

Design and Implementation of an Aircraft Register for Ministry of Defence

SOFT20091 – Software Design and Implementation 2

Molly Redmond, Thomas Knowles

N0383124, N0362921

**Contents**

Use cases……………………………3

Sequence Diagram……………..8

Class diagrams…………………….9

Internal Data Structure……….12

File format………………………….12

User interface model………….13

Testing……………………………….15

**Use Cases**

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case Name:** StartSystem | | **Use Case ID:** 1 | |
| **Short Description:** Start up the system and load the menu | | | |
| **Trigger:** Opening of the program | | | |
| **Type:** Internal | | | |
| **Major Inputs:**  **Description:**  - Program opening | **Source:**  **-** User | **Major Outputs:**  **Description:**  - Menu list | **Destination:**  - User screen |
| **Major Steps Performed:**  - Print menu | | **Information Required:**  - Menu contents | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case Name:** Add Aircraft | | **Use Case ID:** 2 | |
| **Short Description:** Add a new aircraft to the binary tree | | | |
| **Trigger:** User enters ‘1’ at the menu | | | |
| **Type:** External | | | |
| **Major Inputs:**  **Description:**  - Information for each variable within the aircraft | **Source:**  - User | **Major Outputs:**  **Description:**  - New aircraft data  - New node in binary tree | **Destination:**  - Binary tree |
| **Major Steps Performed:**  **-** Ask user set of questions about the new aircraft to gather the data. - New aircraft is created by calling the relevant aircraft constructor and feeding in the data. - Calls binary tree add function which fires the inserting function which finds the correct place for the data in the tree. | | **Information Required:**  - Type of aircraft to identify the right information needed  - All information about specific aircraft from user  - A binary tree | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case Name:** Delete aircraft | | **Use Case ID:** 3 | |
| **Short Description:** Remove an aircraft from the binary tree | | | |
| **Trigger:** User inputs ‘2’ into menu | | | |
| **Type:** | | | |
| **Major Inputs:**  **Description:**  - Call sig of aircraft to remove | **Source:**  - User | **Major Outputs:**  **Description:**  - New binary tree with requested aircraft removed | **Destination:**  - Binary tree |
| **Major Steps Performed:** - Ask user for call sig of the aircraft to delete - Locate the position of the aircraft in the binary tree  - Establish there any child nodes to handle  - Delete the data from the node - Remove the node from the tree and move any child nodes that were found accordingly | | **Information Required:**  - Call sig of the aircraft to be removed - Binary tree | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case Name:** Search (by Name) | | **Use Case ID:** 4 | |
| **Short Description:** Search for and print an aircraft | | | |
| **Trigger:** User inputs ‘3’ into menu | | | |
| **Type:** | | | |
| **Major Inputs:**  **Description:**  - Name of aircraft | **Source:**  - User | **Major Outputs:**  **Description:**  - Information about aircraft printed | **Destination:**  - Console screen |
| **Major Steps Performed:**  - Search through the binary tree for the name matching the requested aircraft  - Print out the information about that aircraft | | **Information Required:**  - Name of aircraft  - Binary tree | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case Name: S**earch (by Call Sig) | | **Use Case ID:** 5 | |
| **Short Description:** Search for an aircraft by Call Sig and print | | | |
| **Trigger:** User inputs ‘4’ into menu | | | |
| **Type:** | | | |
| **Major Inputs:**  **Description:**  - Call sig of aircraft to search | **Source:**  - User | **Major Outputs:**  **Description:**  - Information of the requested aircraft printed | **Destination:**  - Console |
| **Major Steps Performed:**  - Search through the binary tree for the Call Sig matching the requested aircraft  - Print out the information about the aircraft | | **Information Required:**  - Call sig of aircraft  - Binary tree | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case Name:** Print aircrafts | | **Use Case ID:** 6 | |
| **Short Description:** Print all aircrafts in binary tree | | | |
| **Trigger:** User inputs ‘5’ into menu | | | |
| **Type:** | | | |
| **Major Inputs:**  **Description:**  - Menu option | **Source:**  - User | **Major Outputs:**  **Description:**  - List of aircrafts printed | **Destination:**  - Console |
| **Major Steps Performed:**  - Go into Binary Tree  - Add data to vector  - Print each aircraft with all information | | **Information Required:**  - Information in Binary tree | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case Name:** Save to file | | **Use Case ID:** 7 | |
| **Short Description:** Save everything in the binary tree to the datastore | | | |
| **Trigger: User enters ‘7’ into the menu** | | | |
| **Type:** | | | |
| **Major Inputs:**  **Description:**  - Menu option | **Source:**  - User | **Major Outputs:**  **Description:**  - Aircrafts saved to file | **Destination:**  - Txt file |
| **Major Steps Performed:**  - Load the contents of the binary tree into the file | | **Information Required:**  - Binary tree | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case Name:** Load from file | | **Use Case ID:** 8 | |
| **Short Description:** Load aircraft file into Binary Tree | | | |
| **Trigger: User enters ‘8’ into menu** | | | |
| **Type:** | | | |
| **Major Inputs:**  **Description:**  - Menu option  - Aircraft file to load | **Source:**  - User  - User | **Major Outputs:**  **Description:**  - Filled binary tree | **Destination:**  - System |
| **Major Steps Performed:**  - Retrieve the information from the database and load into Binary Tree | | **Information Required:**  - Aircraft information from database | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case Name:** Exit Program | | **Use Case ID:** 9 | |
| **Short Description:** Let user exit the program | | | |
| **Trigger: User pressing 9 on menu** | | | |
| **Type:** | | | |
| **Major Inputs:**  **Description:**  - Menu option | **Source:**  - User | **Major Outputs:**  **Description:**  - Close program | **Destination:** |
| **Major Steps Performed:**  - Check if the user wants to save first  - Save to file if user asks to  - Close down program | | **Information Required:**  - Binary tree and location of file if user wishes to save | |

