|  |  |  |
| --- | --- | --- |
|  | **FACULTY OF COMPUTING, ENGINEERING and SCIENCE** | Final mark awarded:\_\_\_\_\_ |

**Assessment Cover Sheet and Feedback Form 2017/18**

|  |  |  |  |
| --- | --- | --- | --- |
| Module Code:  CS2S564 | Module Title:  Event Driven & GUI Prog. | | Module Lecturer:  Keith Norris |
| Assessment Title and Tasks: Number List Manager (Set Tasks - not-time constrained 1) | | | Assessment No. 1 |
| No. of pages submitted in total including this page:  N/A | | | Word Count of submission  (if applicable): |
| Date Set:  27/10/2017 | | Submission Date:  15/12/2017 | Return Date:  2/2/2018 |

|  |  |
| --- | --- |
| ***Part A: Record of Submission (to be completed by Student)*** | |
| **Extenuating Circumstances**  If there are any exceptional circumstances that may have affected your ability to undertake or submit this assignment, make sure you contact the Advice Centre on your campus prior to your submission deadline. | |
| **Fit to sit policy**:  The University operates a fit to sit policy whereby you, in submitting or presenting yourself for an assessment, are declaring that you are fit to sit the assessment. You cannot subsequently claim that your performance in this assessment was affected by extenuating factors. | |
| **Plagiarism and Unfair Practice Declaration:**  By submitting this assessment, you declare that it is your own work and that the sources of information and material you have used (including the internet) have been fully identified and properly acknowledged as required (University Academic Integrity Regulations). Additionally, the work presented has not been submitted for any other assessment. You also understand that the Faculty reserves the right to investigate allegations of plagiarism or unfair practice which, if proven, could result in a fail in this assessment and may affect your progress. | |
| **Details of Submission:**  Note that all work handed in after the submission date and within 5 working days will be capped at 40%. No marks will be awarded if the assessment is submitted after the late submission date unless extenuating circumstances are applied for and accepted.  Work should be submitted as detailed in your student handbook. You are responsible for checking the method of submission. | |
| **You are required to acknowledge that you have read the above statements by writing your student number (s) in the box:** | Student Number(s):  **14729831** |

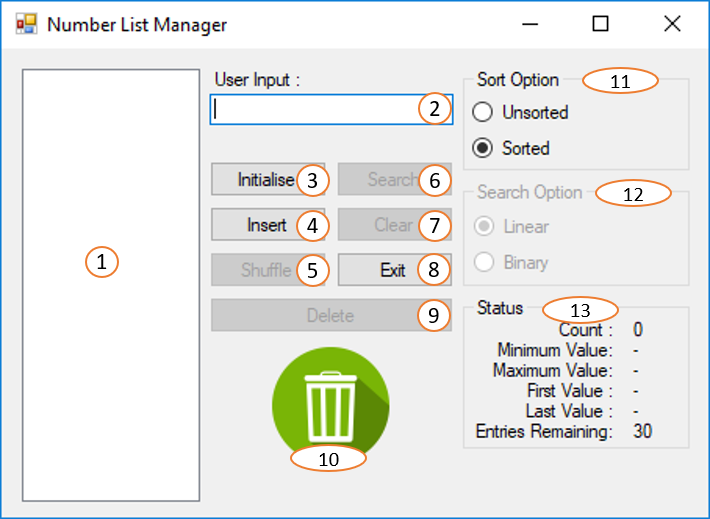
**IT IS YOUR RESPONSIBILITY TO KEEP A RECORD OF ALL WORK SUBMITTED**

|  |
| --- |
| **Part B: Marking and Assessment** |
| This assignment will be marked out of 100  This assignment contributes to 50% of the total module marks.  This assignment is bonded: |
| **Assessment Task:**  See attached details |
| **Learning Outcomes to be assessed** (as specified in the validated module descriptor <http://icis.glam.ac.uk>):  To demonstrate an understanding of the prevailing technologies associated with the development and use of Event-driven Graphical User Interface applications.  To demonstrate the application of the prevailing technologies associated with the design, development and testing of Event-driven Graphical User Interface applications. |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Grading Criteria:**   |  |  |  | | --- | --- | --- | | **Marking Scheme** | **Marks Available** | **Marks Awarded** | | Number List Manager program (Visual Studio and C#) | **60** |  | | Report detailing the design, implementation and testing of the Number List Manager program | **40** |  | |

# Number List Manager Documentation

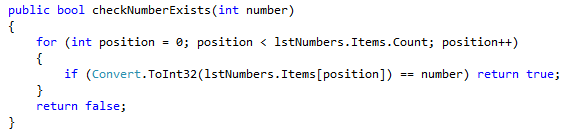
## Design



1. Listbox (lstNumbers) - holds 30 integer values ranging from 0 to 100. Initially enabled.
2. User Input Textbox (txtUserInput) – takes a value inputted by the user to insert into the list or to search the list. Initially enabled.
3. Initialise Button (btnInitialise) – fills the list up with random unique integers ranging from 0 to 100 until the numbers of integers in the listbox is equal to 30. Initially enabled. Disables when the item count is equal to 30, re-enables when the item count is less than 30.
4. Insert Button (btnInsert) – inserts the number entered by the user into the list given that the input is a valid unique integer. Initially enabled. Disables when the item count is equal to 30, re-enables when the item count is less than 30.
5. Shuffle Button (btnShuffle) – shuffles the unsorted list. Initially disabled. Enables when the unsorted radio button is checked and the item count is more than 1. Disables when there are less than 2 items in the unsorted list or the sorted radio button is checked.
6. Search Button (btnSearch) – searches the list for the value inputted by the user. Initially disabled. Enables when the item count is greater than 0. Disables when the list is empty.
7. Clear Button (btnClear) – clears all values from list. Initially disabled. Enables when the item count is greater than 0. Disables when the list is empty.
8. Exit Button (btnExit) – exits the program. Initially enabled.
9. Delete Button (btnDelete) – deletes the selected value in the list. Initially disabled. Enabled when a value is selected. Disabled when there is no item selected.
10. Drag & Drop Delete Bin (picBin) - deletes the selected value from the list dragged into the area. Initially disabled. Enabled when a value is selected. Disabled when there is no item selected.
11. Sort Options (grpSort) – different options for the order of the list. Initially enabled. Contains two radio buttons.
    1. Unsorted radio button (optUnsorted) – display the list as unsorted, enable the shuffle button and disable the binary search option. Initially enabled and checked.
    2. Sorted radio button (optSorted) – display the list as sorted, enable the binary search option and disable the shuffle button. Initially enabled.
12. Search Options (grpSearch) – different options for searching the list. Initially disabled. Enabled when the item count is greater than 0. Disables when the list is empty. Contains two radio buttons.
    1. Linear radio button (optLinear) – changes the behaviour of the search button to a linear search. Initially enabled and checked.
    2. Binary radio button (optBinary) – changes the behaviour of the search button to a binary search. Initially disabled. Enabled when the sorted radio button is checked. Disabled when the unsorted radio button is checked.
13. Status Display (grpStatus) – displays statistics about the list and is refreshed after each initialise, insert, shuffle, delete, clear and sort functions. Initially enabled. Contains 7 labels to hold the values.
    1. Count (lblCountValue) – displays the number of items in the list
    2. Minimum Value (lblMinimumValue) – displays the lowest numerical number in the list.
    3. Maximum Value (lblMaximumValue) – display the highest numerical number in the list.
    4. First Value (lblFirstValue) – displays the number at index 0 in the list.
    5. Last Value (lblLastValue) – displays the number at the last index in the list.
    6. Entries Remaining (lblEntriesRemainingValue) – display the number of values allowed to be entered before the list is full.
    7. Maximum Entries (lblMaximumEntriesValue) – display the maximum number of items allowed in the list.

