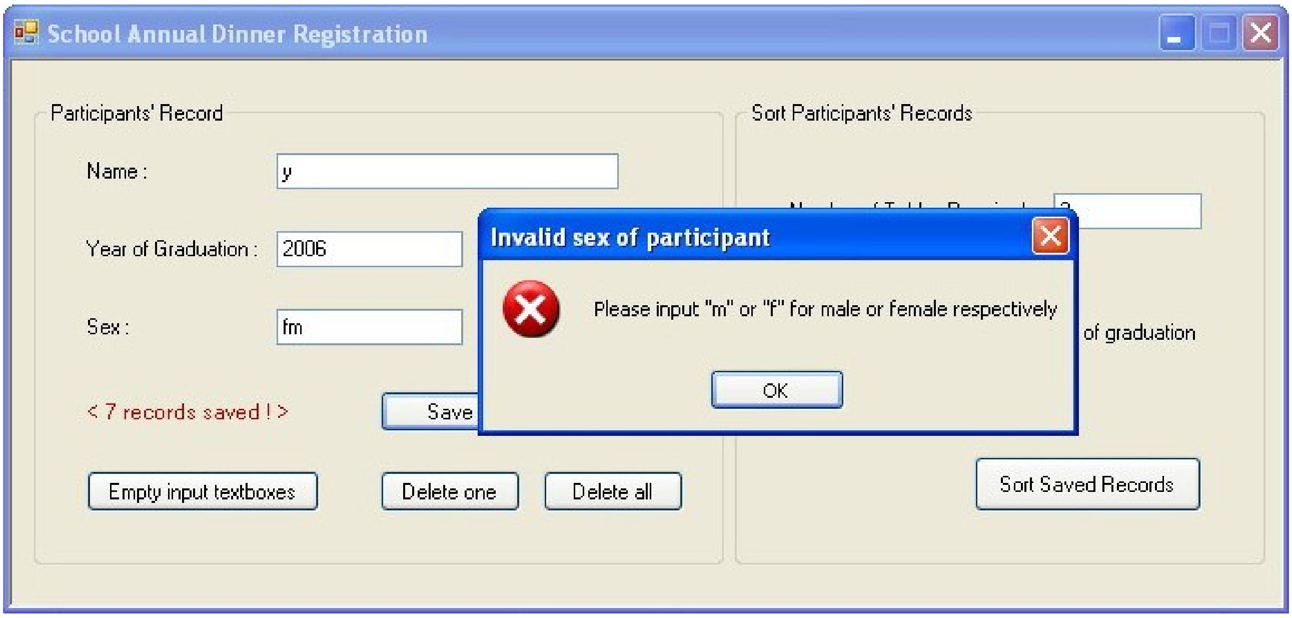
Sort Participants' Records

Number of Tables Required : 2

Invalid year of graduation of participant

Please input a whole number ranging from 1000 to 3000

Finally, the algorithm evaluates if the sex data field is not the string 'M', 'F', 'MALE', 'FEMALE' (case insensitive). It so, the program displays an error message and terminates the data saving subroutine.



School Annual Dinner Registration

Participants' Record

Name : y

Year of Graduation : 2006

Sex : fm

< 7 records saved ! >

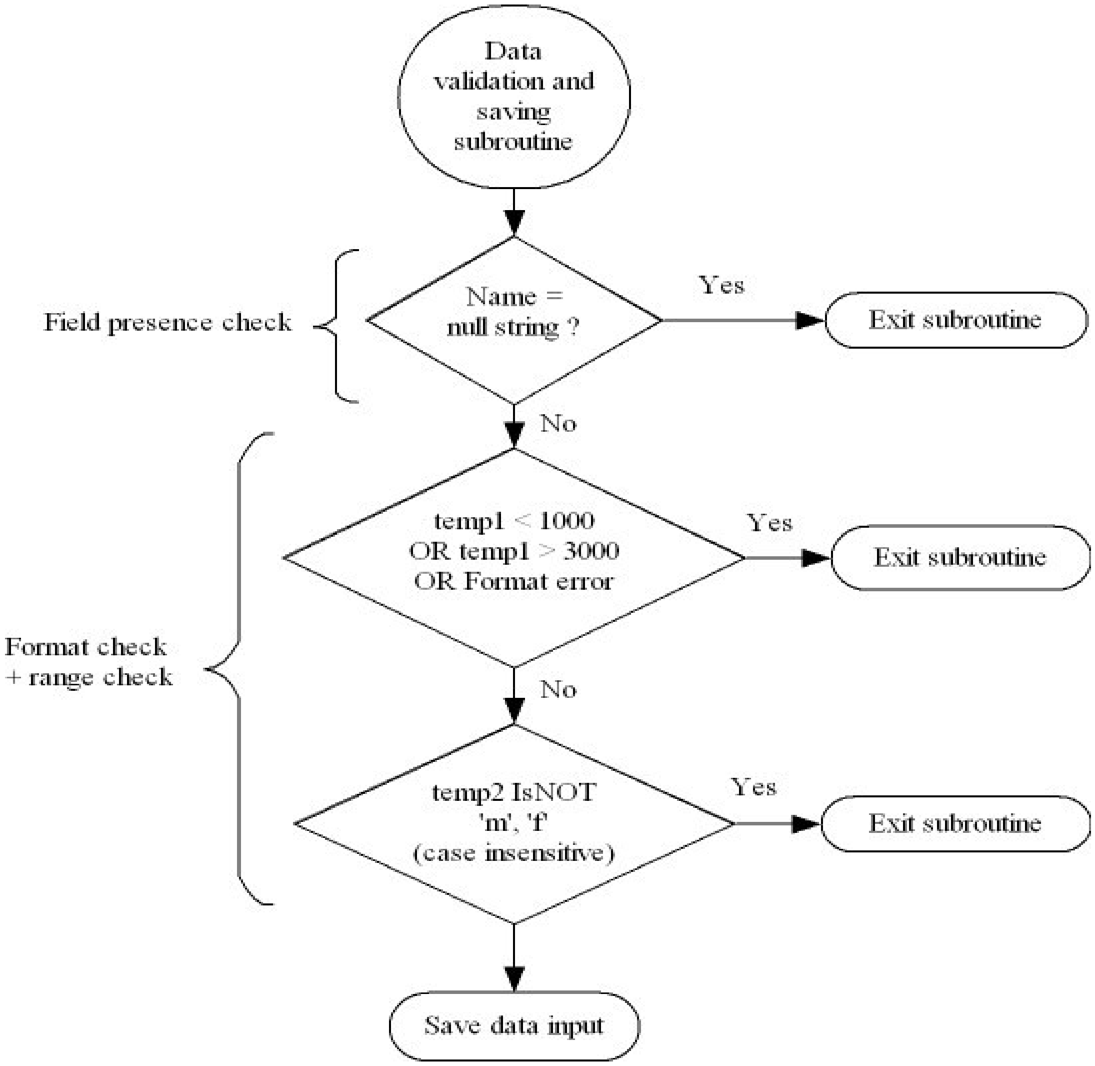
Sort Participants' Records

of graduation

Invalid sex of participant

Please input "m" or "f" for male or female respectively

Summary of the Data Validation Algorithm



Data validation and saving subroutine
Field presence check
Name = null string ?
Yes
Exit subroutine
No
Format check + range check
temp1 < 1000 OR temp1 > 3000 OR Format error
Yes
Exit subroutine
No
temp2 IsNOT 'm', 'f' (case insensitive)
Yes
Exit subroutine
Save data input

Record Saving Algorithm

The record saving algorithm executes after the participants data input pass all the validations. In this program, the maximum records allowed to save in memory is 1000 records. So the algorithm first evaluate the number of records saved by a selection control structure. If it reaches 1000, then the program would display en error message and terminate the subroutine itself.

If it is a value in the range from 0 to 999, then it creates a new *Guest* object as a representation of a record in the computer memory. Then it passes the validated participant data in the three textboxes, *txtName, txtGradYear, txtSex* into the properties *Guest.Name, Guest.GradYear, Guest.Sex* of the *Guest* object respectively. Then it adds the newly initiated *Guest* object into the *AllGuests* collection object.

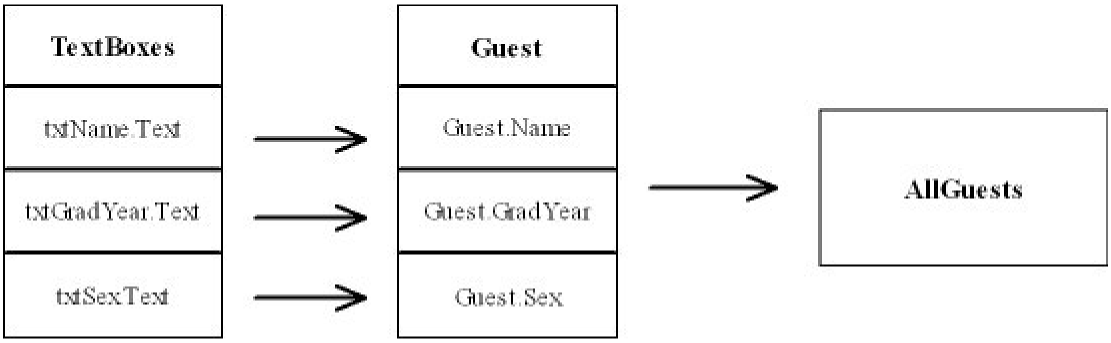
After initiating a record, it calls the *ClearDataInput()* subroutine to empty all textboxes. So that users can immediately type the next record in the blank textboxes.

In the following I would present the code section of record saving.

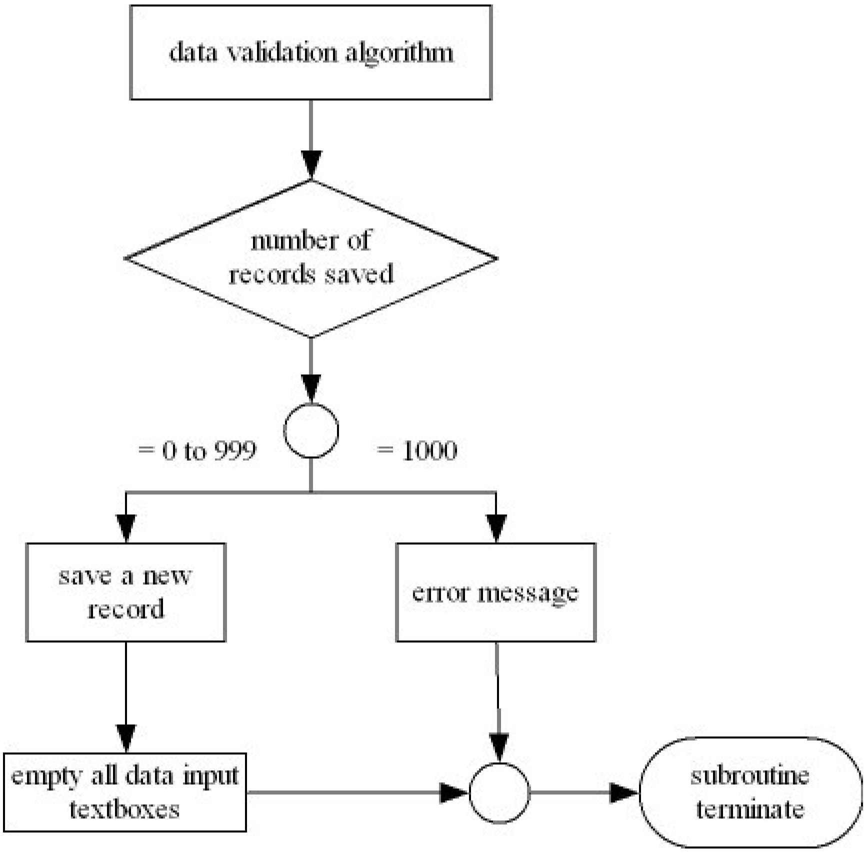
```
.............................
   'create and save a new record
               Select Case AllGuests.Count
            Case 0 To 999
                   Dim Guest As New GuestClass()
                   Guest.Name = txtName.Text
                   Guest.GradYear = temp1
                   Guest.Sex = temp2
                   AllGuests.Add(Guest)
                   Label4.Text = "< " & AllGuests.Count.ToString() & " records saved ! >"
                   Call ClearDataInput()
                 Case 1000
                   MessageBox.Show("You haved reach the maximum number of records to be saved" & vbCrLf & "The
             new record cannot be saved", "This program does not support records saved to be over 1000",
             MessageBoxButtons.OK, MessageBoxIcon.Error)
               End Select
```

Summary of the Record Saving Algorithm

Data flow



Control flow



Other Records Manipulation Algorithms

After users input records into the program, they may want to modify the records in the computer memory. This program provides some minor functionalities for users to do so.

Record Amendment Algorithm

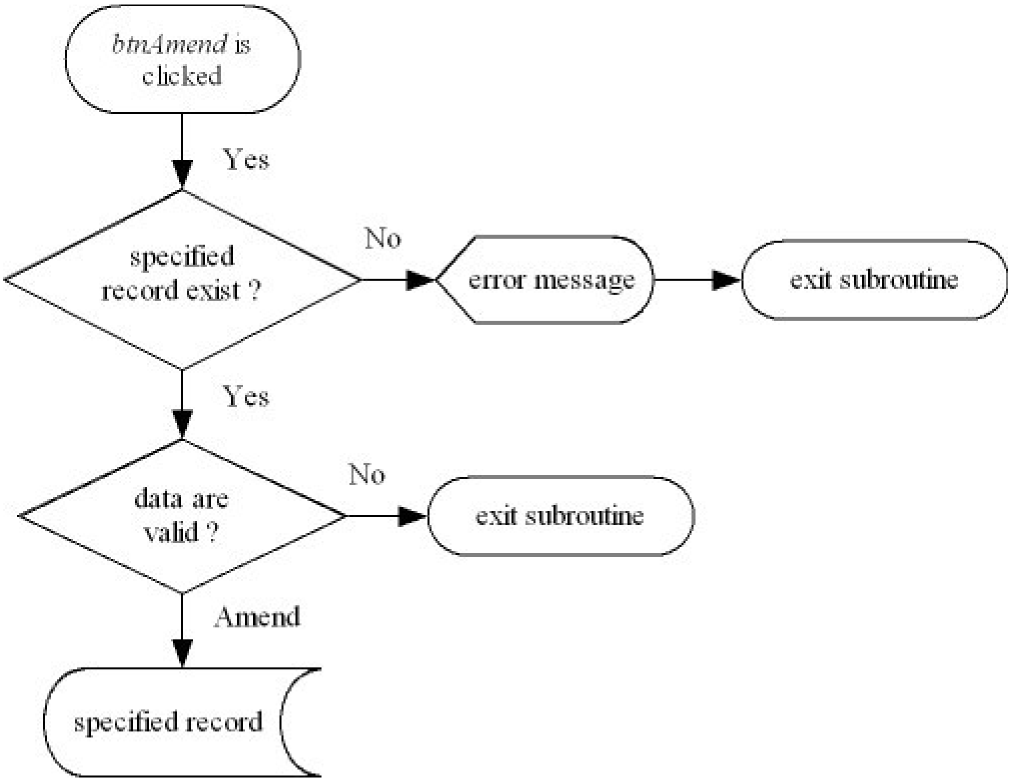
The subroutine is executed whenever the "Amend" button (*btnAmend*) is clicked. It enables users to amend the year of graduation and sex of graduation of records in the computer memory. Users are required to input the name of participant in the textbox *txtName* to tell the program which record to amend. Then the program would amend the specifies record with data in textboxes *txtGradYear* and *txtSex*. Before the amendment, the data go through validation similar to that when a new record is saved.