**Sequence Diagram**

6: Display search result

1: Add aircraft

14: Stores binary tree contents in file

3: Delete aircraft

10: Display all aircrafts

13: Save to file

15: Edit aircraft

11: Load file

**Binary tree**

**Aircraft.txt**

12: Gets and stores data

8: Display search result

7: Search by call sig

9: Print out aircrafts

5: Search by name

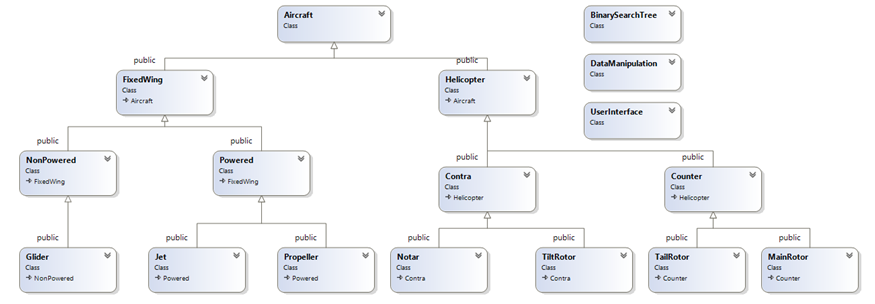
4: Remove from tree

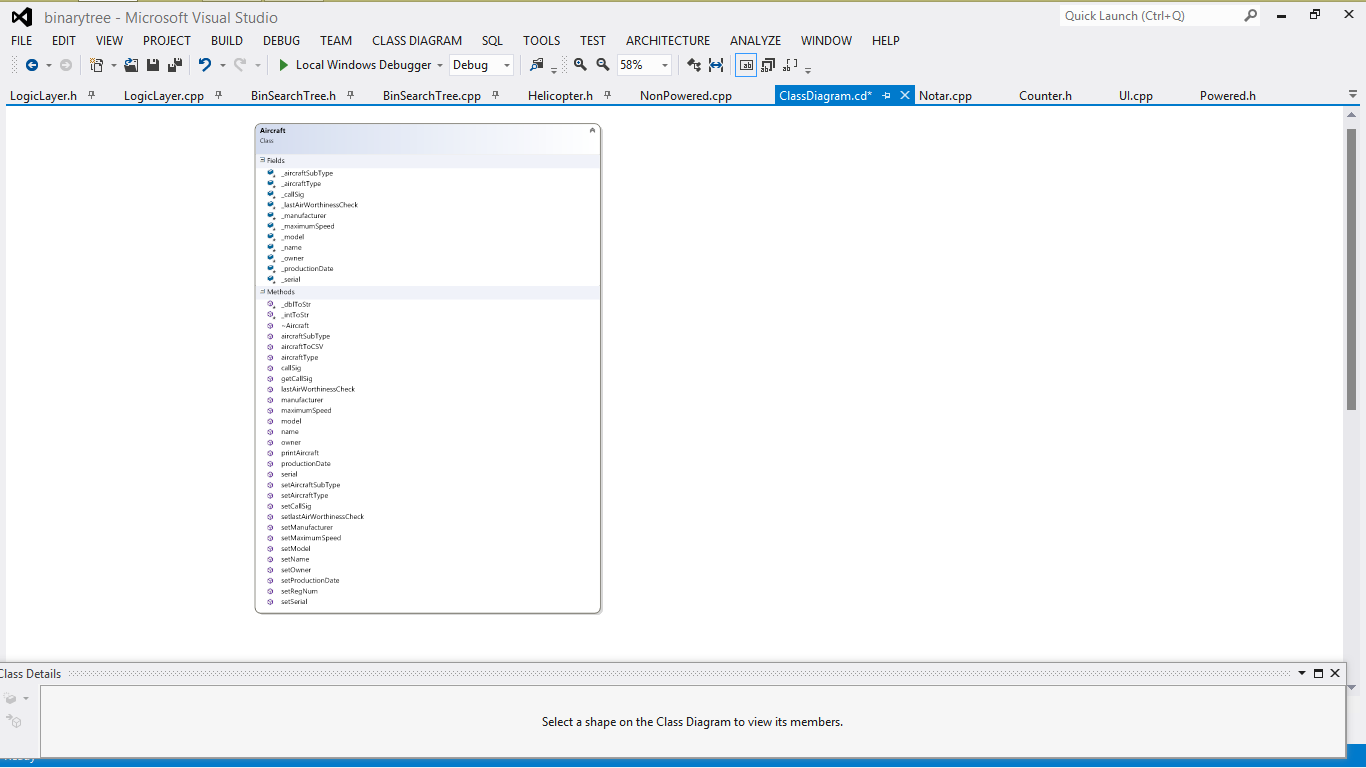
2: Insert new into binary tree

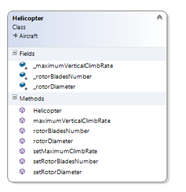
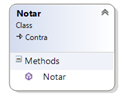
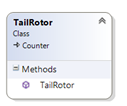
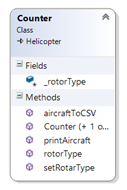
16: Change given information

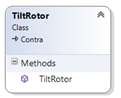
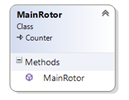
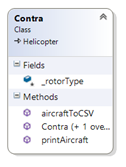
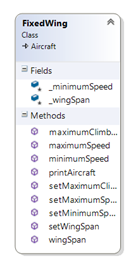
**User  
Interface**

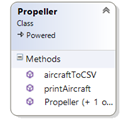
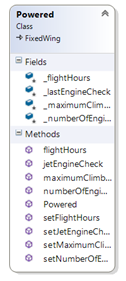
**Class diagrams**

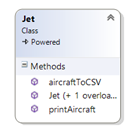


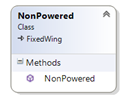
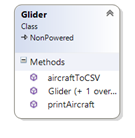


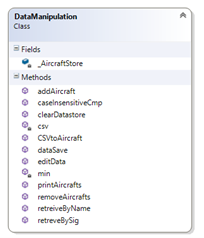


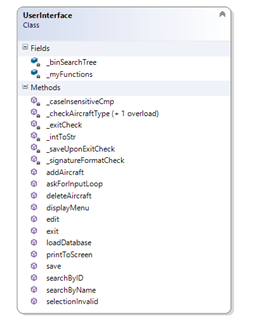


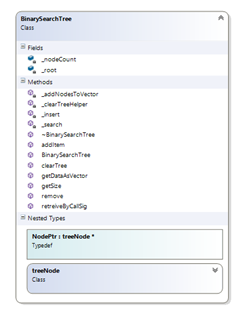












**Internal Data Structure**

The main data structure of the program is the use of a balanced binary tree, which is a dynamic data structure. This data structure uses other internal structures to maximize its effectiveness.