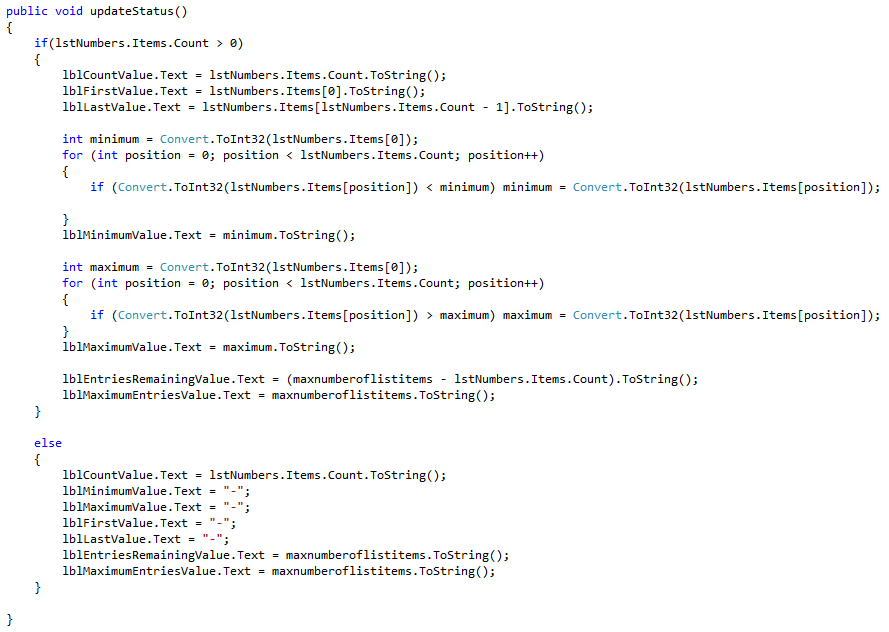
## Implementation

#### checkNumberExists(int number)



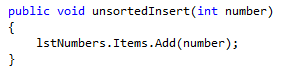
This function returns a Boolean value that represents if the number already exists within the list. The function uses a for loop to search through the list from the beginning. If the number is found in the list, the function will return true. If the number is not found in the list, the for loop will end and the function will return false.

#### updateStatus()



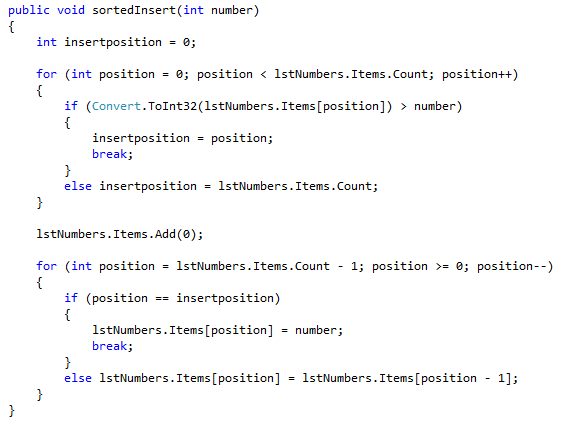
This function updates the values of the status labels every time the list changes. If there are values stored in the list, the countvalue label is set the count of items in the list also the firstvalue and lastvalue labels are set to the value in the first and last position. There are also 2 for loop, one to find the minimum value and one to find the maximum value. For the minimum value, if the number in the current position is less than the current value of minimum then that value becomes the new minimum. The minimumvalue label is then set to the value of the minimum number. For the maximum value, if the number in the current position is more than the current value of maximum then that value becomes the new maximum. The maximumvalue label is then set to the value of the maximum number. The entriesremainingvalue label is set to the number of items that can be inserted into the list. The maximumentriesvalue label is set to the maximum number of values that the list can hold. If the list contains no values, then the labels are set to their default values.

#### unsortedInsert(int number)



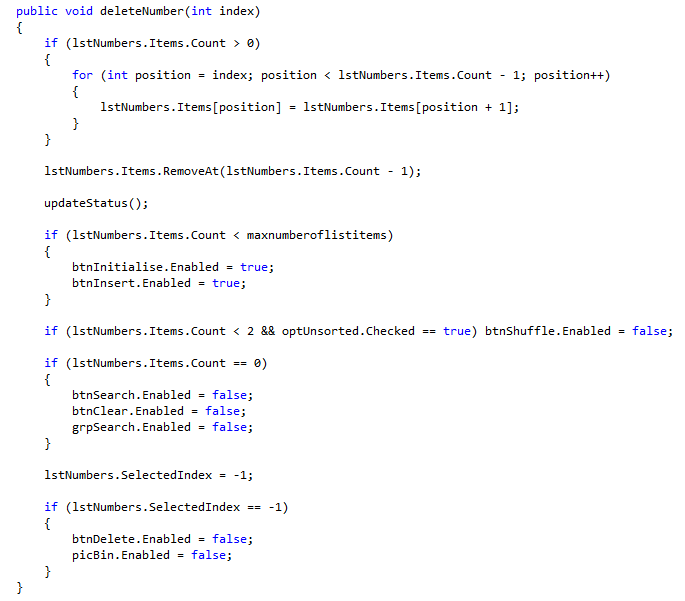
This function is used to insert a number into an unsorted list. So, if the unsorted radio button is checked, this function will take the number passed in as a parameter and append the number to the end of the list.

#### sortedInsert(int number)



This function is used to insert a number into a sorted list. So, if the sorted radio button is checked, this function will use a for loop to search the list to find the first value which is larger than the number passed in as a parameter or select the end of the list as the insert position because this is where number needs to be inserted for the list to still be in order. After the insert position is found, a blank item is added to the end of the list to allow items to be copied down one position to create space for the new number. Then another for loop from the end of the list, works its way up the list looking for the insert position. If the current position is the insert position, then the number is inserted. Else the value of the current position is set equal to the value of the item above it in the list, creating an opening for the new number.

#### deleteNumber(int index)

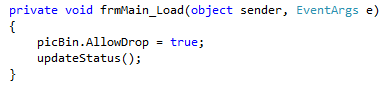


This function is used to delete a number in the list at the position passed in by the index parameter. A for loop is used, starting at the position of the number to be deleted, the loop then runs until the end of the list, copying each value up by one position. This overwrites the number to be deleted and creates a duplicate value at the end of the list which is removed using the RemoveAt() function. The function then calls the updateStatus() function to update the values. The function then goes on to enable and disable different elements of the form depending on the state of the list.

* If the count is below the maximum number of items, initialise and insert buttons are enabled.
* If the count is below 2 and the unsorted radio button is checked, the shuffle button is disabled.
* If the list is empty, the search and clear buttons also the search options group are disabled.

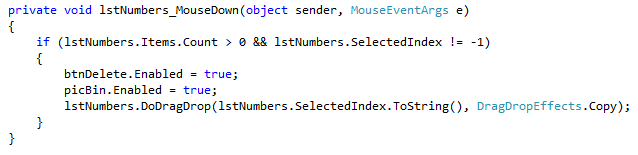
The selected index is then set to -1 so that no item is selected in the list and the delete methods can be disabled.

#### frmMain\_Load(object sender, EventArgs e)



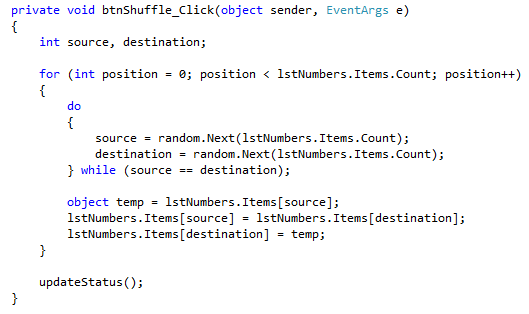
This function runs when the program is initially loaded. The function allows items to be dragged from the list and dropped on the bin image. Also runs the updateStatus() function to set the labels to their default values.

#### lstNumbers\_MouseDown(object sender, MouseEventArgs e)



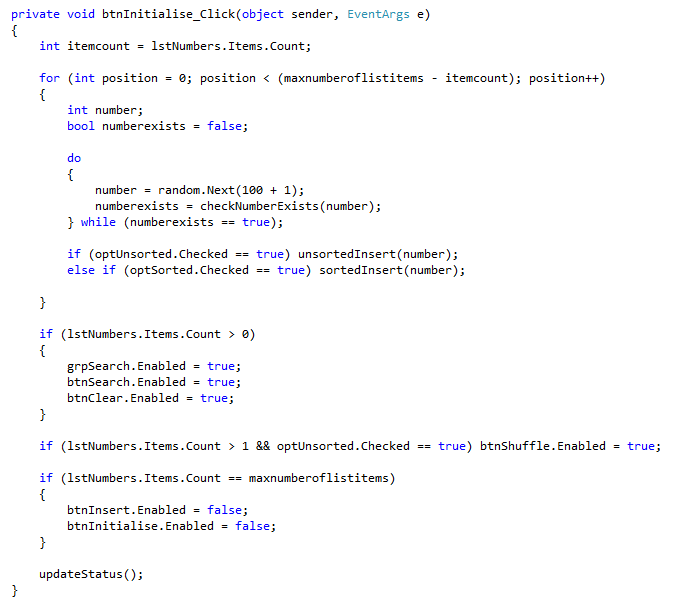
This function runs when an item in the list is clicked on or when the item is being pick up to drag. If the list us not empty and there is an item selected then the 2 delete methods are enabled and the drag-drop event is initiated.