The data amendment section employs two local variable, temp1 and temp2. They are of short data type, meaning that they can store numuric values ranging from -32768 to 32767.

The code section of the record amendment subroutine is presented as in follow :

```
 Private Sub btnAmend_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles
btnAmend.Click
  Dim temp1, temp2 As Short
  If txtName.Text = "" Then
   MessageBox.Show("Please input the name of the participant", "Invalid name of participant",
MessageBoxButtons.OK, MessageBoxIcon.Error)
   txtName.Focus()
   Exit Sub
  End If
  For i = 0 To AllGuests.Count - 1
   If AllGuests(i).Name = txtName.Text Then
    Try
     temp1 = Short.Parse(txtGradYear.Text)
     If (temp1 < 1000) Or (temp1 > 3000) Then Throw New FormatException
    Catch ex As FormatException
     MessageBox.Show("Please input a whole number ranging from 1000 to 3000", "Invalid year of
graduation of participant", MessageBoxButtons.OK, MessageBoxIcon.Error)
     txtGradYear.Text = ""
     txtGradYear.Focus()
     Exit Sub
    End Try
    If (txtSex.Text.ToUpper = "M") Or (txtSex.Text.ToUpper = "MALE") Then
     temp2 = 1
    ElseIf (txtSex.Text.ToUpper = "F") Or (txtSex.Text.ToUpper = "FEMALE") Then
     temp2 = -1
    Else
     MessageBox.Show("Please input ""m"" or ""f"" for male or female respectively", "Invalid sex of
participant", MessageBoxButtons.OK, MessageBoxIcon.Error)
     txtSex.Text = ""
     txtSex.Focus()
     Exit Sub
    End If
    With AllGuests(i)
     .GradYear = temp1
     .Sex = temp2
    End With
    Label4.Text = "< record amended ! >"
    Call ClearTextBoxes()
    Exit Sub
   End If
  Next
  MessageBox.Show("Record not found", "Record not found", MessageBoxButtons.OK,
MessageBoxIcon.Information)
  txtName.Focus()
 End Sub
```

The subroutine first checks if the record specified by users exists. If it does, then it goes through the data validation. Then it access the specified *Guest* object in the *AllGuests* collection object, amend the properties of it and empty all data input textboxes. If the specified record does not exist, then the program displays en error message and terminates itself.



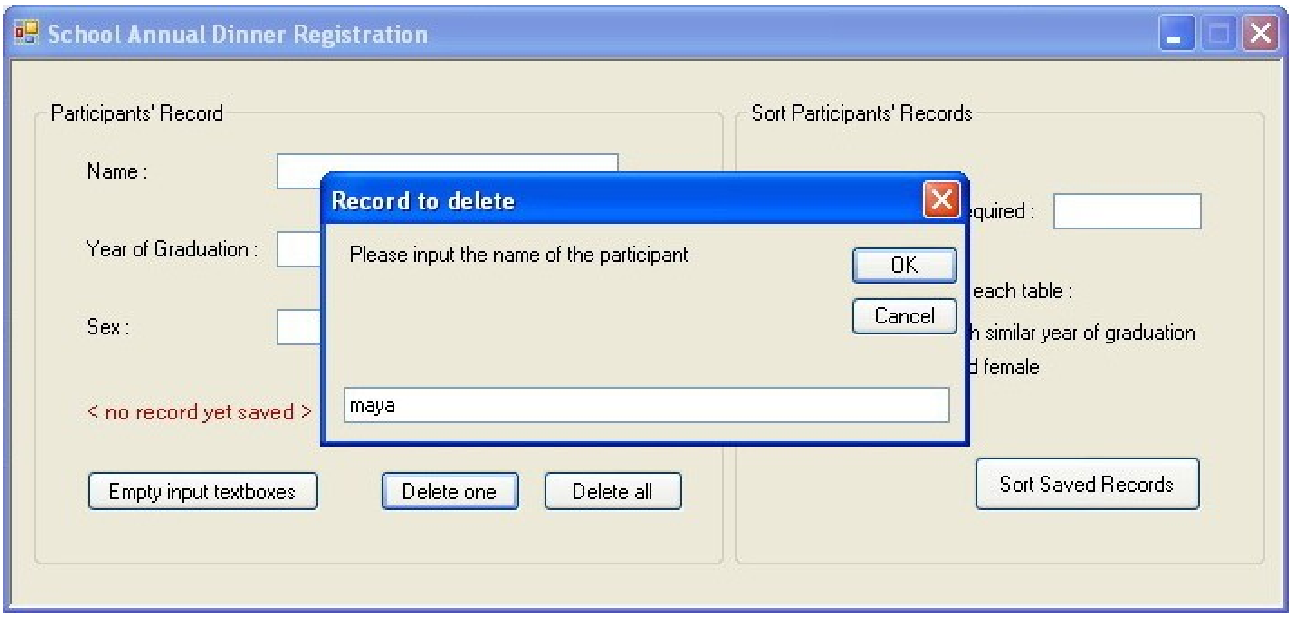
Records Removal Algorithm

Users can choose either to delete one specific record or delete all records in the computer memory, pressing the "Delete one" button (*btnCleanOneGuest)* and the "Delete all" button (*btnCleanAllGuests*) respectively.

Delete one specified record

```
 Private Sub btnCleanOneGuest_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnCleanOneGuest.Click
   Dim tempName As String
   tempName = InputBox("Please input the name of the participant", "Record to delete")
   For i = 0 To AllGuests.Count - 1
     If AllGuests(i).Name = tempName Then
       AllGuests.RemoveAt(i)
       Label4.Text = "< record deleted ! >"
       txtName.Focus()
       Exit Sub
     End If
   Next i
   MessageBox.Show("Record not found", "Record not found", MessageBoxButtons.OK,
MessageBoxIcon.Information)
   txtName.Focus()
 End Sub
```

Whenever users click the "Delete one" button, the subroutine is executed. It first prompts an inputbox to ask users for the record to delete.

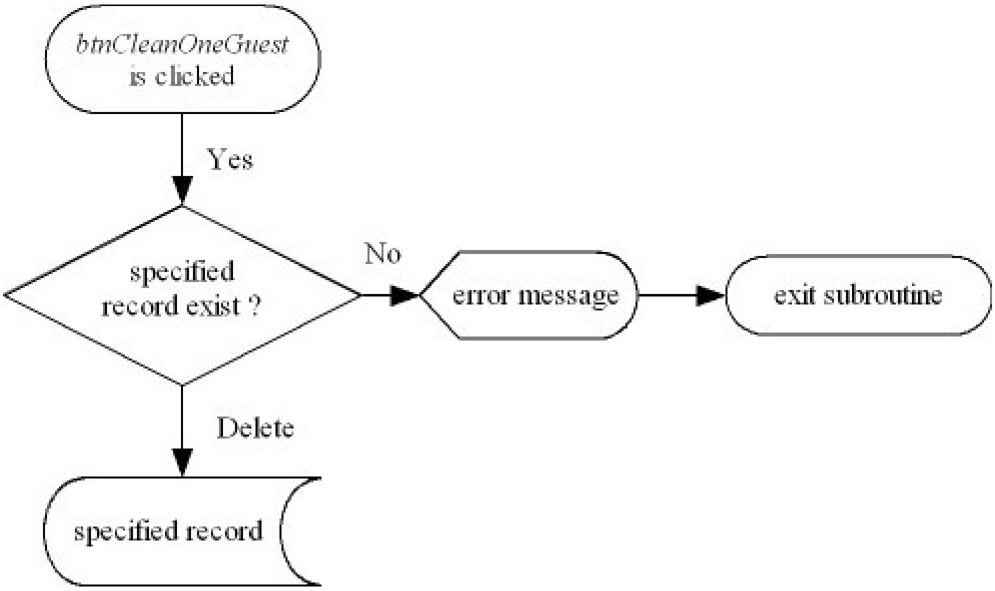


Record to delete

Please input the name of the participant

maya

If the specified record exists, the program deletes it from the computer memory. It removes the specified *Guest* object from the *AllGuests* collection object. Otherwise an error message is displayed, telling users that the specified record does not exist.

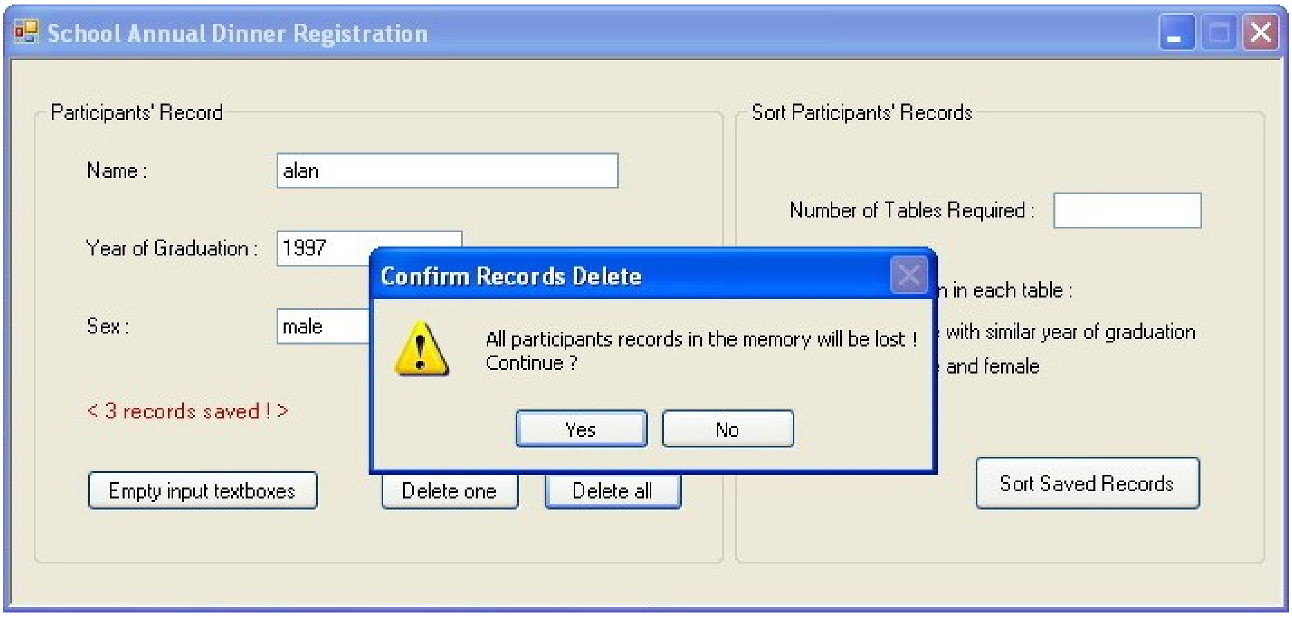


Delete all records

Whenever users click the "Delete all" button, the subroutine is executed.

```
 Private Sub btnCleanAllGuests_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnCleanAllGuests.Click
    'remove all participants record from the memory
    Dim result As DialogResult
    result = MessageBox.Show("All participants records in the memory will be lost !" & vbCrLf & "Continue ?
", "Confirm Records Delete", MessageBoxButtons.YesNo, MessageBoxIcon.Warning)
    If result = Windows.Forms.DialogResult.Yes Then
      If AllGuests.Count > 0 Then
        AllGuests.Clear()
        Label4.Text = "< all records are deleted ! >"
        txtName.Focus()
      ElseIf AllGuests.Count = 0 Then
        MessageBox.Show("There are no participants records to delete", "No records found",
MessageBoxButtons.OK, MessageBoxIcon.Stop)
      End If
    End If
 End Sub
```

It first shows a message box warning that users would lose all records in the computer memory. If the user press "Yes" in the warning message box, then the program checks if there is any record in the computer memory. It checks that if there is any *Guest* object inside the *AllGuests* collection object. If there are one or more records, then the program deletes all records in the computer memory. It removes all Guest objects from the AllGuests collection objct. Otherwise, if there is no *Guest* object in the AllGuest collection object, then the program displays an error message terminates itself.



School Annual Dinner Registration

Participants' Record

Name : alan

Year of Graduation : 1997

Sex : male

< 3 records saved ! >

Sort Participants' Records

Number of Tables Required :

n in each table :

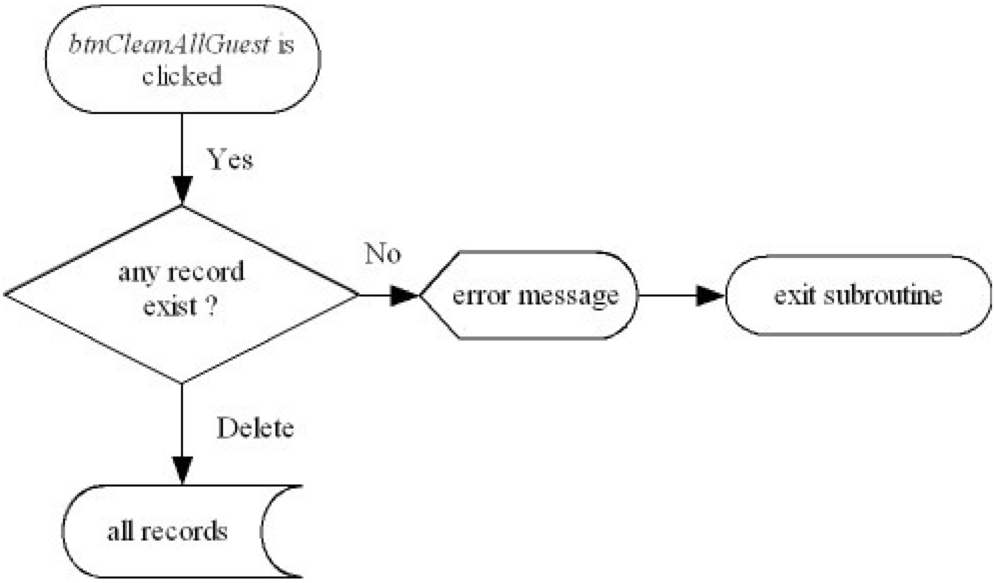
with similar year of graduation

and female

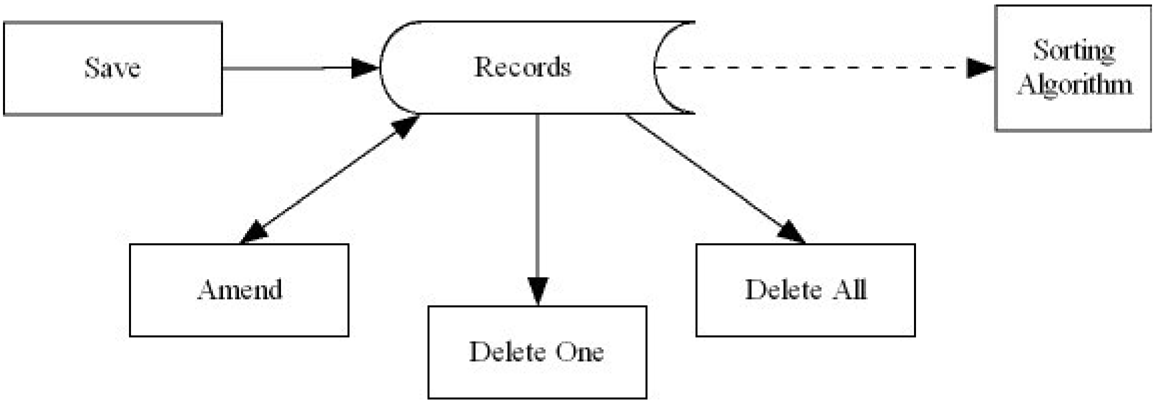
Confirm Records Delete

All participants records in the memory will be lost !
Continue ?