The most important of the data structures is our use of Nodes. The Nodes hold the data component of the Aircraft. In reality this is just a pointer to a certain piece of data ordered by the call signature of the Aircraft. The aircraft class object is made up of the multi-tier inheritance of all the types of aircraft specified, which holds the all of the aircraft data. We also have virtual functions in the Aircraft class which are only called when inherited functions call it. When the user specifies the data for an aircraft (or is inserted/parsed from the permanently stored ASCII text file), the program allocates the aircraft object (with the corresponding added data making up the object) to a new node, which is then placed in the tree at the correct position; if we would have had more time we would of balanced the tree.

The program also makes use of temporary internal data structures. The main temporary one utilized is use of vectors, which is a Standard Template Library (STL) container. The use of vectors was mostly used in the parsing of a text file that contains the permanently stored aircraft data. Vectors were used to temporarily hold the string data that was read in from the text file line by line (one line of data per element of vector) and then added to the nodes of the binary tree; again by a vector.  
Data can be read out of the tree by the use of a (getDataAsVector) function. Vectors are used so prolifically in the project as they are dynamic when it comes to data which is perfect when you don’t know the size of the data you want to add or read from the vector.

**File format**

The ASCII File Format that the aircraft information is stored is in a text file using CSV, meaning that the values for each aircraft are stored in comma separated values. When the user loads the aircrafts from the file into the system by parsing the text file by extracting the data separated by the commas and placing them in the relevant variables within the aircraft structure. This information is then placed into a binary tree (ordered by Call Sig) and that is what is used throughout the program when handling the data, then any changes made to these variables (adding, deleting or editing) can be saved back to the text file so no changes were lost.