#### btnShuffle\_Click(object sender, EventArgs e)



This function is called when the shuffle button is clicked and the list is unsorted. The function selects 2 random positions from the list and swaps the values around. This happens multiple times so the list is then shuffled. The updateStatus() function is also called to update the labels.

#### btnInitialise\_Click(object sender, EventArgs e)

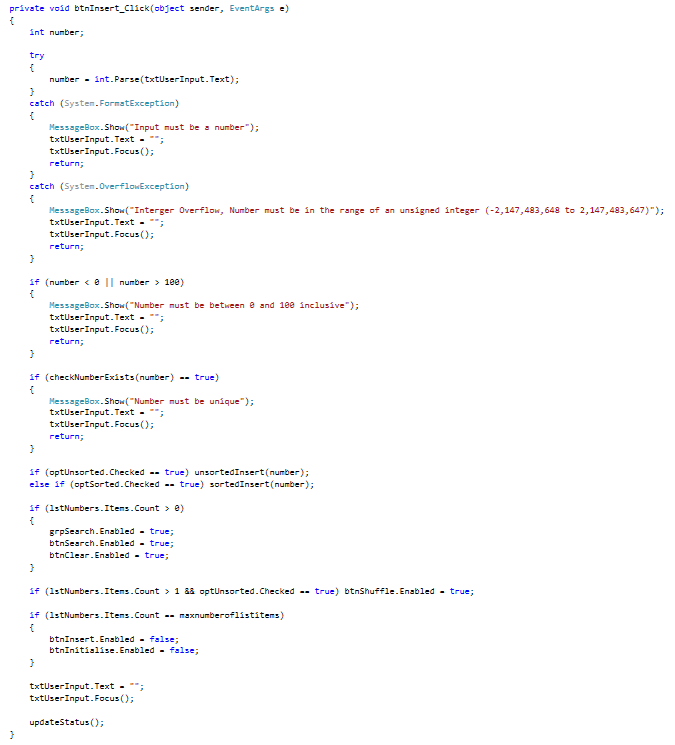


This function runs when the initialise button is clicked and is used to fill the list up with random values until the list is full. A for loop is used until the list is full to fill up the list with a random unique number generated using the random.Next() function. If the list is unsorted, the number is inserted using the unsortedInsert() function. If the list is sorted, the number is inserted using the sortedInsert() function. The function then goes on to enable and disable different elements of the form depending on the state of the list.

* If there are items in the list, the search group, search and clear buttons are enabled.
* If there are more than 1 items in the list and the list is unsorted, the shuffle button is enabled.
* If the list is full, the insert an initialise buttons are disabled.

The updateStatus() function is also called to update the labels.

#### btnInsert\_Click(object sender, EventArgs e)

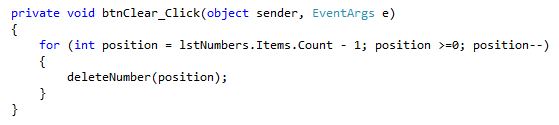


This function is called when the insert button is clicked. The function takes the value from the userinput textbox. The function then tries to parse the string into an integer. If the string is not a number or is too large that it causes an overflow, these exceptions are caught and the relevant error message is displayed. After being parsed into an integer, the user input is then checked to see if it is a valid number between 0 and 100 also to see if it is a unique number. If the number is invalid or not unique, then error messages are also displayed. After all these checks, the number is now verified as a valid number and is inserted using either the sortedInsert() or unsortedInsert() depending on which sort option is checked. The function then goes on to enable and disable different elements of the form depending on the state of the list.

* If there are items in the list, the search group, search and clear buttons are enabled.
* If there are more than 1 items in the list and the list is unsorted, the shuffle button is enabled.
* If the list is full, the insert an initialise buttons are disabled.

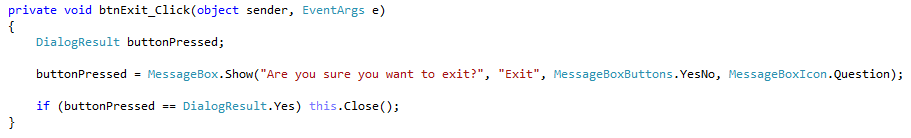
The updateStatus() function is also called to update the labels.

#### btnClear\_Click(object sender, EventArgs e)



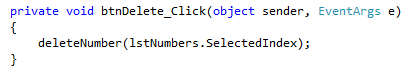
This function is called when the clear button is clicked and is used to delete all the elements in the list. The for loop starts at the end of the list and runs while the list is not empty. The loop moves up the list, calling the deleteNumber() function for each index therefore making the list empty.

#### btnExit\_Click(object sender, EventArgs e)



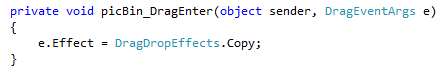
This function is called when the exit button is clicked. The function will display a message box asking the user to confirm if they want to exit the program. If the dialog result is “yes”, the form will close and the program will end.

#### btnDelete\_Click(object sender, EventArgs e)



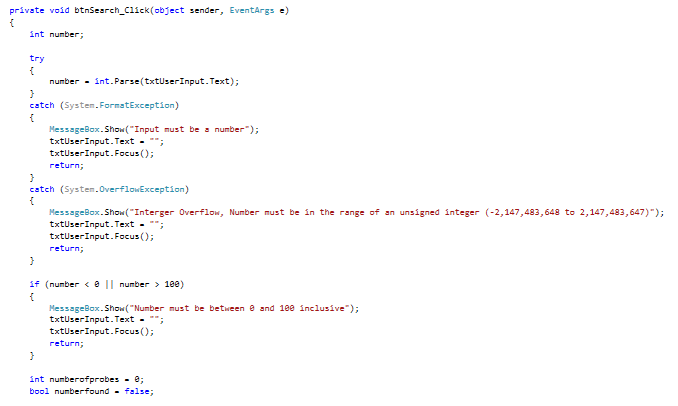
This function is called when the delete button is clicked. The function takes the selected index and passes it into the deleteNumber() function to remove the value from the list.

#### picBin\_DragEnter(object sender, DragEventArgs e)

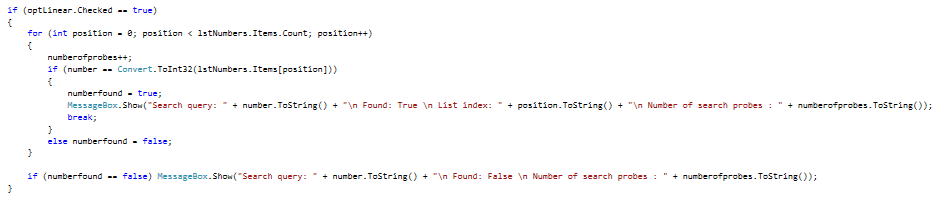


This function is called when an item is dragged over the bin image. This function is the handler for copying the item to drop onto the bin image.

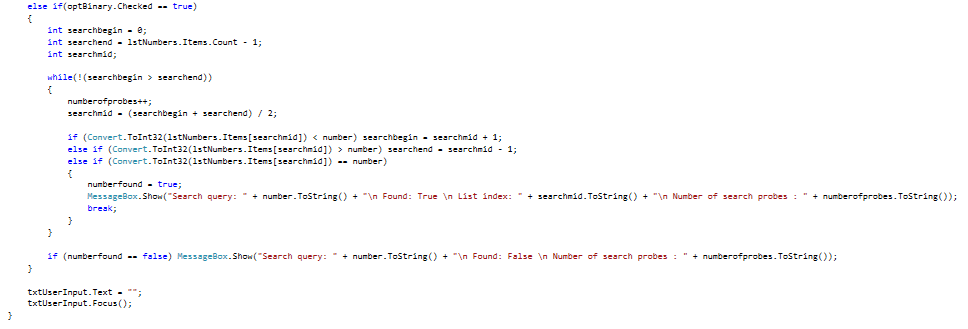
#### btnSearch\_Click(object sender, EventArgs e)



This function runs when the search button is clicked. Firstly, the function tries to parse the string into an integer. If the string is not a number or is too large that it causes an overflow, these exceptions are caught and the relevant error message is displayed. After been parsed into an integer, the user input is then checked to see if it is a valid number between 0 and 100. If the number is invalid, then an error message is displayed. The behaviour of the function depends on which search option is checked.