Control Flow



Summary of the Records Manipulation



3.4 Design of the Sorting Algorithm

The target of the sorting algorithm is to sort the records saved in the computer memory. That is to say, to rearrange the *Guest* objects in the *AllGuests* collection object in a particular pattern. The pattern is defined by the two seat allocation rules.

In this design, I want to give users a greater control by :

- Users can input a parameter, the number tables required, to control how many tables the participants are to be sort into.
- Users can select one out of two sorting orientation, to control which seat allocation rule the seating plan would satisfy more.

The design of the sorting algorithm is quite complicated. It involves selections and loops at most nested in five layers. So it is not my intention here to go through the details. Instead I would only provide an abstraction and highlight the features in the design.

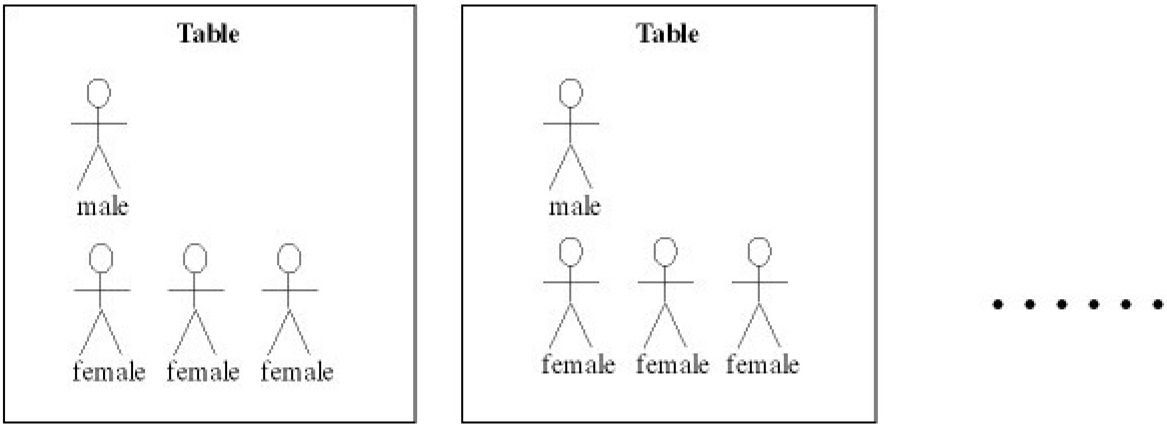
Definion of Male and Female

Although the program accepts users to input 'm' or 'f' as the sex of participants, in the program male and female are stored as numeric values +1 and -1 respectively. The advantages of numeric value storage is that it allows calculations on the sex of participants. It only changes the format of the information, not the information itself.

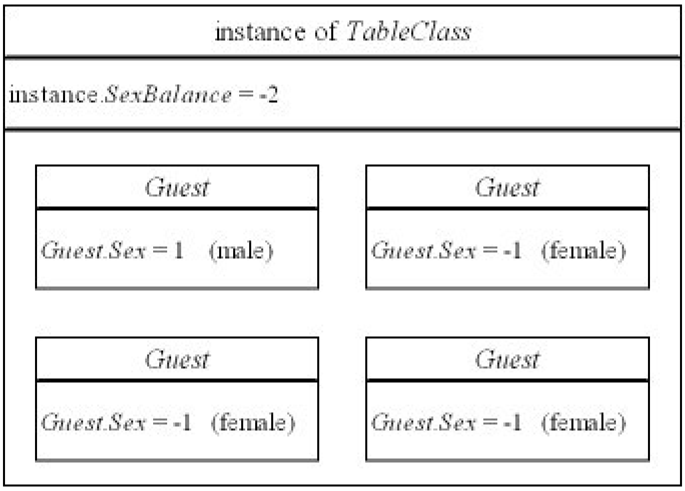
Definition of the Seat Allocation Rules

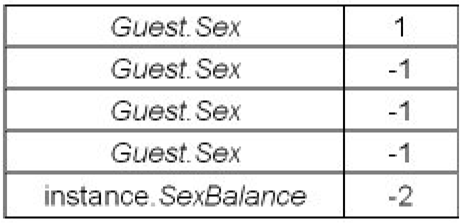
Balance male and female participants in each table

This allocation rule can be defined as, to equalize "the difference of the difference of number of male and female between tables." For example, if every tables contain exactly one male and three female, then the difference of the number of male and female for every tables is 2. Given that the difference of every tables are equal, It is said to be perfectly satisfy the seat allocation rule.



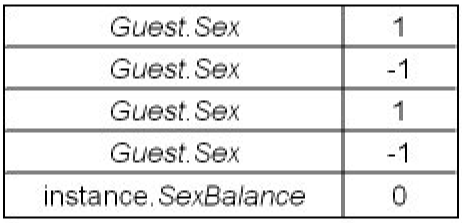
For each instances of the class *TableClass*, a property *SexBalance* is defined. Since male and female is defined as +1 and -1 respectively, the *SexBalance* property is defined as the 'sum' of male and female. So if there is *n* male in a table, and *m* female in the same table, then the *SexBalance* property of that table would be (*n* - *m*).


instance of *TableClass*
instance.*SexBalance* = -2
Guest
Guest.Sex = 1 (male)
Guest
Guest.Sex = -1 (female)
Guest
Guest.Sex = -1 (female)
Guest
Guest.Sex = -1 (female)



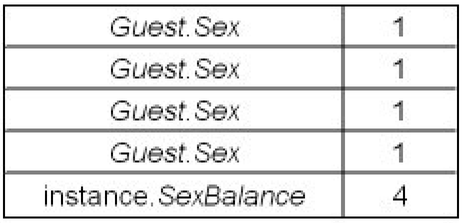
Guest.Sex	1
Guest.Sex	-1
Guest.Sex	-1
Guest.Sex	-1
instance.*SexBalance*	-2

To allocate male and female more evenly, the algorithm should equalize the difference of number of male and female between tables. That is to say, for each instances of *TableClass* in the *AllTables* collection object, their *SexBalance* should be as close as possible.



Guest.Sex	1
Guest.Sex	-1
Guest.Sex	1
Guest.Sex	-1
instance.*SexBalance*	0

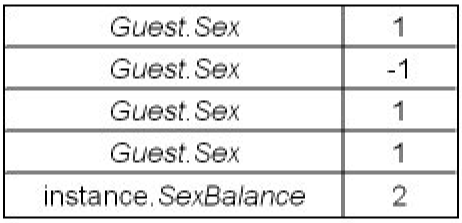
Table A



Guest.Sex	1
Guest.Sex	1
Guest.Sex	1
Guest.Sex	1
instance.*SexBalance*	4

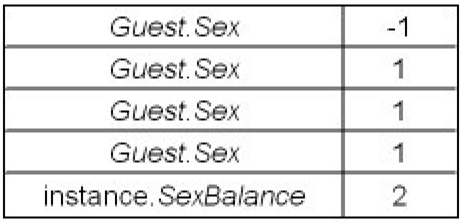
Table B

In the above case, the difference of *SexBalance* between Table A and Table B is 4. Table A itself perfectly balanced with equal number of male and female, though in Table B there are excess male. This configuration does not best satisfy the allocation rule defined.



Guest.Sex	1
Guest.Sex	-1
Guest.Sex	1
Guest.Sex	1
instance.*SexBalance*	2

Table A



Guest.Sex	-1
Guest.Sex	1
Guest.Sex	1
Guest.Sex	1
instance.*SexBalance*	2

Table B

In the above case, the difference of *SexBalance* between Table A and Table B is 0. Table B have swapped a female to Table A with a male. This configuration does best satisfy the allocation rule defined.

Grouping participants of similar year of graduation in each table

By observation, a sorted list naturally groups similar numbers together, as illustrated below :

6	0	1	0	3	2
0	0	1	2	3	6

The upper row of above table is a list of number arranging from left to right. Suppose that every two consecuetive numbers form a group, then it contains groups of {6, 0}, {1, 0}, {3, 2}. After sorting, as in the lower row shows, the groups become {0, 0}, {1, 2}, {3, 6}. It can be observed that after sorting, similar numbers naturally group together. More importantly, the overall dispersion in each particular group is smaller.

This obervation implies how to achieve a similar year of graduation in each table. It shows that I can implement a sorting algorithm to sort the list of participants according to the year of graduation, then in the sorted list, participants of similar year of graduation would naturally be grouped together.

Design of the Record Sorting Algortithm

The sorting algorithm is executed whenever users click the "Sort Saved Record" button (*btnSort)*. It first validates sorting parameters that is input by users. Secondly it sorts the records in memory by year of graduation of participants. Then the algorithm executes further manipulations according to the number of tables required and the sorting orientation chosen.

Sorting parameters validation

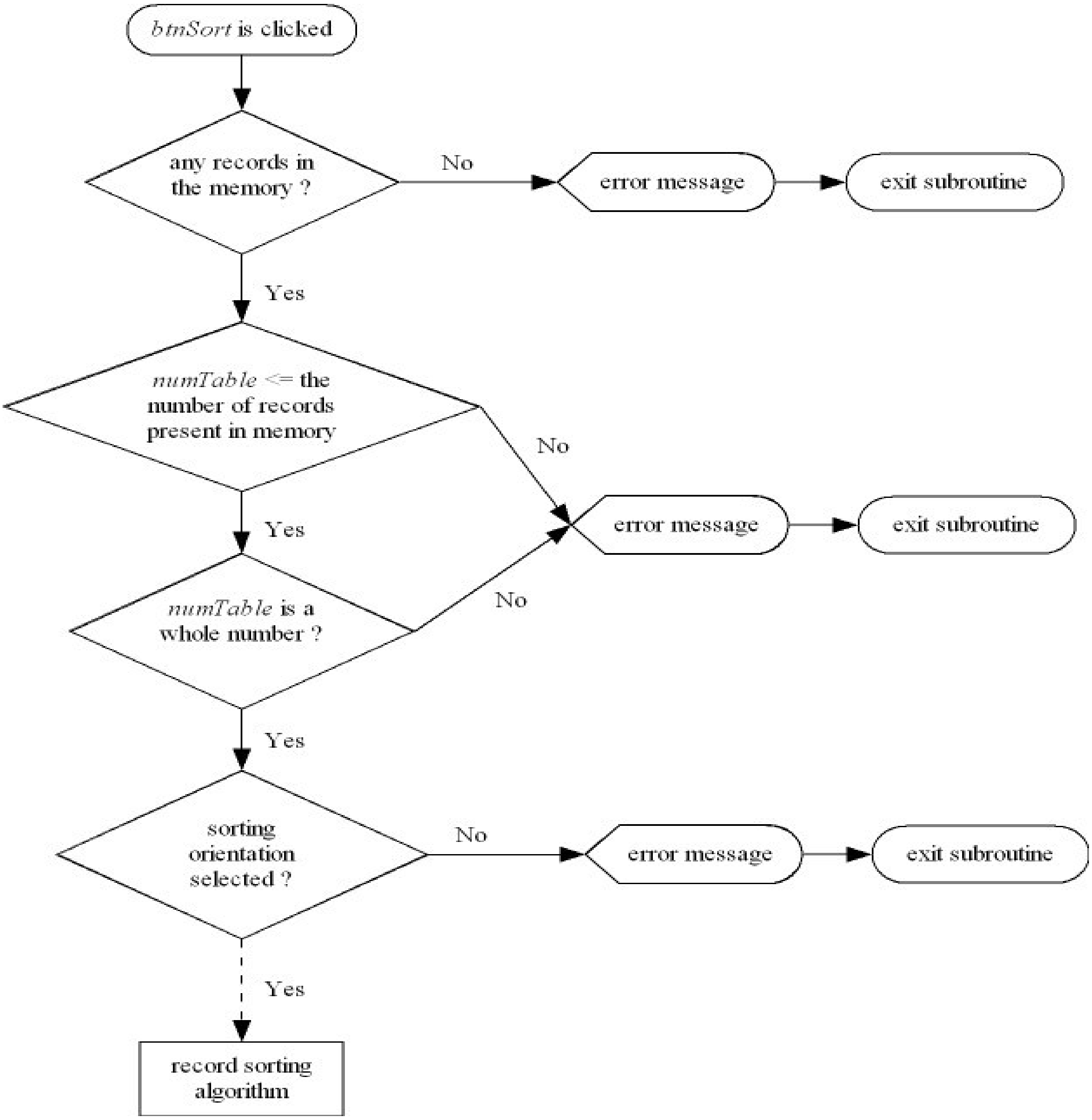
This algorithm is the first algorithm to be executed whenever the "Sort Saved Record" button is clicked. The sorting parameters validation algorithm employs one local variable, *numTable*. It is of short data type, meaning that it can store a numric value ranging from -32768 to 32767.

```
  'parameters validation
  If AllGuests.Count = 0 Then
    MessageBox.Show("Please input at least one participant record", "No participant record for sorting",
MessageBoxButtons.OK, MessageBoxIcon.Error)
    Exit Sub
  End If
  Try
    numTable = Short.Parse(txtTableNum.Text)
    If numTable > AllGuests.Count Then Throw New FormatException
  Catch ex As FormatException
    MessageBox.Show("Please input a whole number that is not larger than" & vbCrLf & "the number of
participants to be sort", "Invalid number of tables", MessageBoxButtons.OK, MessageBoxIcon.Error)
    txtTableNum.Focus()
    Exit Sub
  End Try
  If Not (RadioButton1.Checked Or RadioButton2.Checked) Then
    MessageBox.Show("Please select one sorting orientation in each table", "No sorting orientation selected",
MessageBoxButtons.OK, MessageBoxIcon.Error)
    Exit Sub
  End If
```

The algorithm first checks if there is any record in the computer memory. It checks if there is any *Guest* object in the *AllGuests* collection object. It there is, then the algorithm go on to execute. Otherwise it displays an error message and terminates the record sorting subroutine.