**User interface model**

Save to file

Display Menu

Read user Input

System performs users request depending on the option selected

Add aircraft

Delete aircraft

Search by Name

Search by call sig

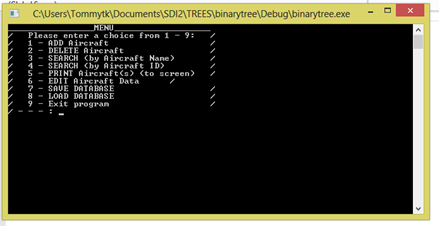
Print all

Load from file

Binary tree

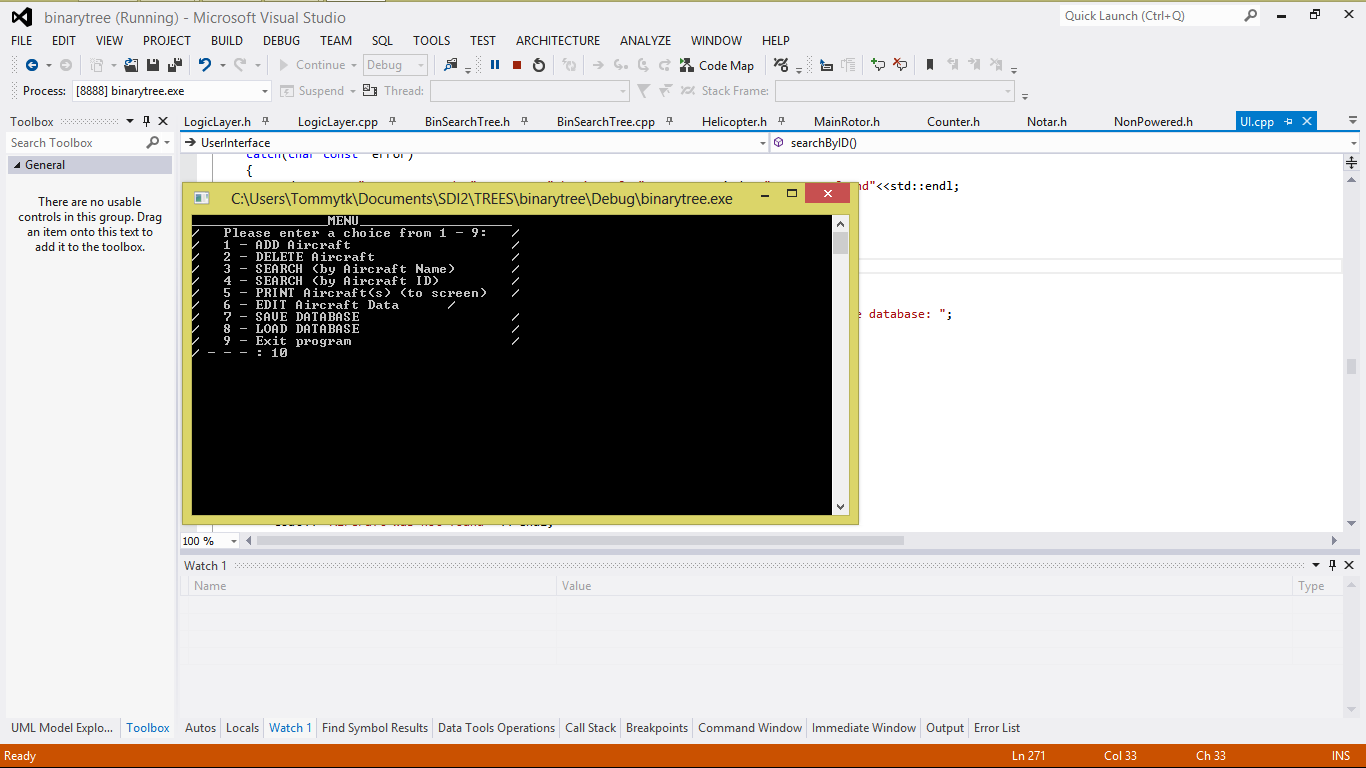
Aircraft file

Exit

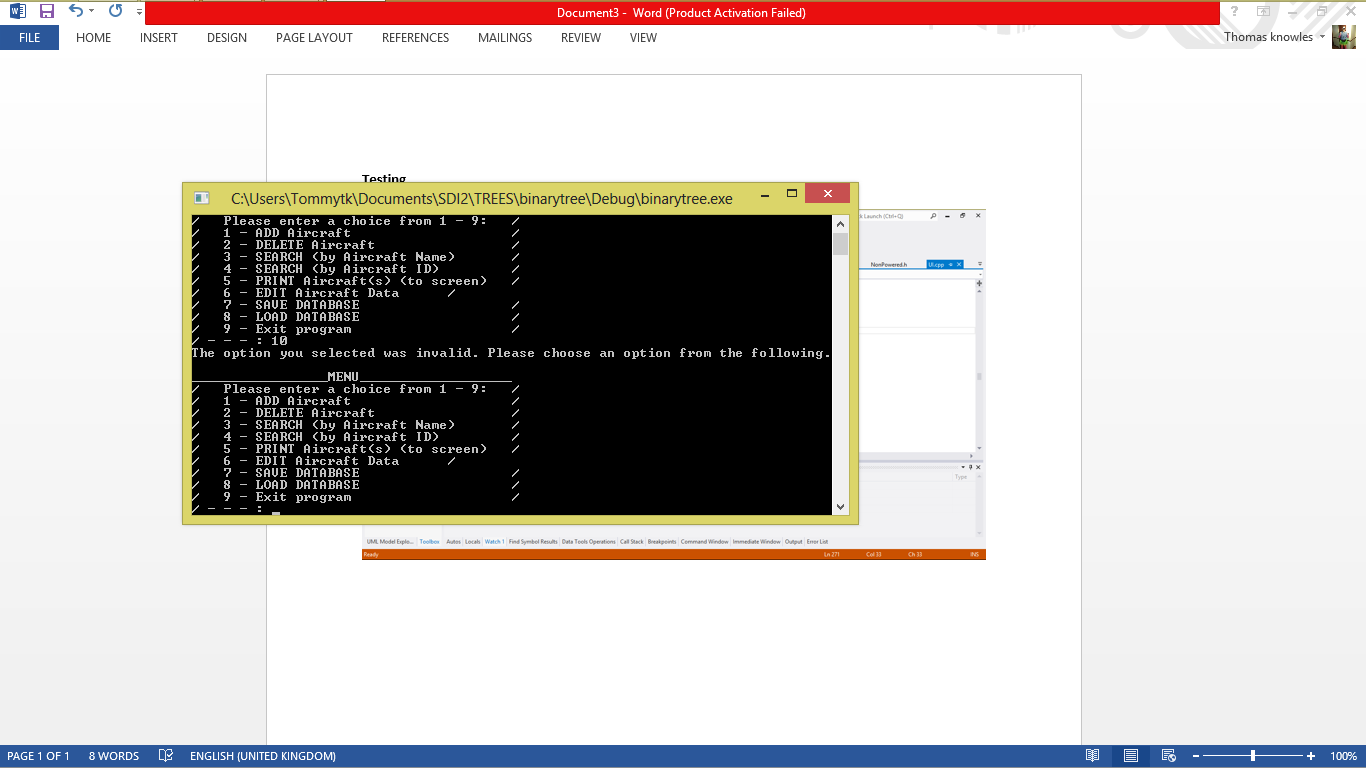
User interface screen:

**Testing:**

Bad data input when choosing menu option-

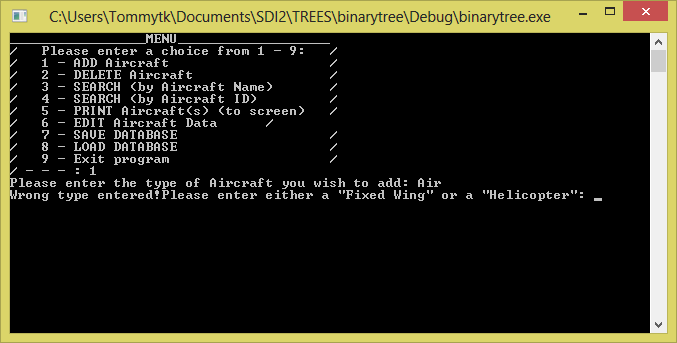


This shows the program dealing with bad inputs when selecting menu option.



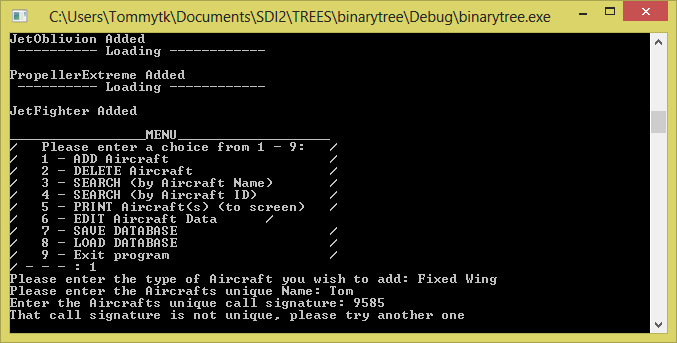
**Adding Aircraft**

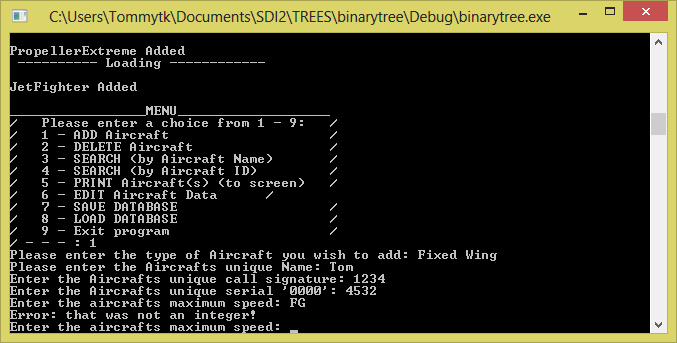
Testing whether the program handles bad data inputted when adding an aircraft.