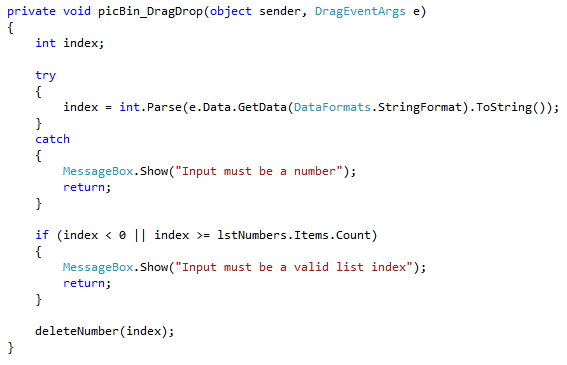


If the linear search option is checked then the code above is executed. This code goes through the list from the begin, checking each value against the search term. If the number is found, a message box is displayed and the loop breaks. Else if the number is not found, the loop ends and a different message box is displayed.



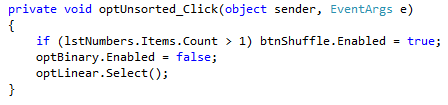
If the binary search option is checked then the code above is executed. This code runs while the searchbegin and searchend do not cross each other. The searchmid is set to the middle of the list. If the number at the middle of the list is less than the search term, the searchbegin is set to the searchmid + 1. Else if the middle number is greater than the search term, the searchend is set to the searchmid – 1. Else if the middle number is equal to the search term then the number has been found, a message box is displays and the loop breaks. If the number is not found, the loop ends and a different message box is displayed.

#### picBin\_DragDrop(object sender, DragEventArgs e)



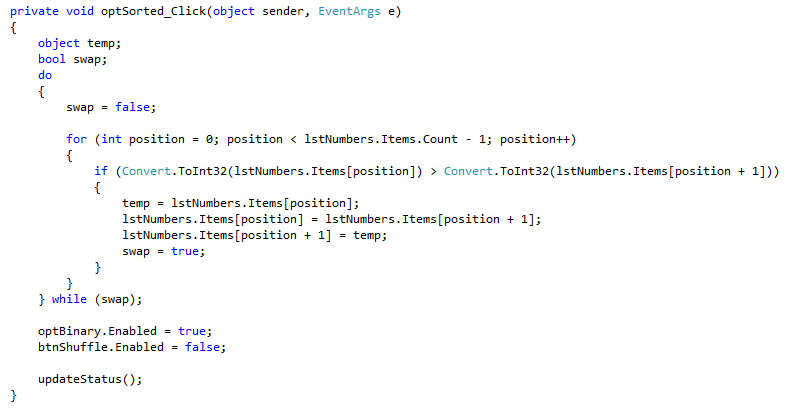
This function is called when an item is dropped over the bin image. The function tries to parse the data from the event as an integer. If it fails to parse the data, an error message will be displayed. The number is then checked if it is a valid index number and displays an error message if it isn’t. The function then calls the deleteNumber() function passing the index variable in as a parameter to delete that element from the list.

#### optUnsorted\_Click(object sender, EventArgs e)



This function is called when the unsorted radio button is checked. The function enabled the shuffle button if the list contains more than one item. The function also disables the binary search option and selects the linear search option.

#### optSorted\_Click(object sender, EventArgs e)



This function is called when the sorted radio button is checked. The function firstly sorts the list using a bubble sort algorithm. This algorithm looks at the next term and swaps it with the current term if the current term is bigger than the next term. This runs until a swap no longer occurs because the list has been sorted. This function also enables the binary search option and disables the shuffle button. The updateStatus() function is also called to update the labels.

## Testing

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Number | Object | Test | Pre-test conditions | Expected post-test conditions | Outcome |
| 1 | btnInitialise | Does the list fill up with 30 random unique integers ranging between 0 to 100 when list is empty? | List is empty.  Initialise and insert buttons are enabled.  Shuffle, search and clear buttons are disabled.  Search options group is disabled.  (Figure 1a) | List is filled with 30 unique integers ranging from 0 to 100.  Initialise and insert buttons are disabled.  Shuffle, search and clear buttons are enabled.  Search options group is enabled. | As Expected  (Figure 1b) |
| 2 | Does a partially fill list fill back up to 30 unique valid values? | List contains 10 values.  Initialise and insert buttons are enabled.  (Figure 2a) | List is filled with an additional 20 values. Initialise and insert buttons are disabled. | As Expected  (Figure 2b) |
| 3 | When the sorted radio is checked, does the list initialise sorted? | List is empty.  Sorted radio button checked.  Initialise and insert buttons are enabled.  Search and clear buttons are disabled.  Search options group is disabled.  Binary search radio button is disabled.  (Figure 3a) | List is filled with 30 unique integers in ascending order.  Initialise and insert buttons are disabled.  Search and clear buttons are enabled.  Search options group is enabled.  Binary search radio button is enabled. | As Expected  (Figure 3b) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4 | btnInsert | When unsorted, is the number inserted in the list? | List is empty.  Unsorted is checked.  Search and clear buttons are disabled.  Search options group is disabled.  (Figure 4a) | Number is inserted into list. Search and clear buttons are enabled.  Search options group is enabled. | As Expected  (Figure 4b) |
| 5 | When unsorted, does inserting a second number enable the shuffle button? | List has one value.  Unsorted is checked.  Shuffle button is disabled.  (Figure 5a) | Shuffle button is enabled when second value is added | As Expected  (Figure 5b) |
| 6 | When sorted, is the number inserted at the begin of the list? | List of 10 elements.  Sorted is checked.  (Figure 6a) | Number is inserted at the begin of the list | As Expected  (Figure 6b) |
| 7 | When sorted, is the number inserted into the middle of the list? | List of 11 elements.  Sorted is checked.  (Figure 7a) | Number is inserted into the middle of the list | As Expected  (Figure 7b) |
| 8 | When sorted, is the number inserted at the end of the list? | List of 12 elements.  Sorted is checked.  (Figure 8a) | Number is inserted at the end of the list | As Expected  (Figure 8b) |
| 9 | When the list is full, does the insert and initialise buttons disable? | List of 29 elements.  Initialise and insert button enabled.  (Figure 9a) | Initialise and insert buttons are disabled | As Expected  (Figure 9b) |
| 10 | Does a message box display when the input is a string? | List of 10 elements.  String in user input field.  (Figure 10a) | Message box appears showing that the input must be a number | As Expected  (Figure 10b) |
| 11 | Does a message box display when the input causes an integer overflow? | List of 10 elements.  Number that would cause overflow in user input field.  (Figure 11a) | Message box appears showing that an overflow has occurred | As Expected  (Figure 11b) |
| 12 | btnInsert | Does a message box display when the input is out of the valid range 0 – 100? | List of 10 elements.  Number that is out of range in the user input  (Figure 12a) | Message box appears showing that the number must be between 0 and 100 | As Expected  (Figure 12b) |
| 13 | Does a message box display when the input is already in the list? | List of 10 elements.  Number that is already in list in the user input  (Figure 13a) | Message box appears showing that the number must be unique | As Expected  (Figure 13b) |
| 14 | btnShuffle | Does pressing the shuffle button reorder an unsorted list? | Full list.  Unsorted checked, shuffle button enabled.  (Figure 14a) | List order is shuffled | As Expected  (Figure 14b) |
| 15 | btnExit | Does pressing the exit button quit the program? | None  (Figure 15a) | A message box appears requesting confirmation then program exits if confirmed. | As Expected  (Figure 15b) |
| 16 | btnClear | Does pressing the clear button reset the list to empty? | Full list.  Initialise and insert buttons are disabled.  Shuffle, search and clear buttons are enabled.  Search options group is enabled.  (Figure 16a) | List becomes empty.  Initialise and insert buttons are enabled.  Shuffle, search and clear buttons are disabled.  Search options group is disabled. | As Expected  (Figure 16b) |
| 17 | btnSearch | Does the linear search work looking for a number that exists? | Full list.  Linear radio button is checked.  (Figure 17a) | Message box appears with the results for the number regardless of its position in the list. | As Expected.  Number found at begin of list.  (Figure 17b)  Number found at end of list.  (Figure 17c)  Number found in the middle of list.  (Figure 17d) |
| 18 | btnSearch | Does the binary search work looking for a number that exists? | Full list.  Sorted radio button is checked.  Binary radio button is checked.  (Figure 18a) | Message box appears with the results for the number regardless of its position in the list. | As Expected.  Number found at begin of list.  (Figure 18b)  Number found at end of list.  (Figure 18c)  Number found in the middle of list.  (Figure 18d) |
| 19 | Does the linear search work looking for a number that doesn’t exist? | Full list.  Linear radio button is checked.  (Figure 19a) | Message box appears with the results showing that the number cannot be found. | As Expected.  (Figure 19b) |
| 20 | Does the binary search work looking for a number that doesn’t exist? | Full list.  Sorted radio button is checked.  Binary radio button is checked.  (Figure 20a) | Message box appears with the results showing that the number cannot be found regardless of list position. | As Expected.  Search begin of list.  (Figure 20b)  Search end of list  (Figure 20c)  Search middle of list.  (Figure 20d) |
| 21 | Does a message box display when the input is a string? | List of 10 elements.  String in user input field.  (Figure 21a) | Message box appears showing that the input must be a number | As Expected  (Figure 21b) |
| 22 | Does a message box display when the input causes an integer overflow? | List of 10 elements.  Number that would cause overflow in user input field.  (Figure 22a) | Message box appears showing that an overflow has occurred | As Expected  (Figure 22b) |
| 23 | Does a message box display when the input is out of the valid range 0 – 100? | List of 10 elements.  Number that is out of range in the user input  (Figure 23a) | Message box appears showing that the number must be between 0 and 100 | As Expected  (Figure 23b) |
| 24 | btnDelete | Does the selected value get deleted when the delete button is pressed? | List of 30 elements.  Item selected which enables delete button.  Initialise and insert buttons are disabled.  (Figure 24a) | Value gets deleted.  No item selected therefore the delete button should be disabled.  Initialise and insert buttons are enabled. | As Expected  (Figure 24b) |
| 25 | When there are less than 2 values, Does the shuffle button disable? | List has two value.  Unsorted is checked.  Shuffle button is enabled.  (Figure 25a) | Shuffle button is disabled when second value is deleted | As Expected  (Figure 25b) |
| 26 | When the last item is deleted, does the form reset? | List has one value.  Search and clear buttons are enabled.  Search options group is enabled.  (Figure 26a) | List is empty.  Search and clear buttons are disabled.  Search options group is disabled. | As Expected  (Figure 26b) |
| 27 | picBin | Does the selected value get deleted when dropped over the bin area? | List of 30 elements.  Item selected which enables drag and drop area.  (Figure 27a) | Value gets deleted.  No item selected therefore the area should be disabled. | As Expected  (Figure 27b) |
| 28 | Does the program catch data dragged from other locations? | List of 30 elements.  Item selected which enables drag and drop area.  Item dragged from desktop.  (Figure 28a) | Message box appears saying the dropped data must be a valid list index. | As Expected  (Figure 28b) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 29 | optUnsorted | Does the unsorted radio button successfully switch states? | List of 30 elements. Sorted radio button checked  Shuffle button disabled.  Binary search radio button enabled.  (Figure 29a) | Unsorted radio button checked.  Shuffle button enabled.  Binary search radio button disabled. | As Expected  (Figure 29b) |
| 30 | optSorted | Does the sorted radio button successfully switch states? | List of 30 elements. Unsorted radio button checked  Shuffle button enabled.  Binary search radio button disabled.  (Figure 30a) | Sorted radio button checked.  Shuffle button disabled.  Binary search radio button enabled.  List in ascending order. | As Expected  (Figure 30b) |