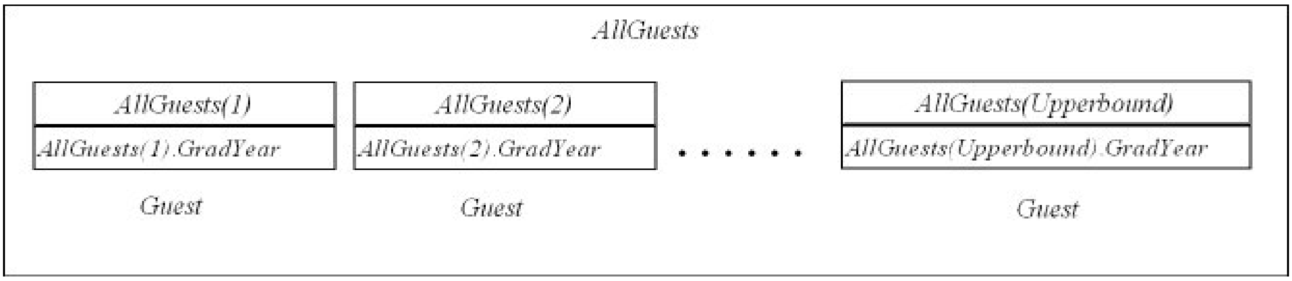
Secondly it validate the variable *numTable.* In the algorithm, *numTable* temporarily stores the input drawn from the *txtTableNum* textbox. It stores the number of tables required in the data type short. If it is number that is larger than the number of records in the computer memory, then there must be empty tables in the seating plan. Thus the program displays an error message. If the number of tables required is not a whole number, an error message is also displayed. After displaying the error messages, the subroutine terminates the record sorting subroutine.

Finally the algorithm checks if users have selected a sorting orientation. It checks if there is any radiobutton being clicked. If it does, then the algorithm continues the execution of the record sorting algorithm. Otherwise, it displays an error message and terminates record sorting subroutine.



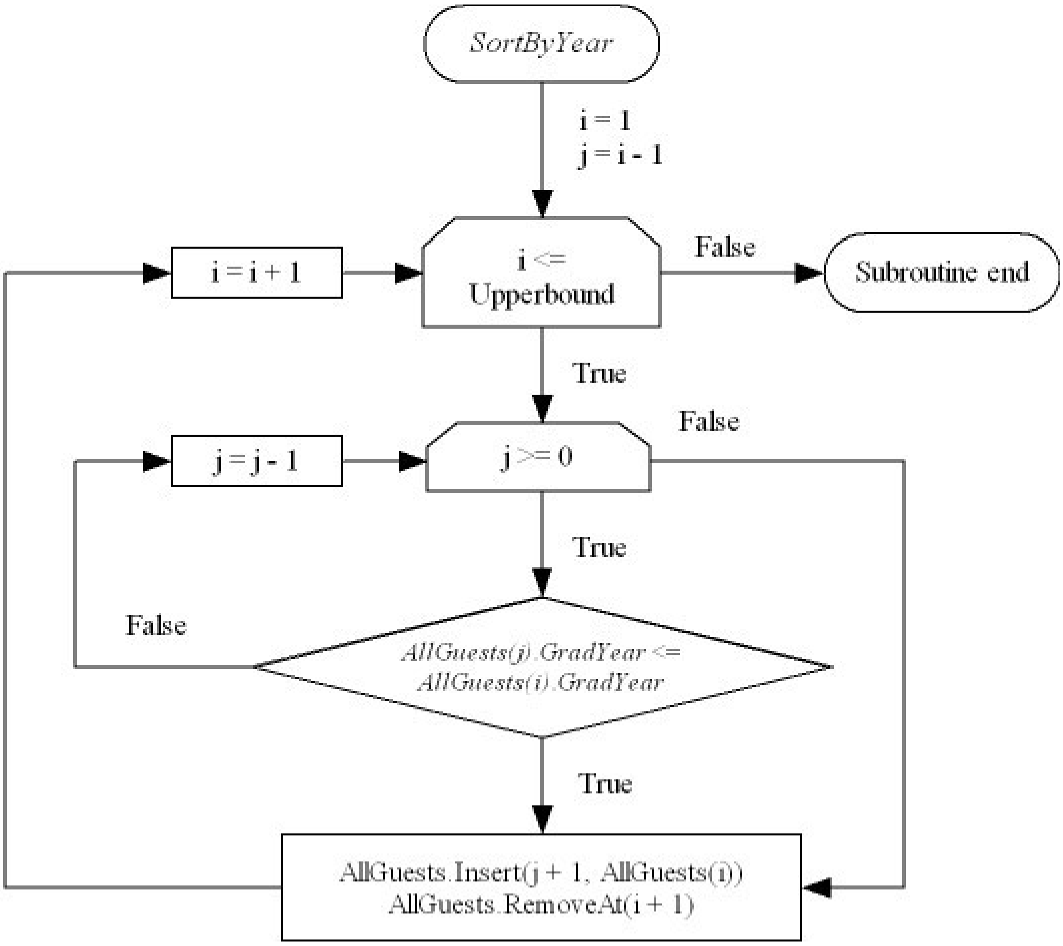
Implementation of insertion sort

At this stage, the records and the sorting parameters are all validated. And the record sorting algorithm begins. The first step to apply an insertion sort to sort the records in the computer memory. It rearranges the records in ascending order of year of graduation of participants. The record sorting algorithm calls the public subroutine *SortByYear* to do the such insertion sort.



The parameter Upperbound is the largest index accessible in the collection object *AllGuests*. It equals to (the number of *Guest* in *AllGuests*) - 1 , the minus one appears because the *AllGuests* collection object adopts zero-based indices (0, 1, 2, ... n). It is used as the final value of the outer loop of the insertion sort algorithm. It represents that the outer loop would loop through all *Guest* objects in the *AllGuest* collection object.

```
Public Sub SortByYear(ByVal UpperBound As Short)
  'rearrange participants records in ascending order of year of graduation by insertion sort
  For i = 1 To UpperBound
    For j = i - 1 To 0 Step -1
      If AllGuests(j).GradYear <= AllGuests(i).GradYear Then Exit For
    Next
    AllGuests.Insert(j + 1, AllGuests(i))
    AllGuests.RemoveAt(i + 1)
  Next i
End Sub
```



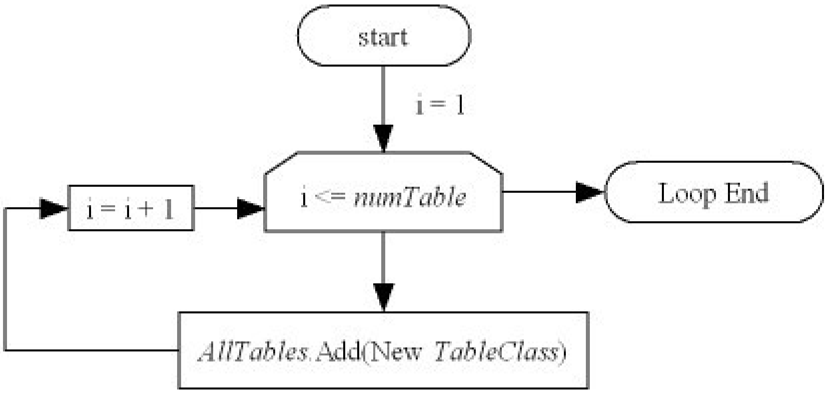
Create required number of tables

After the *SortByYear* subroutine, the records in the computer memory are sorted. That means all *Guest* objects in the *AllGuests* collection object are sorted in the way that, the smaller the index of the *Guest* object in the *AllGuests* collection object, the smaller the year of graduation of that record (*AllGuests(i).GradYear*).

After that the record sorting algorithm creates the required number of tables in the computer memory. Later records would be allocated into these tables one by one.

```
'create the specified number of tables required
AllTables.Clear()
For i = 1 To numTable
  AllTables.Add(New TableClass)
Next
```

The program create new instances of the *TableClass* class, and put the instances into the *AllTables* collection object.



Main section of the record sorting algorithm

One special feature of the sorting algorithm design is the option of a sorting orientation. In most cases, a set of records would not satisfy both allocation rules at the same time, so a tradeoff has to be made. In a tradeoff, there are priorities. In this program, I provide users a choice between the priority in year of graduation of participants and the sex of the participants. Choosing either one of the sorting orientation does not mean the seating plan would best satisfy that seat allocation rule. It just means the seating plan would better satisfy it. Because at the same time, the other rule is still being taken into consideration by the sorting algorithm.

This main section of the record sorting algorithm employs three local variables, *numSeat, numSexBalance* and *numSexDifference*. They are of short data type, meaning that they can store numuric values ranging from -32768 to 32767.

First of all, the algorithm calculates the number seats each tables should have.

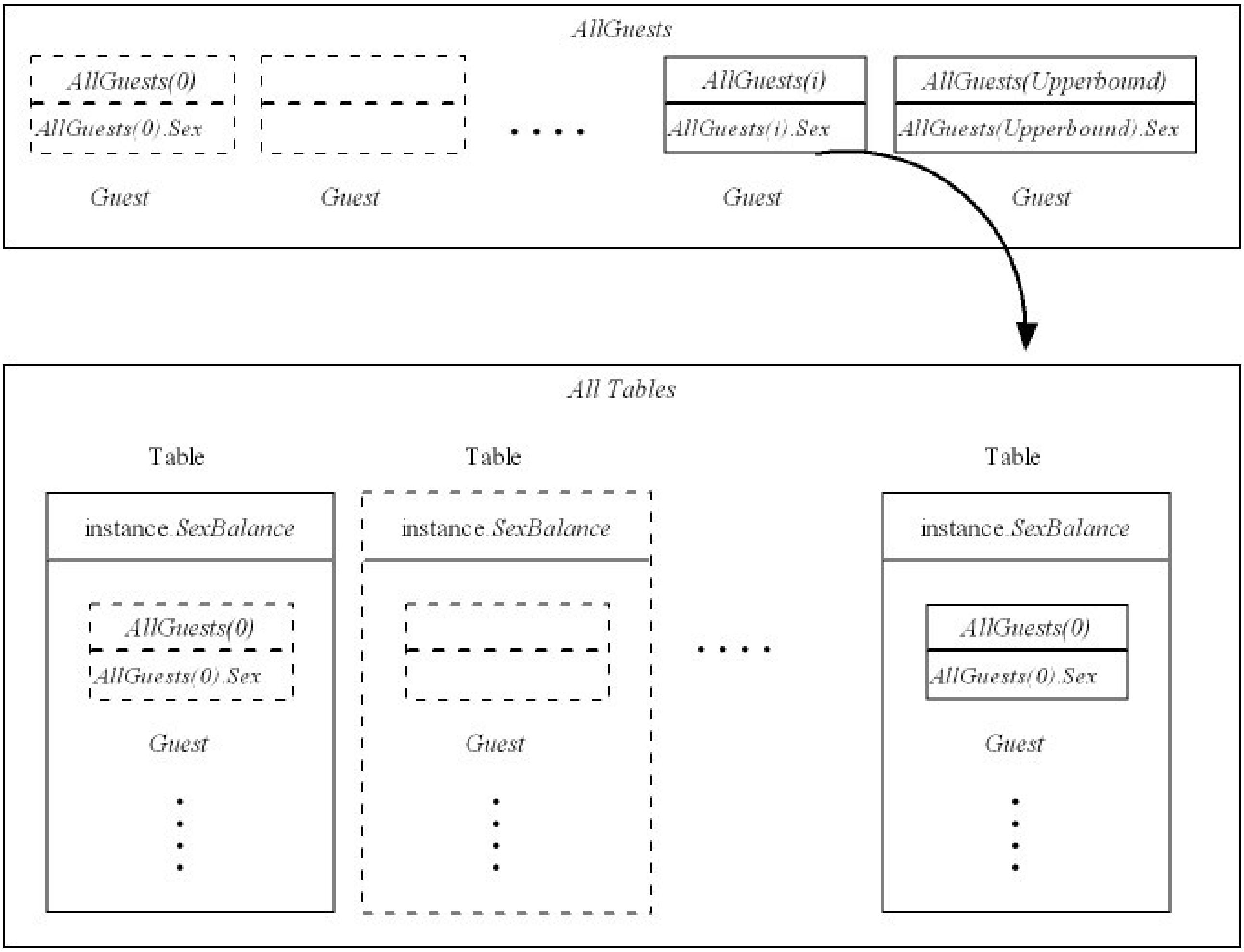
```
numSeat = Math.Ceiling(AllGuests.Count / numTable)
```

Next, the algorithm allocates the sorted list of records into the tables. It starts from the beginning of the list, *AllGuests(0)*, it puts the *Guest* objects into the table collection objects, one by one, in ascending order of year of graduation. Once the number of *Guest* object in a table collection object reaches *numSeat*, the algorithm continues to add the list of *Guest* into a new, empty table collection object.

Meanwhile, every time a *Guest* is added into a table collection, the algorithm checks the *sex* property of the *Guest*, and add it to the *SexBalance* property of the table collection.

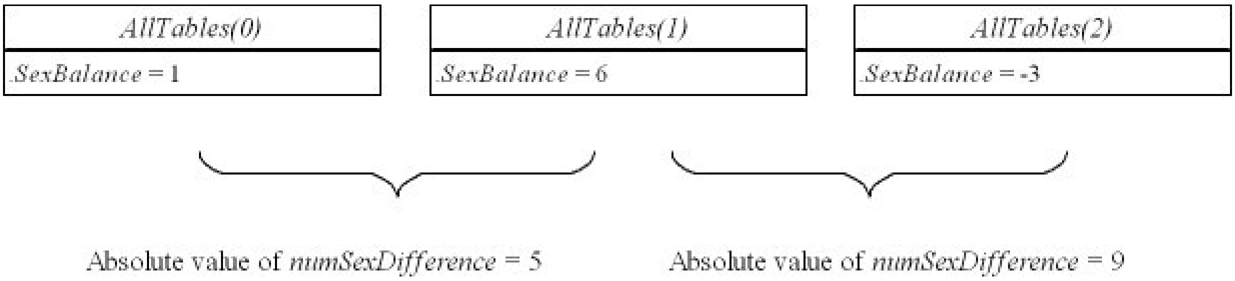
```
For i = 0 To AllTables.Count - 2
  numSexBalance = 0
  For j = 0 To numSeat - 1
    AllTables(i).Add(AllGuests(numSeat * i + j))
    numSexBalance += AllGuests(numSeat * i + j).Sex
  Next j
  AllTables(i).SexBalance = numSexBalance
Next i
numSexBalance = 0
For j = 0 To numSeat - (AllGuests.Count Mod numTable) - 1
  AllTables(i).Add(AllGuests(numSeat * i + j))
  numSexBalance += AllGuests(numSeat * i + j).Sex
Next
AllTables(i).SexBalance = numSexBalance
```

The manipulations on the objects can be visualized as in below :



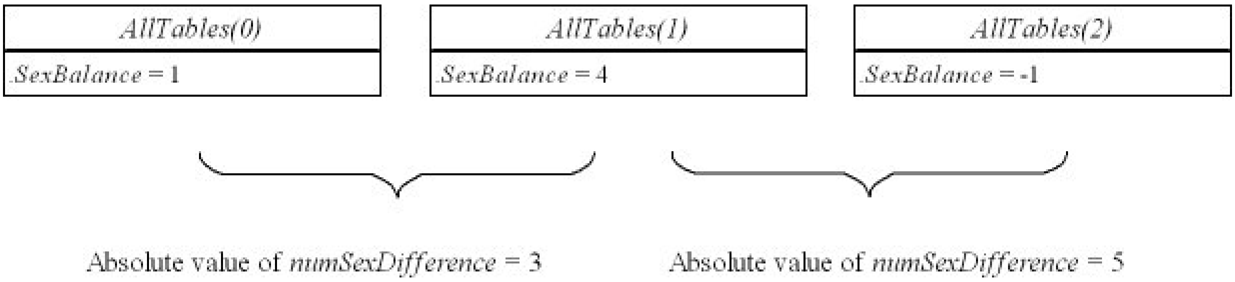
At this stage, the records are allocated to tables in the way such that, the seating plan perfectly groups participants with similar year of graduation together. However the seat allocation rule, balancing male and female participants should also be considered at the same time. This is done by swapping of records between tables with different critiria, accord to the sorting orientation selected by users.