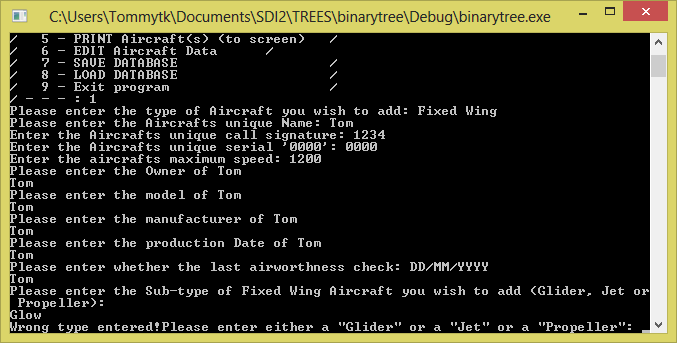
As you can see the program tests the data, validates it and decides whether the user has inputted good or bad data.

Below the program makes sure that the call signature inputted is unique. So it tests it against all the saved call signs to make sure. It then lets the user know if it is not unique.



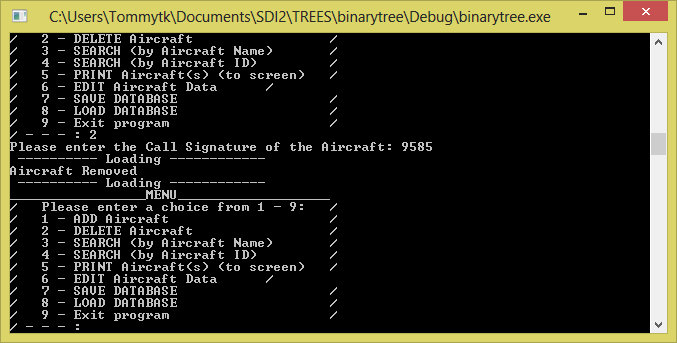


The maximum speed is also validated to make sure you enter an integer.



The program makes sure you enter only a certain 3 variables.

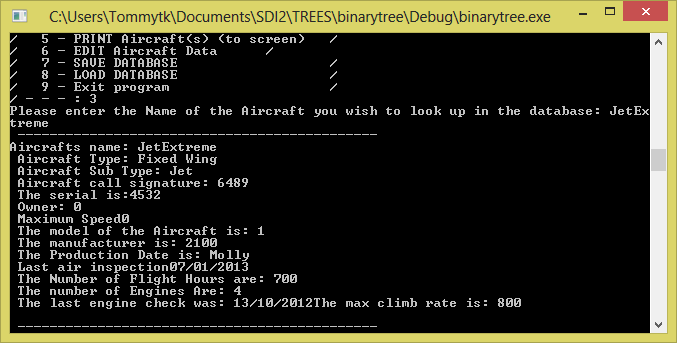
**Delete**



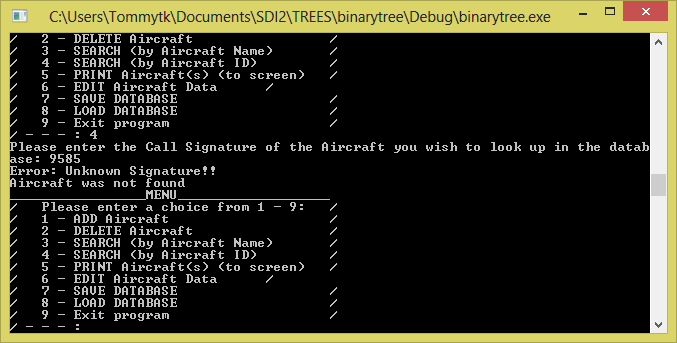
As can be seen above the delete function works as expected.

**Search By Name**

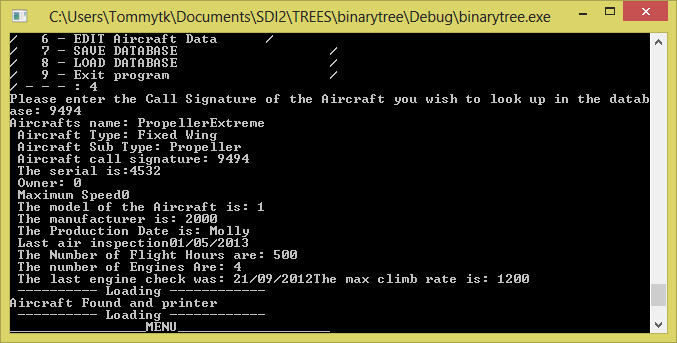
The Search by name function returns the data of the name you entered. Works as expected .