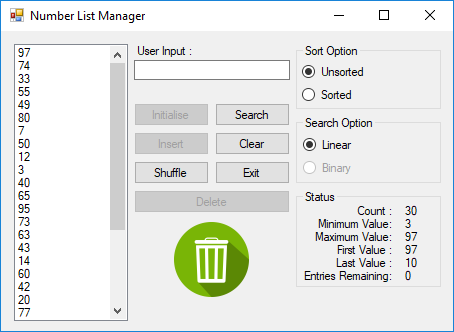
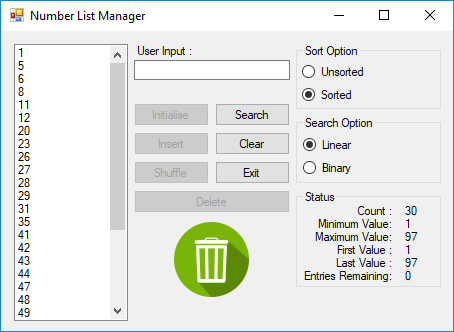


Figure 3b - Test 3 Outcome

Figure 3a - Test 3 Pre-conditions

Figure 2b - Test 2 Outcome

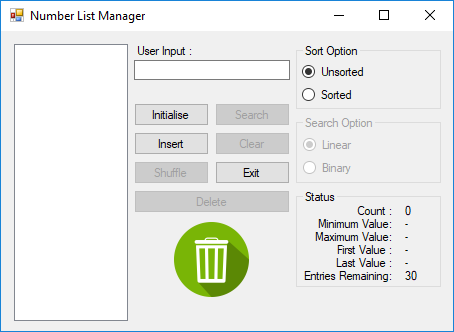
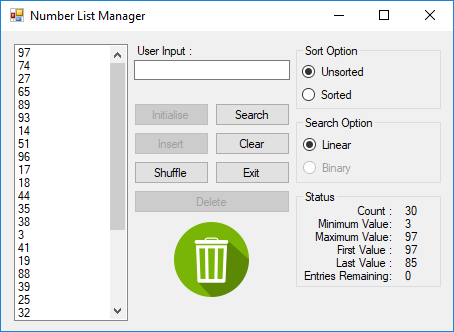
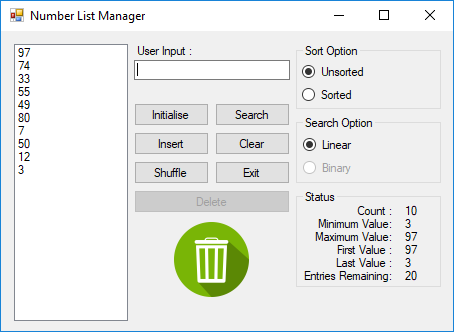
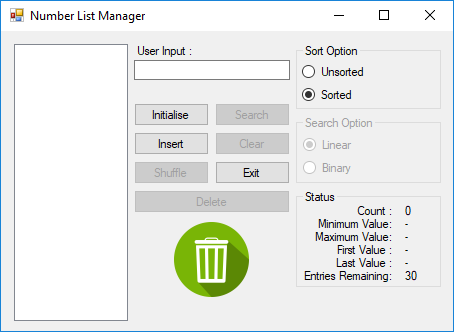


Figure 2a - Test 2 Pre-conditions

Figure 1b - Test 1 Outcome

Figure 1a - Test 1 Pre-conditions

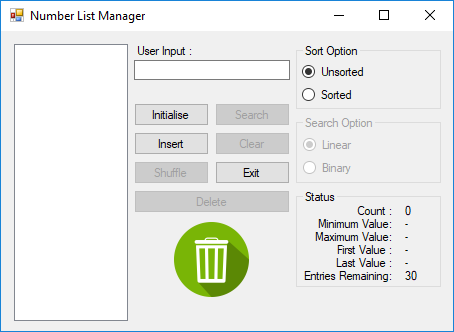
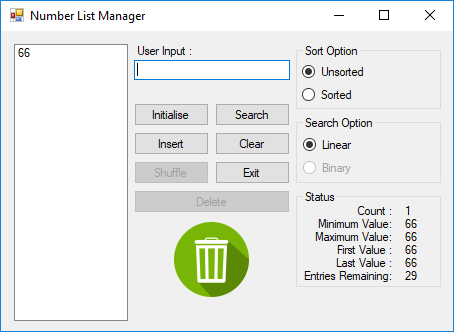
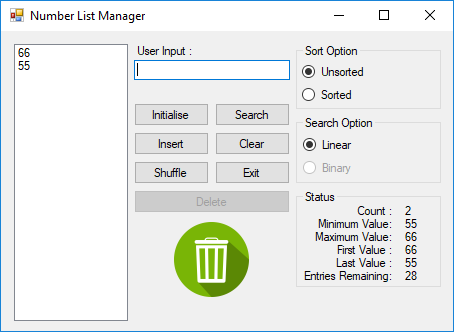
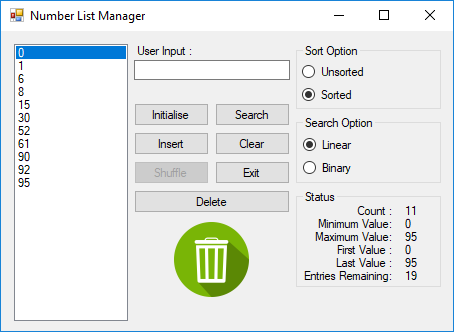