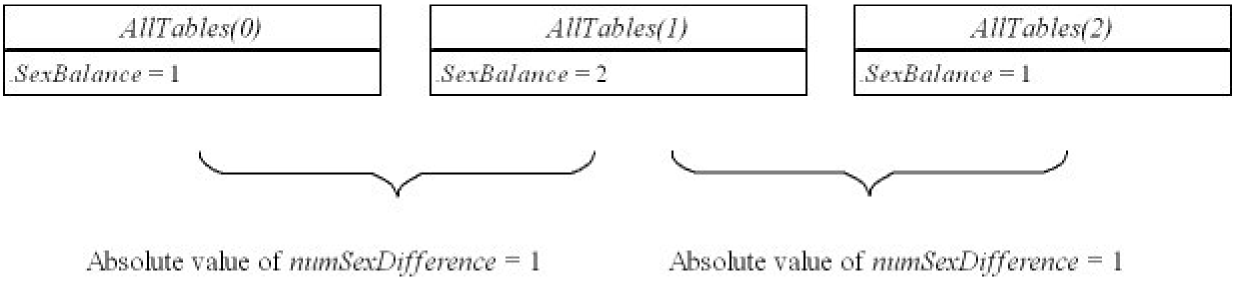
In this algorithm, the *numSexDifference* is used. It is defined as the "the difference of the difference of number of male and female between tables". As defined in section 3.1, to balance male and female participants in each table, *numSexDifference* have to be equalized between tables.



To recall, the *SexBalance* property indicates extra numbers of participants making the table being sexually unbalanced. A positive *SexBalance* points out the number of extra males, whereas a negative *SexBalance* points out the number of extra females.

As a result, if a table having a negative *SexBalance* swap a male with a female, to a table having a positive *SexBalance*, than the *SexBalance* of the former table can increase by 2, whereas the *SexBalance* of the latter table can decrease by 2. Totally the absolute value of *numSexDifference* can decrease by 4.





In principle, the algorithm evaluates the *numSexDifference* between each pairs of adjacent *AllTables(index)*. The algorithm would continously loop through all tables, and swap *Guests* between tables, until the absolute value of *numSexDifference* is smaller than or equal to three. Minimizing this sexual gradient means balancing male and female participants in each table, by definition.

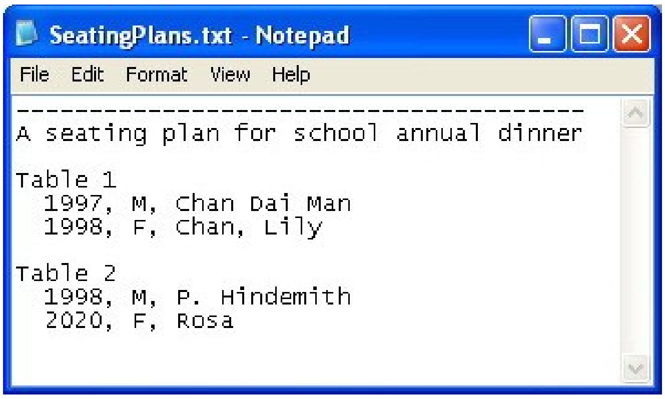
The final and most important point to note. The seating plans differentiate by different sorting orientation. The reason is that during the swapping process illustrated above, the sorting orientations adopts different criteria. If the user chooses to balance male and female participants in each table, then the algorithm would swap the *Guests* objects unconditionally until the *numSexDifference* between tables are minimized. But if the user prefer to group participants of similar year of graduation in each tables, then the swapping of *Guest* objects between tables would only be executed when the year of graduation of both *Guest* objects are the same. In this way the original year-of-graduation-oriented configuration can be preserved, thus satisfy the chosen orientation.

3.5 Design of the text-file appending Algorithm

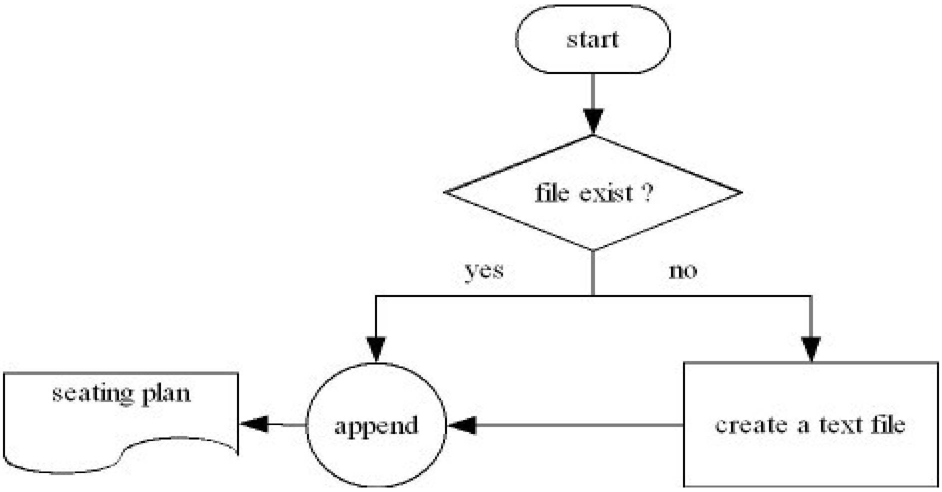
This algorithm executes after the record sorting algorithm. Assuming the records are sorted, this algorithm scans through all the records in the computer memory, and append them one by one to a text file in an appropriate output format. Then it saves the appended text file in the current directory of the program.

```
Public Sub SaveToTextFile()
  'save the records into a text file
  Dim file As System.IO.StreamWriter
  file = My.Computer.FileSystem.OpenTextFileWriter(My.Computer.FileSystem.CurrentDirectory &
"\SeatingPlans.txt", True)
  file.WriteLine("---------------------------------------")
  file.WriteLine("A seating plan for school annual dinner")
  For i = 0 To AllTables.Count - 1
    file.WriteLine()
    file.WriteLine("Table " & (i + 1).ToString())
    For Each oneGuest As GuestClass In AllTables(i)
      file.Write("  " & oneGuest.GradYear.ToString() & ", ")
      If oneGuest.Sex = 1 Then file.Write("M, ")
      If oneGuest.Sex = -1 Then file.Write("F, ")
      file.Write(oneGuest.Name)
      file.WriteLine()
    Next
 Next
  file.WriteLine()
  file.Close()
End Sub
```

This algorithm format seating plans by indentation. An example is presented as in follow :



```
---------------------------------------
A seating plan for school annual dinner

Table 1
  1997, M, Chan Dai Man
  1998, F, Chan, Lily

Table 2
  1998, M, P. Hindemith
  2020, F, Rosa
```



3.6 Implementation

Working with Data

The focal point of this program is the batch processing of the participants records. However this program does not work on its own. To help the school annual dinner registration, all the necessary raw data must first be collected. This can be done by phone, e-mail etc. Then the raw data have to be well-prepared. At least all the personal data match to the correct person. Then the source document have to be verified to eliminate all the data source errors, transcription errors and transposition errers. At this stage, the data are ready to be input into the program. But during the input process, the person should be very careful and verify the input after each entry is typed in. Finally the data are validated by the program. At this stage the data are assumed to be accurate and can be processed to generate output.

Help Actual Decisions

A major implementation of this program is to generate different seating plans, by specifying different number of tables required and try different sorting orientation. By doing so, users can compare different possible seating plans for a set of participants. Thus the program can help users to decide how the arrange the seats in the dinner, more than to look for how a decision looks like.

System Requirements

This program requires users to have a computer, monitor, keyboard, mouse, and running on Windows 98 or higher,

This program is developed with a rather new programing langauge. Users are recommended to run this program on Windows XP. In some cases, the program is still not working. Then users should update their Windows XP, and install the .NET Framework 2.0 as the runtime service provider for this program.

This program is best viewed at screen resolution 1024 * 768.

Chapter 4 : Program Testing and Evaluation

The testing process of this program is highly output dependent. Sometimes in testing or debugging, I only know the probable range of lines that hides a mistake. In this situation I will apply the trial and error methodology. And it is so often that I can get the expected output even I do not completely understand the code I have written.

4.1 Debugging

There has been once I encountered a logic error in my the program. I entered four records into the program, and it output three records to me. It is a serious data corruption. And the output is also completely unexpected. In this section I want to share my experience in handling a bug like this.

```
Seating Plans for school annual dinner

Table 1
  1897 Terry
  2005 Josef

Table 2
  2012 Maya
```

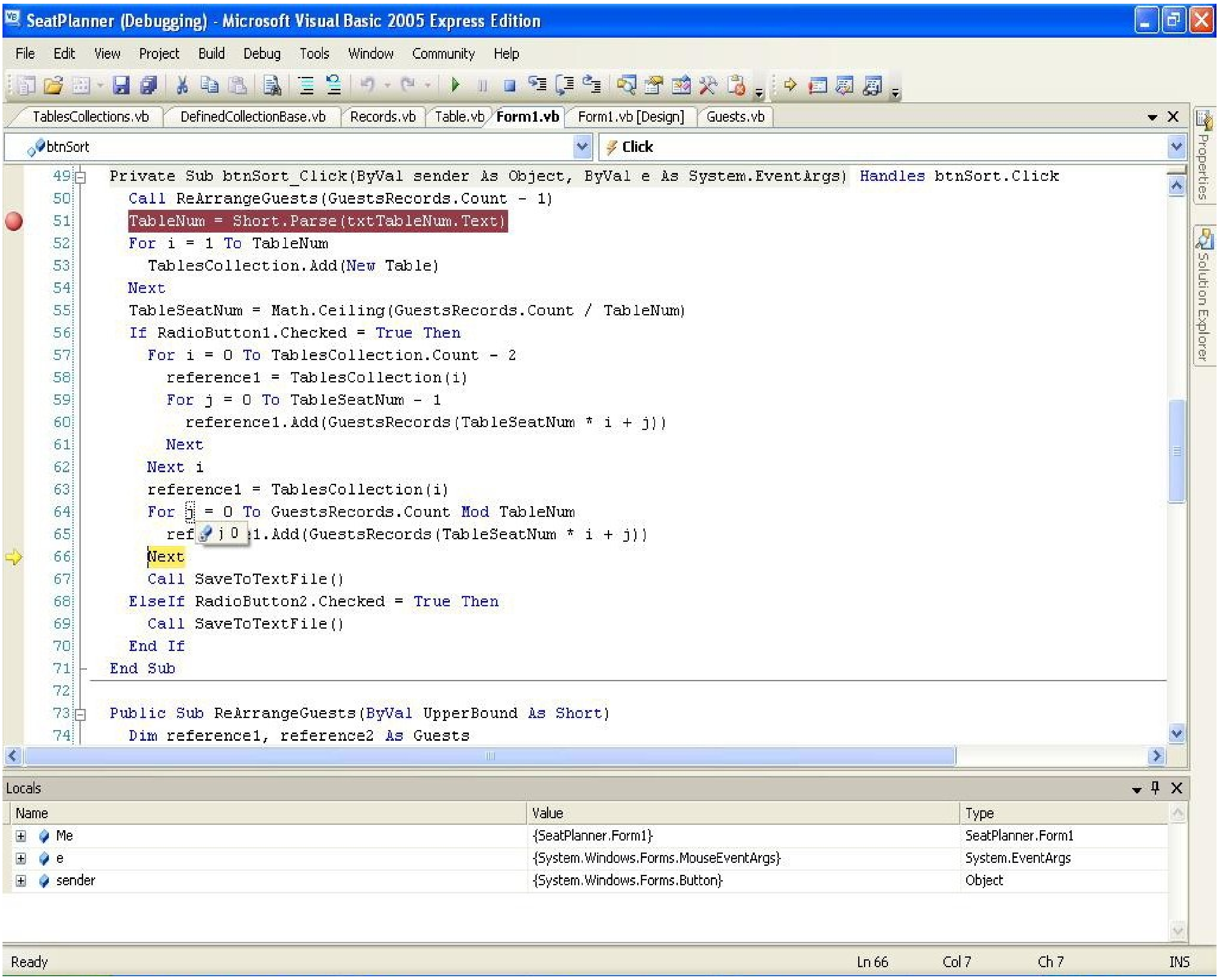
This was the unexpectedly output that I generated. The expected output should contain four records. But soon I discovered that except the missing record, the other three records were perfectly falled into my expectation. So I suspected that it was not the problem of the sorting algorithm. I suspect that the error occured at where *Guest* objects were being put into Table collection objects.

After that I typed in three additional records, this time the output was flawed in a more expectable way.

```
Seating Plans for school annual dinner

Table 1
  1897 Terry
  1924 Tina
  2005 Josef
  2005 John

Table 2
  2012 Maya
  2012 Tanya
```

This time, the flaw is about the same. Except the missing record, other records are all sorted into expected positions. More importantly, in both case the missing record is the same record (year of graduation of participant is 2020, the name of participant is Rosa).