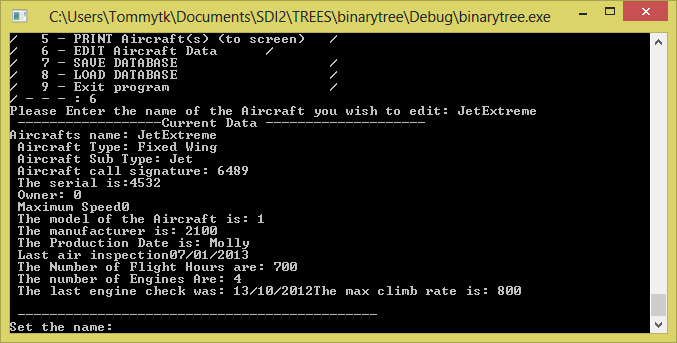
**Search by CallSig/ID**



Above shows the search by ID function handling a bad sig and below the function working as expected.

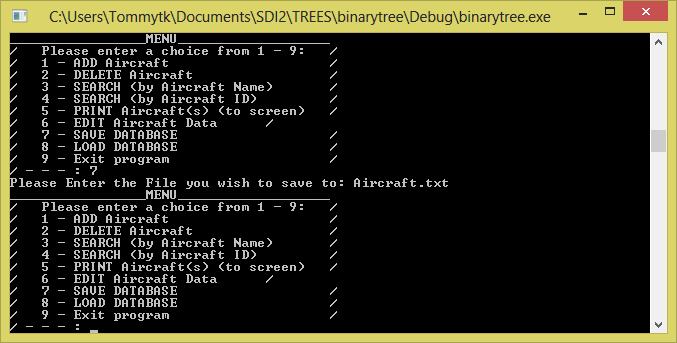


**Edit Data**



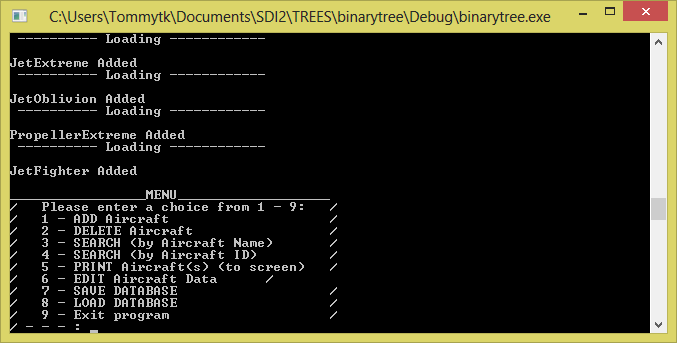
When the user inputs good data the program will work as expected. However because of the try and catch blocks. When the user inputs bad data the program throws an unhandled exception.

**Save Data**



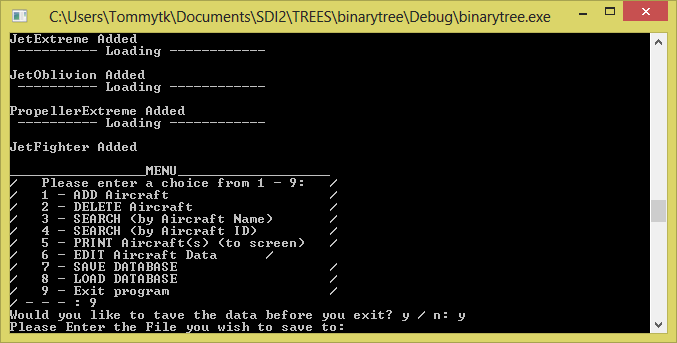
The save function works as expected.

**Load Function**



The load function lists all the data names that have been entered. Works as expected.

**Exit function**



The exit function asks if you want to save before you exit. Works as expected.