Figure 6b - Test 6 Outcome

Figure 5b - Test 5 Outcome

Figure 4b - Test 4 Outcome

Figure 4a - Test 4 Pre-conditions

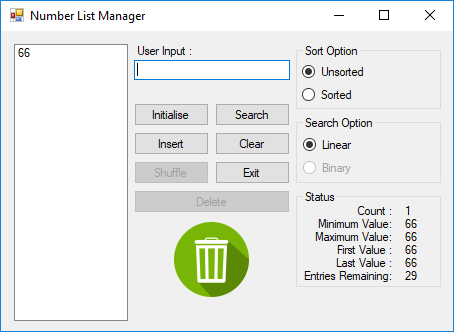
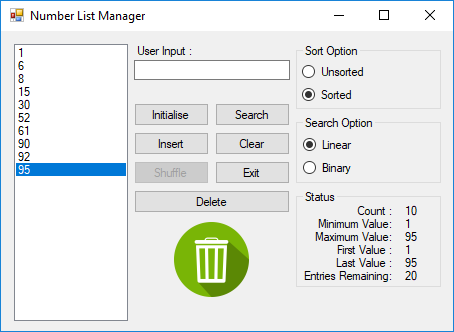


Figure 6a - Test 6 Pre-conditions

Figure 5a - Test 5 Pre-conditions

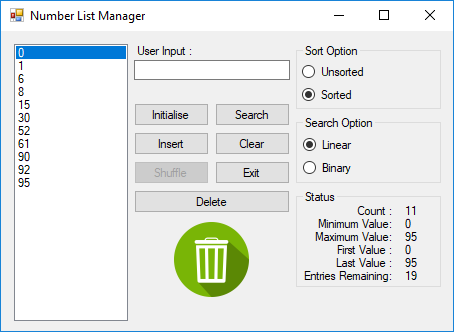
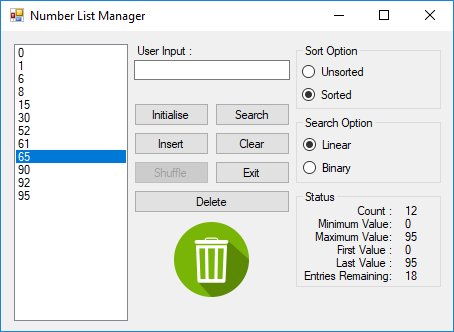
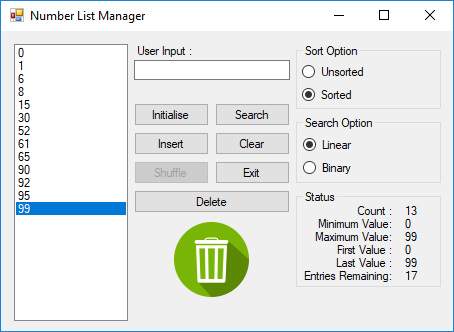
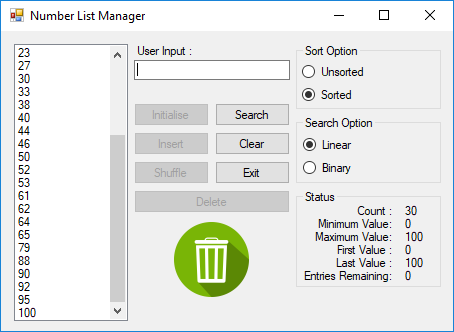


Figure 9b - Test 9 Outcome

Figure 8b - Test 8 Outcome

Figure 7b - Test 7 Outcome

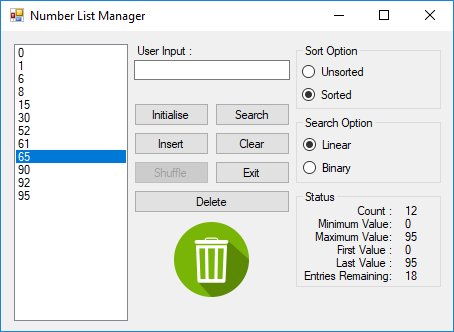
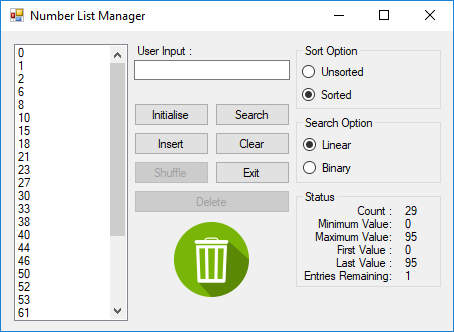


Figure 9a - Test 9 Pre-conditions

Figure 8a - Test 8 Pre-conditions

Figure 7a - Test 7 Pre-conditions

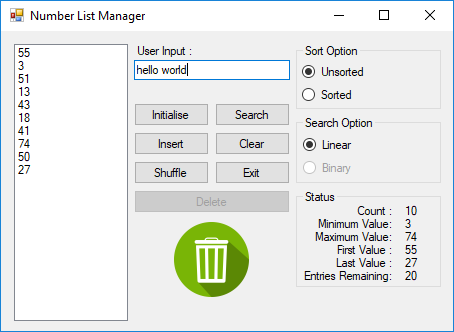


Figure 10a - Test 10 Pre-conditions

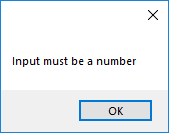
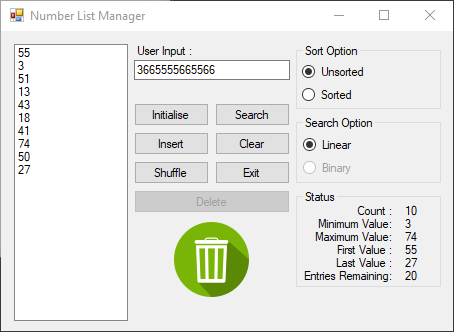


Figure 10b - Test 10 Outcome



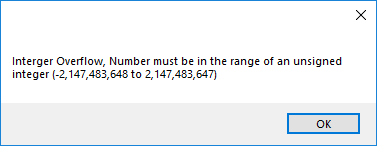
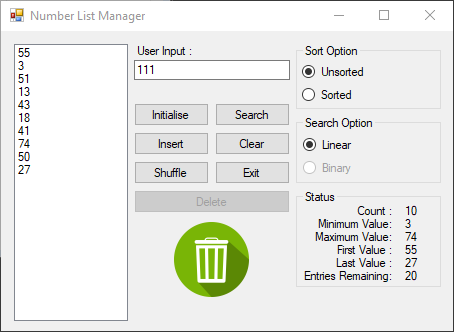


Figure 11b - Test 11 Outcome

Figure 11a - Test 11 Pre-conditions



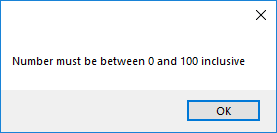
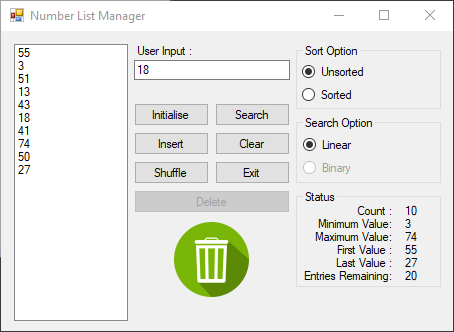