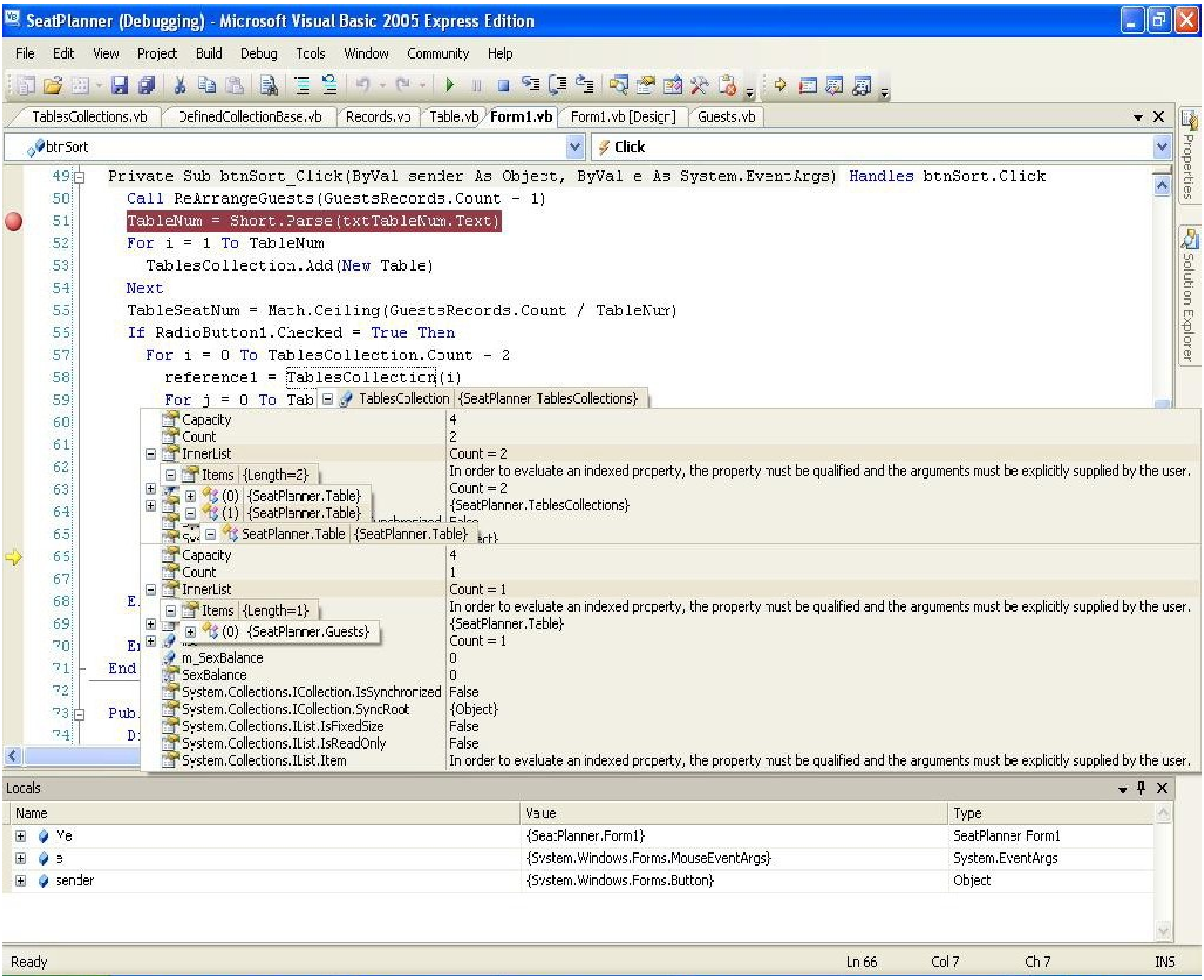
So I narrowed the range where the suspected mistake is located. Firstly I toggled a breakpoint in a position where the execution of the insertion sort is over because the insertion sort seemed to have no problem at all.


SeatPlanner (Debugging) - Microsoft Visual Basic 2005 Express Edition
File Edit View Project Build Debug Tools Window Community Help
TablesCollections.vb DefinedCollectionBase.vb Records.vb Table.vb Form1.vb Form1.vb [Design] Guests.vb
btnSort
Click
j 0
Properties
Solution Explorer
Locals
Ready
Ln 66
Col 7
Ch 7
INS

```vb
Private Sub btnSort_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles btnSort.Click
  Call ReArrangeGuests(GuestsRecords.Count - 1)
  TableNum = Short.Parse(txtTableNum.Text)
  For i = 1 To TableNum
    TablesCollection.Add(New Table)
  Next
  TableSeatNum = Math.Ceiling(GuestsRecords.Count / TableNum)
  If RadioButton1.Checked = True Then
    For i = 0 To TablesCollection.Count - 2
      reference1 = TablesCollection(i)
      For j = 0 To TableSeatNum - 1
        reference1.Add(GuestsRecords(TableSeatNum * i + j))
      Next
    Next i
    reference1 = TablesCollection(i)
    For j = 0 To GuestsRecords.Count Mod TableNum
      reference1.Add(GuestsRecords(TableSeatNum * i + j))
    Next
    Call SaveToTextFile()
  ElseIf RadioButton2.Checked = True Then
    Call SaveToTextFile()
  End If
End Sub

Public Sub ReArrangeGuests(ByVal UpperBound As Short)
  Dim reference1, reference2 As Guests
```

Name	Value	Type
Me	{SeatPlanner.Form1}	SeatPlanner.Form1
e	{System.Windows.Forms.MouseEventArgs}	System.EventArgs
sender	{System.Windows.Forms.Button}	Object

In the debugging process, I stepped into every statements to trace the state of variable values, the properties of the objects, and the configuration of the collection objects.


SeatPlanner (Debugging) - Microsoft Visual Basic 2005 Express Edition
File Edit View Project Build Debug Tools Window Community Help
TablesCollections.vb DefinedCollectionBase.vb Records.vb Table.vb Form1.vb Form1.vb [Design] Guests.vb
btnSort
Click
Properties
Solution Explorer

```
Private Sub btnSort_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles btnSort.Click
  Call ReArrangeGuests(GuestsRecords.Count - 1)
  TableNum = Short.Parse(txtTableNum.Text)
  For i = 1 To TableNum
    TablesCollection.Add(New Table)
  Next
  TableSeatNum = Math.Ceiling(GuestsRecords.Count / TableNum)
  If RadioButton1.Checked = True Then
    For i = 0 To TablesCollection.Count - 2
      reference1 = TablesCollection(i)
      For j = 0 To Tab
```

```
TablesCollection {SeatPlanner.TablesCollections}
  Capacity                      4
  Count                         2
  InnerList                     Count = 2
    Items {Length=2}            In order to evaluate an indexed property, the property must be qualified and the arguments must be explicitly supplied by the user.
      (0) {SeatPlanner.Table}   Count = 2
      (1) {SeatPlanner.Table}   {SeatPlanner.TablesCollections}
        SeatPlanner.Table {SeatPlanner.Table}
          Capacity                      4
          Count                         1
          InnerList                     Count = 1
            Items {Length=1}            In order to evaluate an indexed property, the property must be qualified and the arguments must be explicitly supplied by the user.
              (0) {SeatPlanner.Guests}  {SeatPlanner.Table}
                                        Count = 1
          m_SexBalance                  0
          SexBalance                    0
          System.Collections.ICollection.IsSynchronized  False
          System.Collections.ICollection.SyncRoot        {Object}
          System.Collections.IList.IsFixedSize           False
          System.Collections.IList.IsReadOnly            False
          System.Collections.IList.Item                  In order to evaluate an indexed property, the property must be qualified and the arguments must be explicitly supplied by the user.
```

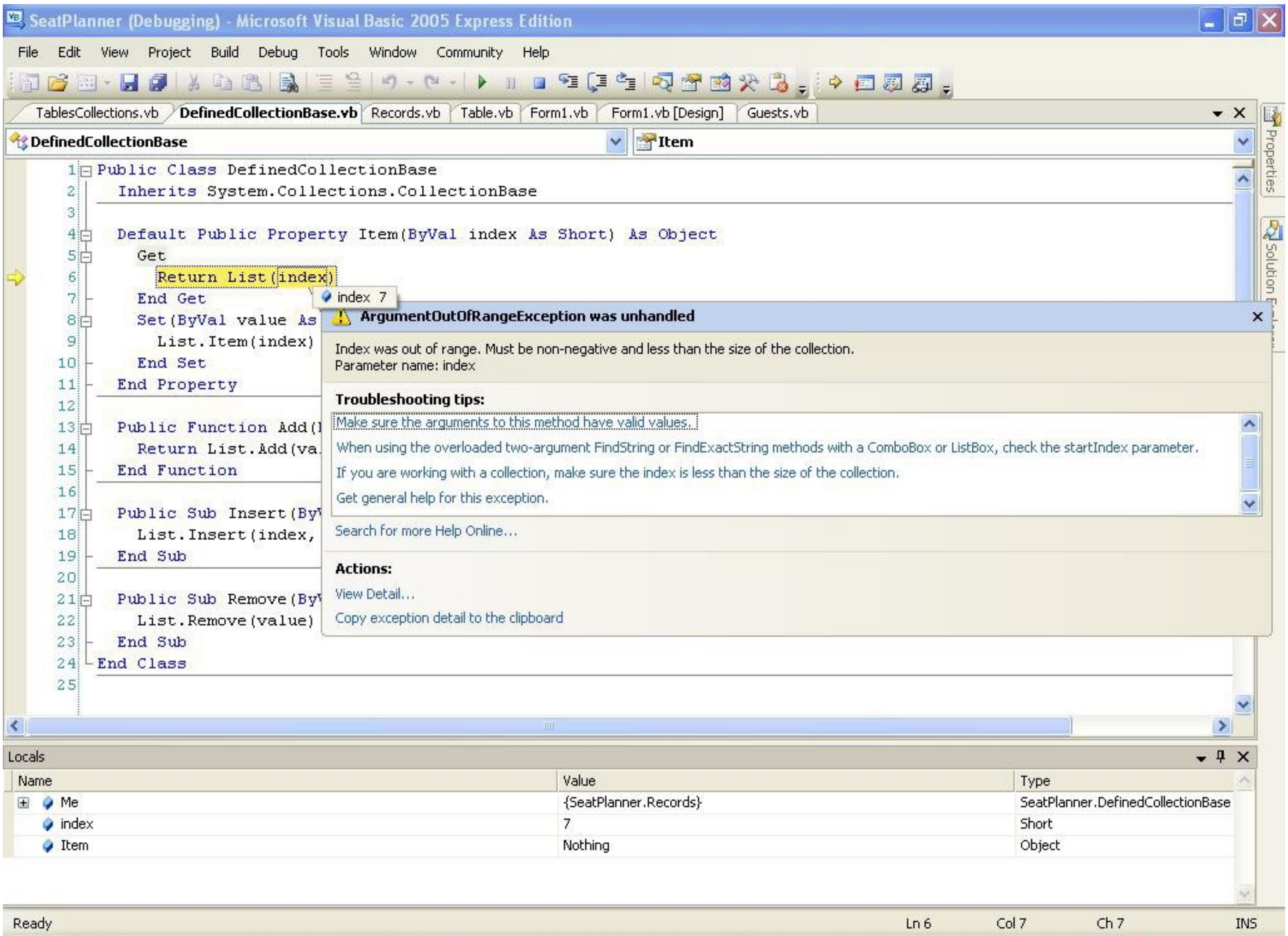
```
    E
    E
  End
  Pub
    Di
```

Locals

Name	Value	Type
Me	{SeatPlanner.Form1}	SeatPlanner.Form1
e	{System.Windows.Forms.MouseEventArgs}	System.EventArgs
sender	{System.Windows.Forms.Button}	Object

Not only the variables, but I checked every collection objects, and every items in the collections, to see if they behave in the same way that I expect during run-time. Fortunately the VB2005 Integrated Development Environment provided me an organized way view the nested data structure.

In the process, the integrated development environment discovered another error in the program. Fortunately this time the development environment could direct many resources to aid me. I knew what the bug was and handled it at once.



```
Public Class DefinedCollectionBase
  Inherits System.Collections.CollectionBase

  Default Public Property Item(ByVal index As Short) As Object
    Get
      Return List(index)
    End Get
    Set(ByVal value As
      List.Item(index)
    End Set
  End Property

  Public Function Add(I
    Return List.Add(va
  End Function

  Public Sub Insert(By
    List.Insert(index,
  End Sub

  Public Sub Remove(By
    List.Remove(value)
  End Sub
End Class
```

ArgumentOutOfRangeException was unhandled

Index was out of range. Must be non-negative and less than the size of the collection.
Parameter name: index

Troubleshooting tips:

Make sure the arguments to this method have valid values.

When using the overloaded two-argument FindString or FindExactString methods with a ComboBox or ListBox, check the startIndex parameter.

If you are working with a collection, make sure the index is less than the size of the collection.

Get general help for this exception.

Search for more Help Online...

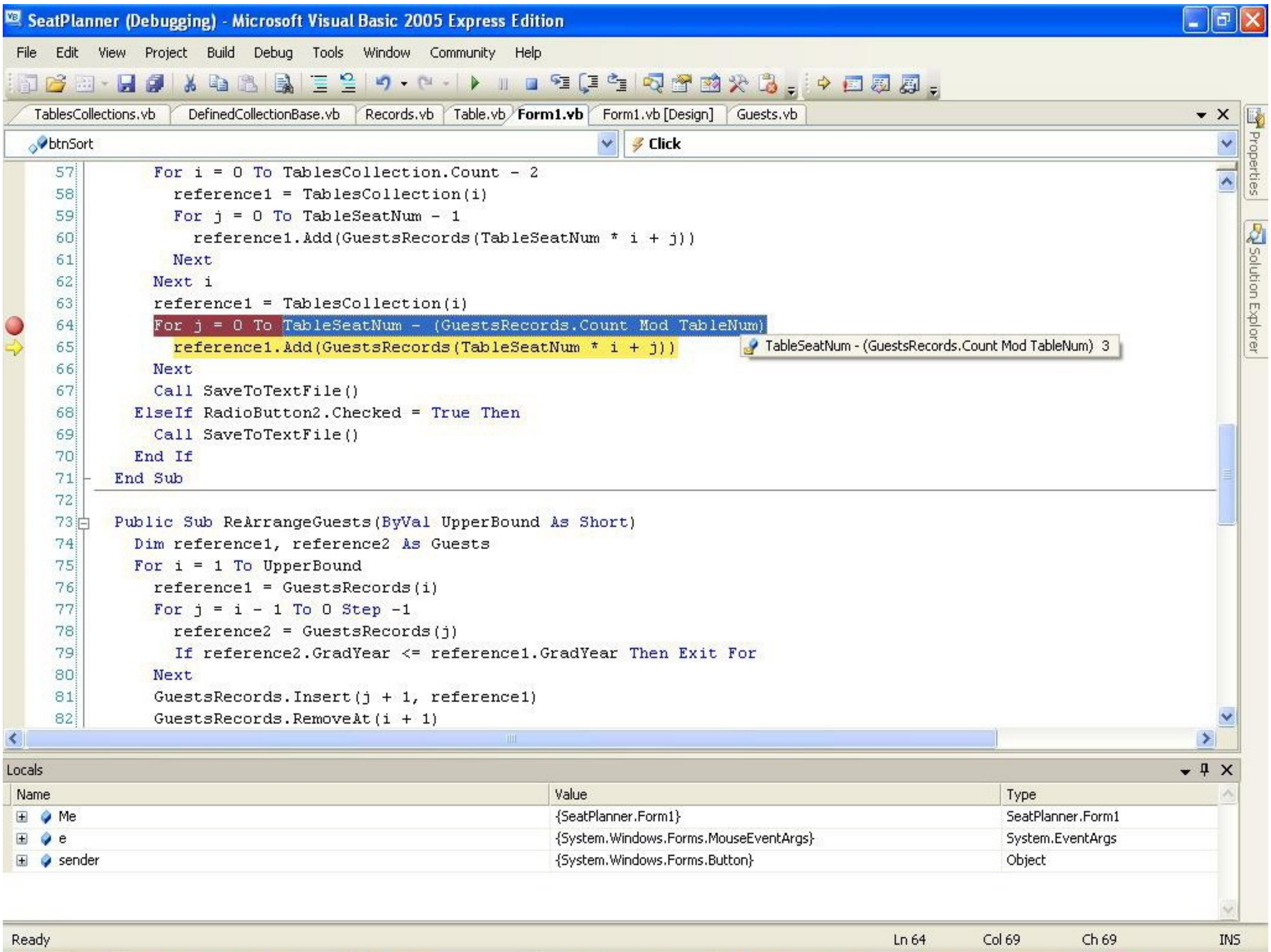
Actions:

View Detail...

Copy exception detail to the clipboard

Locals

Name	Value	Type
⊞ Me	{SeatPlanner.Records}	SeatPlanner.DefinedCollectionBase
index	7	Short
Item	Nothing	Object


SeatPlanner (Debugging) - Microsoft Visual Basic 2005 Express Edition
File Edit View Project Build Debug Tools Window Community Help
TablesCollections.vb DefinedCollectionBase.vb Records.vb Table.vb Form1.vb Form1.vb [Design] Guests.vb
btnSort
Click
Properties
Solution Explorer

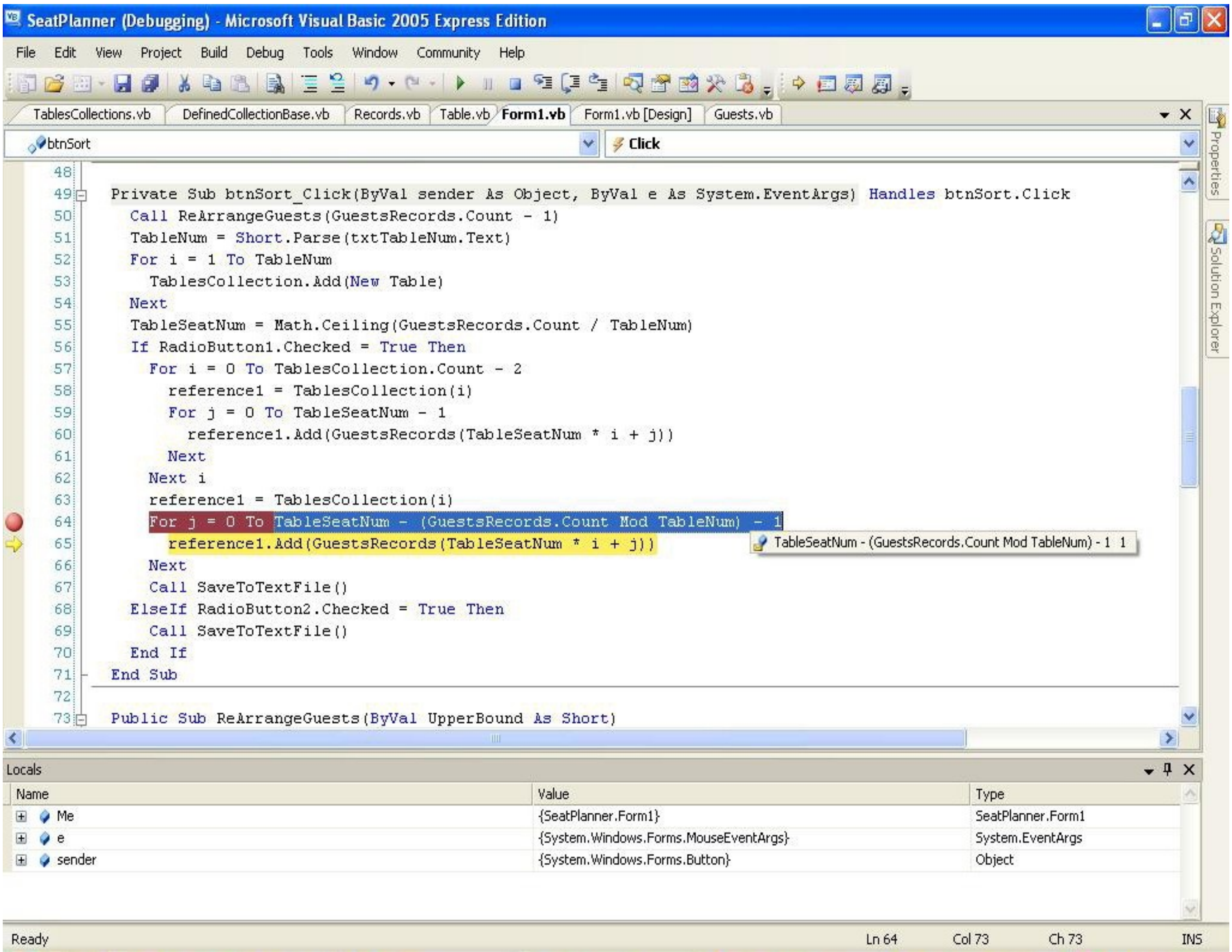
```
    For i = 0 To TablesCollection.Count - 2
      reference1 = TablesCollection(i)
      For j = 0 To TableSeatNum - 1
        reference1.Add(GuestsRecords(TableSeatNum * i + j))
      Next
    Next i
    reference1 = TablesCollection(i)
    For j = 0 To TableSeatNum - (GuestsRecords.Count Mod TableNum)
      reference1.Add(GuestsRecords(TableSeatNum * i + j))
    Next
    Call SaveToTextFile()
  ElseIf RadioButton2.Checked = True Then
    Call SaveToTextFile()
  End If
End Sub

Public Sub ReArrangeGuests(ByVal UpperBound As Short)
  Dim reference1, reference2 As Guests
  For i = 1 To UpperBound
    reference1 = GuestsRecords(i)
    For j = i - 1 To 0 Step -1
      reference2 = GuestsRecords(j)
      If reference2.GradYear <= reference1.GradYear Then Exit For
    Next
    GuestsRecords.Insert(j + 1, reference1)
    GuestsRecords.RemoveAt(i + 1)
```

TableSeatNum - (GuestsRecords.Count Mod TableNum) 3

Locals

Name	Value	Type
Me	{SeatPlanner.Form1}	SeatPlanner.Form1
e	{System.Windows.Forms.MouseEventArgs}	System.EventArgs
sender	{System.Windows.Forms.Button}	Object

After some time, I stepped into an iteration which is responsible for putting the *Guest* objects into the Table collecion objects. I found that the final value of the iteration was 3 while it should be 2 according to my dry run on another paper.



```
Private Sub btnSort_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles btnSort.Click
  Call ReArrangeGuests(GuestsRecords.Count - 1)
  TableNum = Short.Parse(txtTableNum.Text)
  For i = 1 To TableNum
    TablesCollection.Add(New Table)
  Next
  TableSeatNum = Math.Ceiling(GuestsRecords.Count / TableNum)
  If RadioButton1.Checked = True Then
    For i = 0 To TablesCollection.Count - 2
      reference1 = TablesCollection(i)
      For j = 0 To TableSeatNum - 1
        reference1.Add(GuestsRecords(TableSeatNum * i + j))
      Next
    Next i
    reference1 = TablesCollection(i)
    For j = 0 To TableSeatNum - (GuestsRecords.Count Mod TableNum) - 1
      reference1.Add(GuestsRecords(TableSeatNum * i + j))
    Next
    Call SaveToTextFile()
  ElseIf RadioButton2.Checked = True Then
    Call SaveToTextFile()
  End If
End Sub

Public Sub ReArrangeGuests(ByVal UpperBound As Short)
```

TableSeatNum - (GuestsRecords.Count Mod TableNum) - 1 1

Locals

Name	Value	Type
Me	{SeatPlanner.Form1}	SeatPlanner.Form1
e	{System.Windows.Forms.MouseEventArgs}	System.EventArgs
sender	{System.Windows.Forms.Button}	Object

Ready Ln 64 Col 73 Ch 73 INS

Then I just modified that final value of the iteration by substracting 1 from it.

The testing result was satisfying. I entered the seven records as same as before. The following was the seating plan output :

```
Seating Plans for school annual dinner

Table 1
  1897 Terry
  1924 Tina
  2005 Josef
  2005 John

Table 2
  2012 Maya
  2012 Tanya
  2020 Rosa
```

The seven records were sorted according to year of graduation. The last record “2020, Rosa” finally appeared in the expected location. This indicates the suspected error had been eliminated.

To confirm that the errors were gone, I input a larger data set. Again the seating it generated was sorted neatly and correctly matched my expected output. By using tools provided by the integrated development environment and some observations, a bug was eliminated.

```
Seating Plans for school annual dinner

Table 1
  1897 Terry
  1900 Kevin
  1923 Pauline
  1924 Tina

Table 2
  2005 Josef
  2005 John
  2006 Tom
  2012 Maya

Table 3
  2012 Tanya
  2019 Kelvin
```

4.2 Testing and Evaluation

The aim of testing and evaluation is to see if the algorithm works as expectation. In this section I would only present the result of the test on the record sorting algorithm, since the records manipulation algorithms can only be tested during run-time.

To test the record sorting algorithm, I have formulated a testing plan. It is a list containing 19 records to be sort. I input them into the program, and then sort them using different number of tables required, and tried the two sorting orientation respectively.

The list of participants to presented as in below :

Name	Year of Graduation	Sex	Name	Year of Graduation	Sex
Lily Lau	1987	F	Mickey Cheung	1988	M
Niki Chau	1988	F	Michael Fong	1988	M
Michael Tzang	1987	M	Adam Wong	1989	M
Adam Li	1990	M	Liza Lui	1990	F
Denny Li	1990	M	Sam Hu	1988	M
Loretta Leung	1992	F	Ben Wong	1993	M
Lydia Siu	1993	F	Crystal Wong	1994	F
George Brown	1987	M	Tom Lee	1990	M
Peter Wong	1988	M	Eddie Chan	1991	M
Sammy Zheng	1994	F			

With 19 records input, the testing result was frustrating. In both seating plans there are some records missing.

```
--------------------------------------
A seating plan for school annual dinner

Table 1
  1987, F, Lily Lau
  1987, M, Michael Tzang
  1987, M, George Brown
  1988, M, Mickey Cheung
  1988, M, Peter Wong

Table 2
  1988, F, Niki Chau
  1988, M, Michael Fong
  1988, M, Sam Hu
  1989, M, Adam Wong
  1990, M, Adam Li

Table 3
  1990, M, Denny Li
  1990, F, Liza Lui
  1990, M, Tom Lee
  1991, M, Eddie Chan
  1992, F, Loretta Leung

Table 4
  1993, F, Lydia Siu
  1993, M, Ben Wong
```

```
--------------------------------------
A seating plan for school annual dinner

Table 1
  1987, F, Lily Lau
  1987, M, Michael Tzang
  1987, M, George Brown
  1988, M, Peter Wong

Table 2
  1988, F, Niki Chau
  1988, M, Mickey Cheung
  1988, M, Michael Fong
  1988, M, Sam Hu

Table 3
  1989, M, Adam Wong
  1990, M, Adam Li
  1990, M, Denny Li
  1990, F, Liza Lui

Table 4
  1990, M, Tom Lee
  1991, M, Eddie Chan
  1992, F, Loretta Leung
  1993, F, Lydia Siu

Table 5
```

So I added the 20th record, debugger, 1990, F, to see what will happen :

```
--------------------------------------
A seating plan for school annual dinner

Table 1
  1987, F, Lily Lau
  1987, M, Michael Tzang
  1987, M, George Brown
  1988, M, Mickey Cheung
  1988, M, Peter Wong

Table 2
  1988, F, Niki Chau
  1988, M, Michael Fong
  1988, M, Sam Hu
  1989, M, Adam Wong
  1990, F, Liza Lui

Table 3
  1990, M, Denny Li
  1990, M, Adam Li
  1990, M, Tom Lee
  1990, F, debugger
  1992, F, Loretta Leung

Table 4
  1991, M, Eddie Chan
  1993, F, Lydia Siu
  1993, M, Ben Wong
  1994, F, Sammy Zheng
  1994, F, Crystal Wong

--------------------------------------
A seating plan for school annual dinner

Table 1
  1987, F, Lily Lau
  1987, M, Michael Tzang
  1987, M, George Brown
  1988, M, Peter Wong

Table 2
  1988, F, Niki Chau
  1988, M, Mickey Cheung
  1988, M, Michael Fong
  1988, M, Sam Hu

Table 3
  1989, M, Adam Wong
  1990, M, Adam Li
  1990, M, Denny Li
  1990, F, Liza Lui

Table 4
  1990, M, Tom Lee
  1990, F, debugger
  1991, M, Eddie Chan
  1992, F, Loretta Leung

Table 5
  1993, F, Lydia Siu
  1993, M, Ben Wong
  1994, F, Sammy Zheng
  1994, F, Crystal Wong
```

The result is not satisfying too. Though there is no records missing, but the algorithm behaves entirely different compare to that when handling ten records or less. The sorting orientation does not make much difference too. It is quite disappointing.

Chapter 5 : Conclusion and Discussion

Implementation Conclusion

The objective of the project is to write a computer program, that is able to generate a seating plan for the school annual dinner registration. The implementation process shows that this program can achieve the objective with just a few minorities to be improved. The algorithm design is not that powerful, but the program is the one which users can find it comfortable to work with.

Program Limitation

The record sorting algorithm of the program is a bit limited. The whole algorithm design may be somewhat intuituve or involves redundant steps. It can furthur be improved and simplified. Also more participants' data field and sorting parameters can be taken into consideration to sort out a seating plan, so as to make the program be more robust and adaptive.

As shown through implementation , the input of participants records into the program can be quite demanding if there are too many participants attending the school annual dinner. Users may feel monotonus or make input errors at a higher chance. Likewise, if the number of participants records exceeds a certain level, then the seating plan sorted would be inclined to group participants of similar year of graduation, even with the preference of balancing male and female in each table be chosen. All in all, the sorting algorithm is limited to handle a lesser amount of participants.

Program Accomplishment

However the final release of this program fairly lives up to my expectation. The design of the user interface is simple and clear. Users can learn and use the program easily. The final release also presents no run-time errors in any tested circumstances. During the re-tests, the data input and sorting parameters validation.algorithm performed well to prevent errors to occur. Finally, the record saving algorithm appends the seating plan in an appropriate format, regardless the number of participants and tables required.

Furthur Development

Web-based User Interface

The most dominant example of a web-based user interface is a web page. By such an implementation, users need not to set up a program in their harddrives. Also the program can be easily distributed by forwarding a URL. The drawback is that data security is weaker on the net.

Web-based Data Collection

A typical annual school dinner would have over hundreds of participants attending. So during implementation, records have to be typed in over hundreds of times as well. The implementation can be inefficient. The advantage of web-based data collection is that it saves the intended users a lot of typing work. Registrations can take place in the Internet. An electronic registration form can be designed. Then participants fill in the form and submit it through e-mail, instant messages, and so forth. This suffices that intended users can type much less in the program.

Database and Real Time Registration

The advantage of using database emerges when the number of records increase. As the program expands, a database engine can provide effective data management. Also the program and the data are separated. So that it provides a greater flexibility in control. A real time registration system can also be developed. Participants can connect to a database through the Internet. So that they can view the real time seat allocation status online, and then registered for a perferred seat in the dinner.

Afterthought

After this project, I have learnt to plan before the work actually starts. That is to say, to firstly define the problem, analysis it, set an objective, and gradually work out the solution. This altitude can be applied in almost any other works. I have learnt to set a much tighter working schedule, because during this project there were many accidents hindering me to finish it on time.