Figure 12b - Test 12 Outcome

Figure 12a - Test 12 Pre-conditions



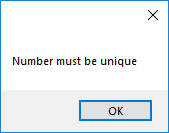


Figure 13a - Test 13 Pre-conditions

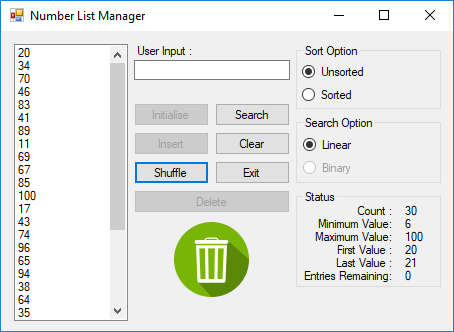
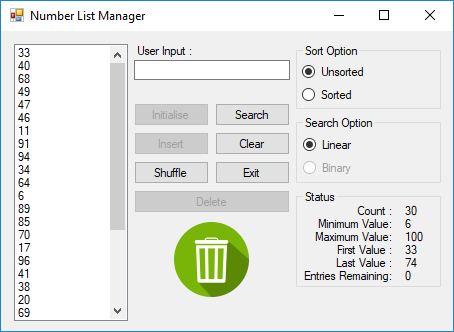
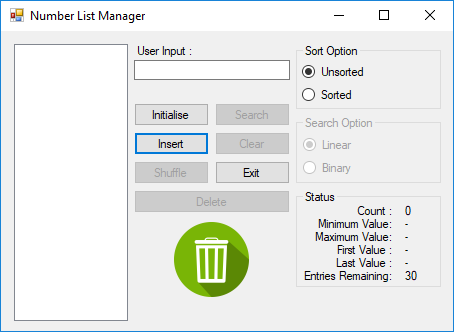


Figure 14b - Test 14 Outcome

Figure 13b - Test 13 Outcome

Figure 14a - Test 14 Pre-conditions



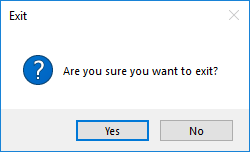


Figure 15b - Test 15 Outcome

Figure 15a - Test 15 Pre-conditions

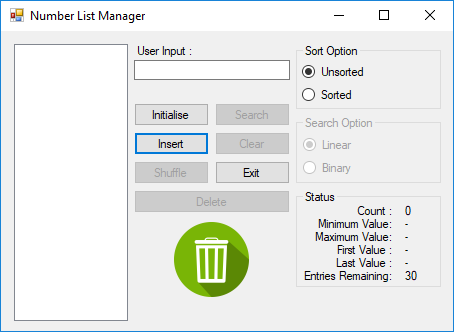


Figure 16b – Test 16 Outcome

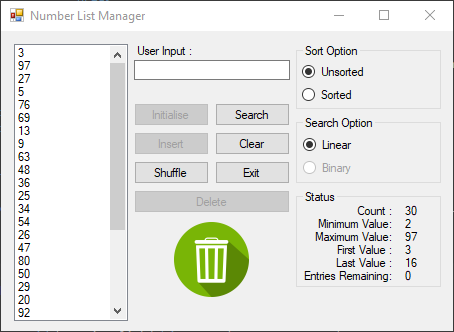
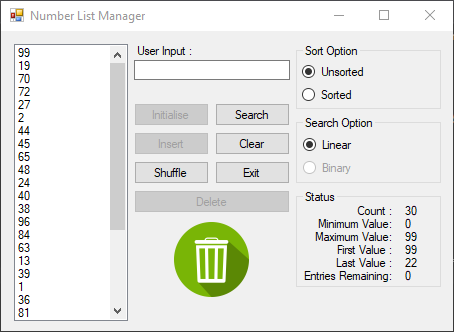
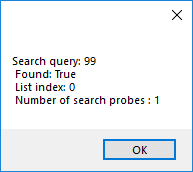
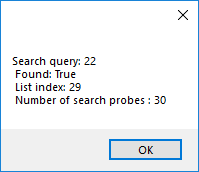


Figure 16a - Test 16 Pre-conditions

Figure 17b – Test 17 Outcome

Figure 17a - Test 17 Pre-conditions



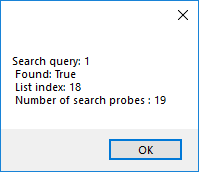


Figure 17d – Test 17 Outcome

Figure 17c – Test 17 Outcome

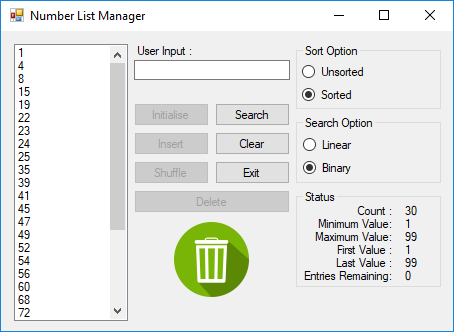
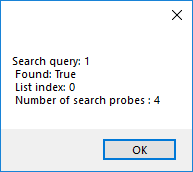


Figure 18b – Test 18 Outcome

Figure 18a - Test 18 Pre-conditions

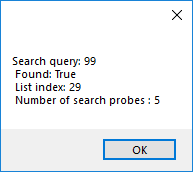
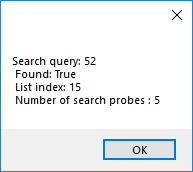
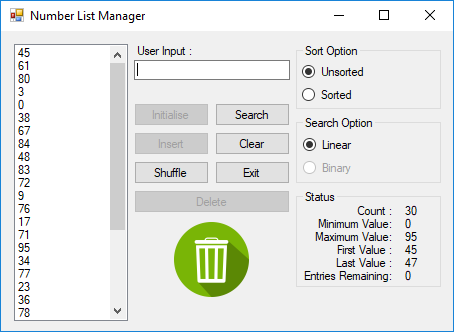


Figure 18c – Test 18 Outcome

Figure 18d – Test 18 Outcome



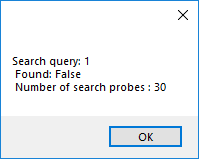


Figure 19b – Test 19 Outcome

Figure 19a - Test 19 Pre-conditions

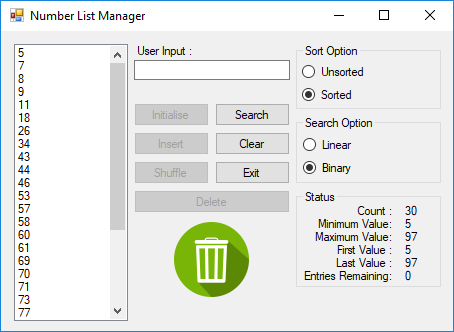
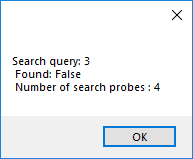


Figure 20a - Test 20 Pre-conditions

Figure 20b – Test 20 Outcome

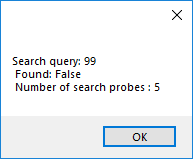
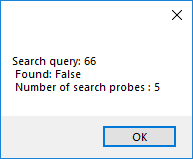
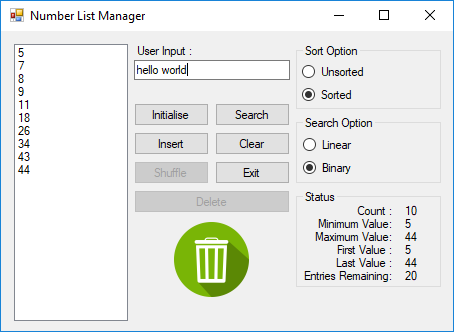


Figure 20c – Test 20 Outcome

Figure 20d – Test 20 Outcome



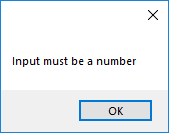
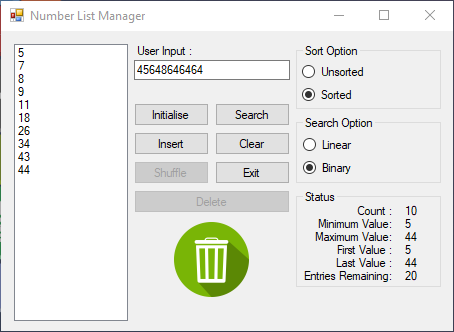