Also I realized that programming is a very professional and laborius job. Even for this small scale project, I have taken weeks for data and IT tools research, to come up with design ideas, transform the ideas to codes, to debug and re-test the program, and to build up this accompanied documentation. I cannot imagine the efforts involved in building large scale projects, like the programming program, Visual Studio itself. I also wonder how generous are the programmers sharing source codes, developing open source freewares on the World Wide Web. After this project, I would appreciate more and pay more respects to the programmers in the world.

Appendix

List of employed Variables and Classes-

Variable	Data Type	Role
class Form1		
i	short	global loop counter
j	short	global loop counter
k	short	global loop counter
OKFlag	boolean	global true / false counter
numTable	short	local variable that stores number of tables required
numSeat	short	local variable that stores number of seat in each table
numSexBalance	short	local variable that stores the unbalanced gender state of a table
numSexDifference	short	local variable that stores
tempName	string	local variable that stores a name of participant
temp1	short	local variable that stores a year of graduation of participant
temp2	short	local variable that stores a sex of participant
class CollectionBaseClass		
class AllTablesClass		
class TableClass		
m_SexBalance	short	private field that stores the SexBalance property for instances of the class
class AllGuestsClass		
class GuestsClass		
m_Name	string	private field that stores the Name property for instances of the class
m_GradYear	short	private field that stores the GradYear property for instances of the class
m_Sex	short	private field that stores the Sex property for instances of the class

Source Code -

```
Public Class GuestClass
  Private m_Name As String
  Private m_GradYear As Short
  Private m_Sex As Short

  Public Property Name() As String
    Get
      Return m_Name
    End Get
    Set(ByVal value As String)
      m_Name = value
    End Set
  End Property

  Public Property GradYear() As Short
    Get
      Return m_GradYear
    End Get
    Set(ByVal value As Short)
      m_GradYear = value
    End Set
  End Property

  Public Property Sex() As Short
    Get
      Return m_Sex
    End Get
    Set(ByVal value As Short)
      m_Sex = value
    End Set
  End Property
End Class


Public Class CollectionBaseClass
  Inherits System.Collections.CollectionBase

  Default Public Property Item(ByVal index As Short) As Object
    Get
      Return List(index)
    End Get
    Set(ByVal value As Object)
      List.Item(index) = value
    End Set
  End Property

  Public Function Add(ByVal value As Object) As Short
    Return List.Add(value)
  End Function

  Public Sub Insert(ByVal index As Short, ByVal value As Object)
    List.Insert(index, value)
  End Sub

  Public Sub Remove(ByVal value As Object)
    List.Remove(value)
  End Sub
End Class
```

```
Public Class AllGuestsClass
  Inherits SeatPlanner.CollectionBaseClass
End Class


Public Class TableClass
  Inherits SeatPlanner.CollectionBaseClass
  Private m_SexBalance As Short

  Public Property SexBalance() As Short
    Get
      Return m_SexBalance
    End Get
    Set(ByVal value As Short)
      m_SexBalance = value
    End Set
  End Property
End Class


Public Class AllTablesClass
  Inherits SeatPlanner.CollectionBaseClass
End Class


Public Class Form1
  Dim AllGuests As New AllGuestsClass
  Dim AllTables As New AllTablesClass
  Dim i, j, k As Short
  Dim OKFlag As Boolean

#Region "Manipulation Buttons"

  Private Sub btnClearTextBoxes_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles
btnClearTextBoxes.Click
    Call ClearTextBoxes()
  End Sub

  Private Sub btnSave_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles btnSave.Click

    'participants data input validation
    Dim temp1, temp2 As Short
    If txtName.Text = "" Then
      MessageBox.Show("Please input the name of the participant", "Invalid name of participant",
MessageBoxButtons.OK, MessageBoxIcon.Error)
      txtName.Focus()
      Exit Sub
    End If
    For i = 0 To AllGuests.Count - 1
      If txtName.Text = AllGuests(i).Name Then
        MessageBox.Show("There is a record with the same name" & vbCrLf & "Please input another unique name of
participant", "Invalid name of participant", MessageBoxButtons.OK, MessageBoxIcon.Error)
        txtName.Focus()
        Exit Sub
      End If
    Next
    Try
      temp1 = Short.Parse(txtGradYear.Text)
      If (temp1 < 1000) Or (temp1 > 3000) Then Throw New FormatException
    Catch ex As FormatException
      MessageBox.Show("Please input a whole number ranging from 1000 to 3000", "Invalid year of graduation of
participant", MessageBoxButtons.OK, MessageBoxIcon.Error)
      txtGradYear.Text = ""
      txtGradYear.Focus()
      Exit Sub
    End Try
    If (txtSex.Text.ToUpper = "M") Or (txtSex.Text.ToUpper = "MALE") Then
      temp2 = 1
    ElseIf (txtSex.Text.ToUpper = "F") Or (txtSex.Text.ToUpper = "FEMALE") Then
      temp2 = -1
    Else
      MessageBox.Show("Please input ""m"" or ""f"" for male or female respectively", "Invalid sex of participant",
MessageBoxButtons.OK, MessageBoxIcon.Error)
      txtSex.Text = ""
      txtSex.Focus()
      Exit Sub
    End If

    'create and save a new record
    Select Case AllGuests.Count
      Case 0 To 999
        Dim Guest As New GuestClass()
        Guest.Name = txtName.Text
        Guest.GradYear = temp1
        Guest.Sex = temp2
        AllGuests.Add(Guest)
        Label4.Text = "< " & AllGuests.Count.ToString() & " records saved ! >"
        Call ClearTextBoxes()
      Case 1000
        MessageBox.Show("You haved reach the maximum number of records to be saved" & vbCrLf & "The new record
cannot be saved", "This program does not support records saved to be over 1000", MessageBoxButtons.OK,
MessageBoxIcon.Error)
    End Select
  End Sub

  Private Sub btnAmend_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles btnAmend.Click
    Dim temp1, temp2 As Short
    If txtName.Text = "" Then
      MessageBox.Show("Please input the name of the participant", "Invalid name of participant",
MessageBoxButtons.OK, MessageBoxIcon.Error)
      txtName.Focus()
      Exit Sub
    End If
    For i = 0 To AllGuests.Count - 1
      If AllGuests(i).Name = txtName.Text Then
        Try
          temp1 = Short.Parse(txtGradYear.Text)
          If (temp1 < 1000) Or (temp1 > 3000) Then Throw New FormatException
        Catch ex As FormatException
          MessageBox.Show("Please input a whole number ranging from 1000 to 3000", "Invalid year of graduation of
participant", MessageBoxButtons.OK, MessageBoxIcon.Error)
          txtGradYear.Text = ""
          txtGradYear.Focus()
          Exit Sub
        End Try
        If (txtSex.Text.ToUpper = "M") Or (txtSex.Text.ToUpper = "MALE") Then
          temp2 = 1
        ElseIf (txtSex.Text.ToUpper = "F") Or (txtSex.Text.ToUpper = "FEMALE") Then
          temp2 = -1
        Else
          MessageBox.Show("Please input ""m"" or ""f"" for male or female respectively", "Invalid sex of participant",
MessageBoxButtons.OK, MessageBoxIcon.Error)
          txtSex.Text = ""
          txtSex.Focus()
          Exit Sub
        End If
        With AllGuests(i)
          .GradYear = temp1
          .Sex = temp2
        End With
        Label4.Text = "< record amended ! >"
        Call ClearTextBoxes()
        Exit Sub
      End If
    Next
    MessageBox.Show("Record not found", "Record not found", MessageBoxButtons.OK,
MessageBoxIcon.Information)
    txtName.Focus()
  End Sub

  Private Sub btnCleanOneGuest_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnCleanOneGuest.Click
    Dim tempName As String
    tempName = InputBox("Please input the name of the participant", "Record to delete")
    For i = 0 To AllGuests.Count - 1
      If AllGuests(i).Name = tempName Then
        AllGuests.RemoveAt(i)
        Label4.Text = "< record deleted ! >"
        txtName.Focus()
        Exit Sub
      End If
    Next i
    MessageBox.Show("Record not found", "Record not found", MessageBoxButtons.OK,
MessageBoxIcon.Information)
    txtName.Focus()
  End Sub

  Private Sub btnCleanAllGuests_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnCleanAllGuests.Click
    'remove all participants record from the memory
    Dim result As DialogResult
    result = MessageBox.Show("All participants records in the memory will be lost !" & vbCrLf & "Continue ?",
"Confirm Records Delete", MessageBoxButtons.YesNo, MessageBoxIcon.Warning)
    If result = Windows.Forms.DialogResult.Yes Then
      If AllGuests.Count > 0 Then
        AllGuests.Clear()
        Label4.Text = "< all records are deleted ! >"
        txtName.Focus()
      ElseIf AllGuests.Count = 0 Then
        MessageBox.Show("There are no participants records to delete", "No records found", MessageBoxButtons.OK,
MessageBoxIcon.Stop)
      End If
    End If
  End Sub

#End Region
```

```
  Private Sub btnSort_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles btnSort.Click
    Dim numTable, numSeat As Short
    Dim numSexBalance, numSexDifference As Short

    'parameters validation
    If AllGuests.Count = 0 Then
      MessageBox.Show("Please input at least one participant record", "No participant record for sorting", MessageBoxButtons.OK, MessageBoxIcon.Error)
      Exit Sub
    End If
    Try
      numTable = Short.Parse(txtTableNum.Text)
      If numTable > AllGuests.Count Then Throw New FormatException
    Catch ex As FormatException
      MessageBox.Show("Please input a whole number that is not larger than" & vbCrLf & "the number of participants to be sort", "Invalid number of tables", MessageBoxButtons.OK, MessageBoxIcon.Error)
      txtTableNum.Focus()
      Exit Sub
    End Try
    If Not (RadioButton1.Checked Or RadioButton2.Checked) Then
      MessageBox.Show("Please select one sorting orientation in each table", "No sorting orientation selected", MessageBoxButtons.OK, MessageBoxIcon.Error)
      Exit Sub
    End If

    'main sorting algorithm

    Call SortByYear(AllGuests.Count - 1)

    'create the specified number of tables required
    AllTables.Clear()
    For i = 1 To numTable
      AllTables.Add(New TableClass)
    Next

    numSeat = Math.Ceiling(AllGuests.Count / numTable)

    For i = 0 To AllTables.Count - 2
      numSexBalance = 0
      For j = 0 To numSeat - 1
        AllTables(i).Add(AllGuests(numSeat * i + j))
        numSexBalance += AllGuests(numSeat * i + j).Sex
      Next j
      AllTables(i).SexBalance = numSexBalance
    Next i
    numSexBalance = 0
    For j = 0 To numSeat - (AllGuests.Count Mod numTable) - 1
      AllTables(i).Add(AllGuests(numSeat * i + j))
      numSexBalance += AllGuests(numSeat * i + j).Sex
    Next
    AllTables(i).SexBalance = numSexBalance

    Do
      OKFlag = True
      For i = 0 To AllTables.Count - 2
        numSexDifference = AllTables(i).SexBalance - AllTables(i + 1).SexBalance
        If numSexDifference >= 3 Then
          For j = AllTables(i).Count - 1 To 0 Step -1
            If AllTables(i)(j).Sex = 1 Then Exit For
          Next
          For k = 0 To AllTables(i + 1).Count - 1
            If AllTables(i + 1)(k).Sex = -1 Then Exit For
          Next
          If RadioButton2.Checked Then
            Call SwapBetweenTables()
            AllTables(i).SexBalance += -2
            AllTables(i + 1).SexBalance += 2
          ElseIf RadioButton1.Checked Then
            If AllTables(i)(j).GradYear = AllTables(i + 1)(k).GradYear Then
              Call SwapBetweenTables()
              AllTables(i).SexBalance += -2
              AllTables(i + 1).SexBalance += 2
            ElseIf (i = AllTables.Count - 2) And Not (AllTables(i)(j).GradYear = AllTables(i + 1)(k).GradYear) Then
              Exit Do
            End If
          End If
          OKFlag = False
        ElseIf numSexDifference <= -3 Then
          For j = AllTables(i).Count - 1 To 0 Step -1
            If AllTables(i)(j).Sex = -1 Then Exit For
          Next j
          For k = 0 To AllTables(i + 1).Count - 1
            If AllTables(i + 1)(k).Sex = 1 Then Exit For
          Next k
          If RadioButton2.Checked Then
            Call SwapBetweenTables()
            AllTables(i).SexBalance += 2
            AllTables(i + 1).SexBalance += -2
          ElseIf RadioButton1.Checked Then
            If AllTables(i)(j).GradYear = AllTables(i + 1)(k).GradYear Then
              Call SwapBetweenTables()
              AllTables(i).SexBalance += 2
              AllTables(i + 1).SexBalance += -2
            ElseIf (i = AllTables.Count - 2) And Not (AllTables(i)(j).GradYear = AllTables(i + 1)(k).GradYear) Then
              Exit Do
            End If
          End If
          OKFlag = False
        End If
      Next i
    Loop Until OKFlag = True

    Call SaveToTextFile()

    'display system information after sorting
    If AllGuests.Count = 1 Then
      Label4.Text = "< " & AllGuests.Count.ToString() & " record in memory >"
    Else
      Label4.Text = "< " & AllGuests.Count.ToString() & " records in memory >"
    End If
    MessageBox.Show("The seating plan is saved" & vbCrLf & "in the same location as this program", "Seating plan
successfully built")
  End Sub

  Public Sub ClearTextBoxes()
    'empty all participatns data input textboxes
    txtName.Text = ""
    txtGradYear.Text = ""
    txtSex.Text = ""
    txtName.Focus()
  End Sub
  Public Sub SortByYear(ByVal UpperBound As Short)
    'rearrange participants records in ascending order of year of graduation by insertion sort
    For i = 1 To UpperBound
      For j = i - 1 To 0 Step -1
        If AllGuests(j).GradYear <= AllGuests(i).GradYear Then Exit For
      Next
      AllGuests.Insert(j + 1, AllGuests(i))
      AllGuests.RemoveAt(i + 1)
    Next i
  End Sub

  Public Sub SwapBetweenTables()
    AllTables(i + 1).Insert(k + 1, AllTables(i)(j))
    AllTables(i).Insert(j + 1, AllTables(i + 1)(k))
    AllTables(i).RemoveAt(j)
    AllTables(i + 1).RemoveAt(k)
  End Sub

  Public Sub SaveToTextFile()
    'save the records into a text file
    Dim file As System.IO.StreamWriter
    file = My.Computer.FileSystem.OpenTextFileWriter(My.Computer.FileSystem.CurrentDirectory &
"\SeatingPlans.txt", True)
    file.WriteLine("--------------------------------------")
    file.WriteLine("A seating plan for school annual dinner")
    For i = 0 To AllTables.Count - 1
      file.WriteLine()
      file.WriteLine("Table " & (i + 1).ToString())
      For Each oneGuest As GuestClass In AllTables(i)
        file.Write("  " & oneGuest.GradYear.ToString() & ", ")
        If oneGuest.Sex = 1 Then file.Write("M, ")
        If oneGuest.Sex = -1 Then file.Write("F, ")
        file.Write(oneGuest.Name)
        file.WriteLine()
      Next
    Next
    file.WriteLine()
    file.Close()
  End Sub

End Class
```