Figure 21b – Test 21 Outcome

Figure 21a - Test 21 Pre-conditions



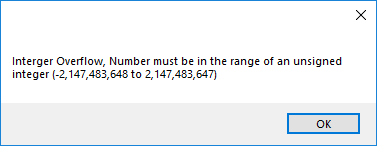
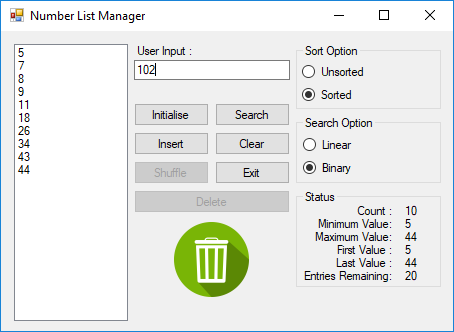
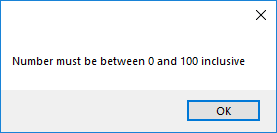


Figure 22b – Test 22 Outcome

Figure 22a - Test 22 Pre-conditions



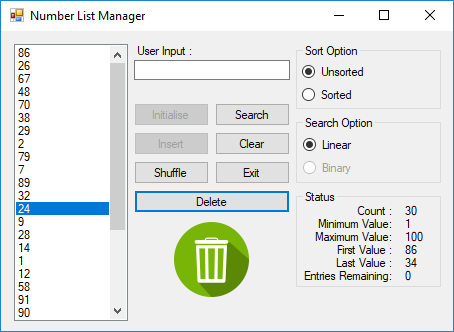
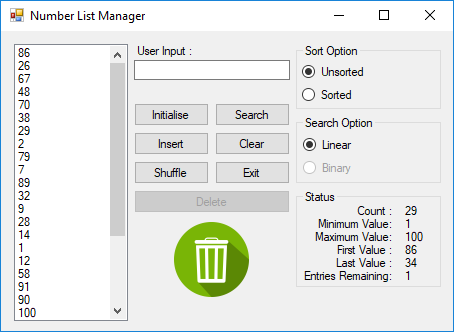


Figure 24b – Test 24 Outcome

Figure 23b – Test 23 Outcome

Figure 24a - Test 24 Pre-conditions

Figure 23a - Test 23 Pre-conditions

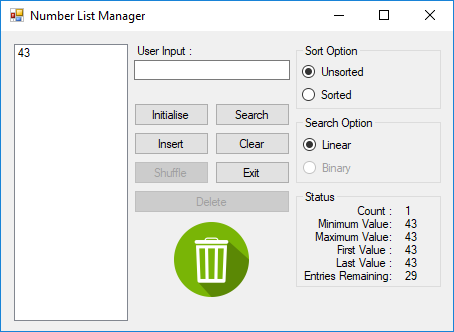
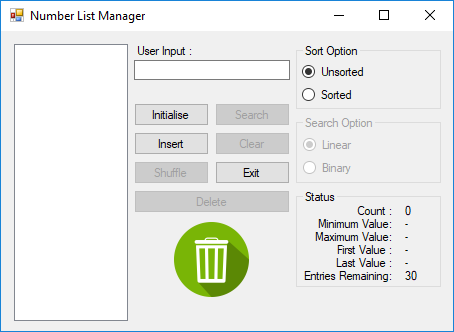
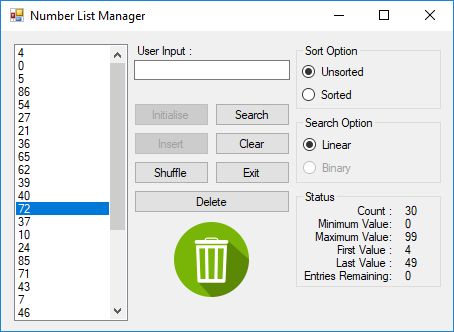
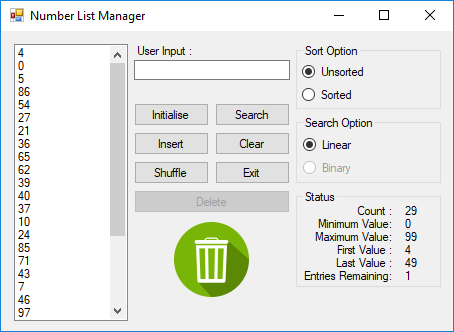


Figure 27b – Test 27 Outcome

Figure 26b – Test 26 Outcome

Figure 25b – Test 25 Outcome

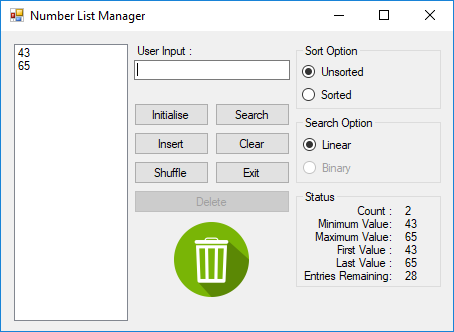
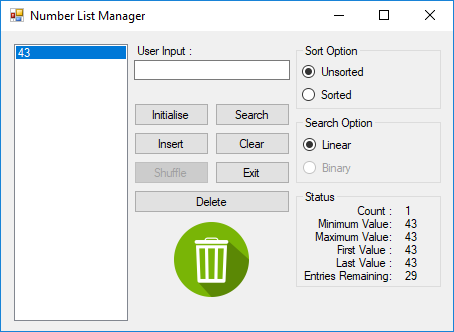


Figure 27a - Test 27 Pre-conditions

Figure 26a - Test 26 Pre-conditions

Figure 25a - Test 25 Pre-conditions

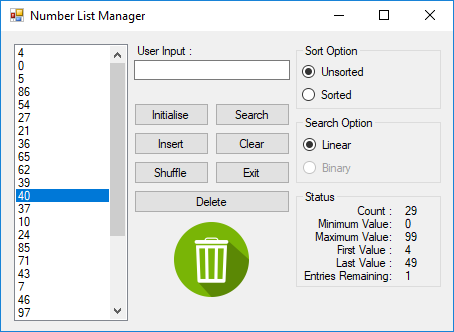
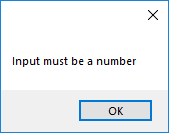


Figure 28a - Test 28 Pre-conditions

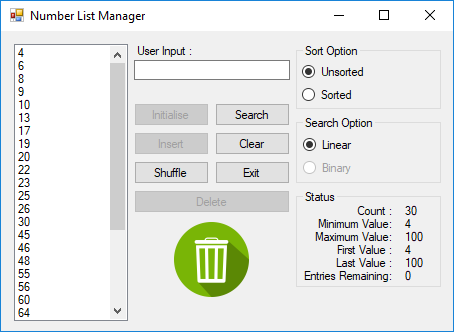
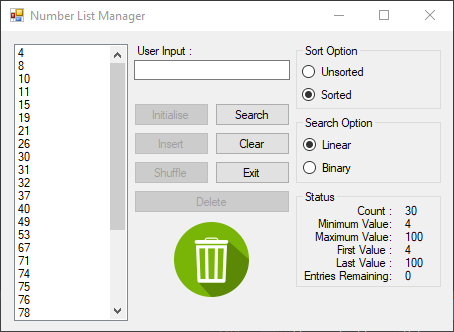


Figure 30b – Test 30 Outcome

Figure 29b – Test 29 Outcome

Figure 28b – Test 28 Outcome

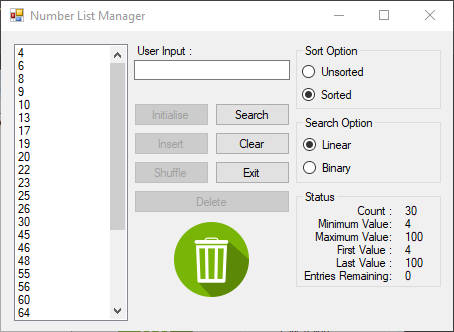
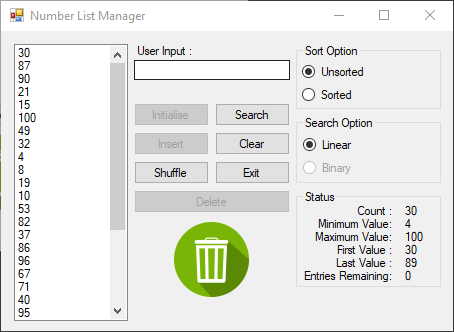


Figure 30a - Test 30 Pre-conditions

Figure 29a - Test 29 Pre-conditions

## Source Code

[See PDF document submitted in ZIP file]