

**Software Design and Implementation 2**

Design and Implementation of an aircraft register  
 for the Ministry of Defence

Arka Mitra – N0388235

Kok Khong Thomas Cheong – N0382624

Contents

[Introduction 3](#_Toc355891084)

[Use Case Diagram 4](#_Toc355891085)

[Use Cases Description 5](#_Toc355891086)

[Sequence Diagram 15](#_Toc355891087)

[Class Diagram 16](#_Toc355891088)

[System menus and Navigation 17](#_Toc355891089)

[Internal data structures 18](#_Toc355891090)

[Aircraft 18](#_Toc355891091)

[FixedWing 19](#_Toc355891092)

[Glider 19](#_Toc355891093)

[PropellerDrivenAircraft 19](#_Toc355891094)

[Helicopter 20](#_Toc355891095)

[Node 20](#_Toc355891096)

[BinaryTree 22](#_Toc355891097)

[Global 24](#_Toc355891098)

[File Format Used 26](#_Toc355891099)

[Conclusion 26](#_Toc355891100)

[Test plan 27](#_Toc355891101)

# Introduction

Databases play a key role in most businesses today. Big organisations stores huge amount of data in databases and it allows users to quickly browse, search (specific item), add, delete and many more functionalities, so they are definitely very useful for storing data and makes searching extremely efficient.

As part of Software Design and Implementation 2 coursework it was required for the duo (Arka Mitra and Thomas Cheong) to create efficient software that allows user to add Aircraft data into the system and store it securely in a file.   
The system also allows user to search Aircrafts based on call signs (this is the primary key of aircraft because every aircraft has a specific call sign), it also allows user to edit existing Aircraft information (everything can be changed except the call sign), the system allows efficient browsing facility; so by pressing ‘5’ user can view all the existing aircrafts that are stored in the system.   
User has the option to add an aircraft as mentioned earlier and then there is additional option of actually saving the changes to the file, this means next time the program is opened the saved aircrafts in the file will get loaded.

The software is extremely robust and user friendly at the same time. A fully functional binary tree data structure has been implemented in the system for efficient searching.   
Binary search is a very inexpensive way of searching, it is extremely fast and proficient compared to linear search. Another aspect of the software that is worth mentioning is that style guide provided in NOW was followed through-out the project to make the most effective and efficient software solution.

The software was developed for Ministry of Defence so that they can store aircraft information in the system.   
Aircraft is too broad so it was divided into ‘helicopter’ and fixed wing aircraft and then fixed wing aircraft was divided into ‘jet’, ‘glider’ and ‘propeller driven aircraft’. For further information on how the classes were organised please refer to the class diagram.

# C:\Users\Asus\Desktop\AR-Use case diagram.pngUse Case Diagram

# Use Cases Description

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case Name:** Start Up System | | **ID Number:** 1 | |
| **Short Description:** When the user starts up the system, the menu screen of the program will appear, and there will be few options available to the user. | | | |
| **Trigger:** User wants run the system and manage the aircraft register. | | | |
| **Type:** | External | | |
| **Major Input Description:** When the user start up the system, the program will fetch all data stored in the external text file.  On main menu, there will be 8 available options to choose from:   * Press 1 to ‘Add Aircraft’ * 2 to ‘Remove Aircraft’ * 3 to ‘Search Aircraft’ * 4 to ‘Edit Aircraft’ * 5 for ‘Browse all Aircraft’ * 6 to ‘Clear Database’ * 7 to ‘Load data’ from text file * 8 to ‘Save data’ to text file * 9 to ‘Exit’ the program.   A different screen will appear after a number is chosen. **Source:** User | | | **Major Output Description:**  On the menu screen, the system will prompt the user to enter a number range from 1-9 for different options.  **Destination:**  User |
| **Major Steps Performed:**   1. After the user run the program, the menu screen will appear. 2. On the menu page, the user will need to enter a number between 1 to 9 for the system to change screen. 3. If user wants to go back to main menu after selected a screen, he/she can press the ‘home’ key anytime to return. | | | **Information Required:**   1. User start up system. 2. User enters a value between 1-9 to select an option. 3. Number must be between 1-9. 4. User input (9) in any page to return to main menu. |
| **Use Case Name:** Insert Aircraft | | **ID Number:** 2 | |
| **Short Description:** The user is on the main menu and wishes to insert an aircraft. | | | |
| **Trigger:** User wishes to enter a new aircraft into the system. | | | |
| **Type:** | External | | |
| **Major Input Description:** User can add new aircraft to the system using this feature. User also has the choice of selecting different aircraft types before entering its details and inserting into the system. The aircraft types includes:   * Jet (Fixed wing) * Propeller Driven Aircraft (Fixed Wing) * Glider (Fixed wing) * Helicopter (Helicopter)   The user can return the main menu by entering ‘9’.  **Source:** User | | | **Major Output Description:** The system will receive the details entered by the user and then store it to the binary tree.  If user tries to enter a invalid input or an aircraft’s call sign that already exists, the system will throw an exception (Error Message)  **Destination:**  User |
| **Major Steps Performed:**   1. User enters ‘1’ on the main menu to access the ‘insert aircraft’ screen. 2. User can select between ‘1’ to ‘4’ for aircraft types or ‘9’ to go back to main menu. 3. If user selected an aircraft type, the system will ask the user to enter the details of the new aircraft. This includes: ‘Call Sign’, ‘Serial Number’, ‘Owner’s Name’, ‘manufacturer’s name’, ‘minimum speed’, ‘maximum speed’. Depend on which aircraft type the user wish to enter (Fixed Wing) or (Helicopter), the system ask for additional details. | | | **Information Required:**   1. User selection of ‘1 – insert aircraft’ on main menu 2. User selection of ‘aircraft type’ or ‘return to main menu’ 3. User entering details of aircraft to the system. |

|  |  |  |
| --- | --- | --- |
| **Use Case Name:** Remove Aircraft | | **ID Number:** 3 |
| **Short Description:** The user is on the main menu and wishes to remove an aircraft. | | |
| **Trigger:** User wish to remove an aircraft from the database system. | | |
| **Type:** | External | |
| **Major Input Description:**  Assuming that the user is on the main menu and wish to remove an aircraft from the system, he/she can select ‘2’ which bring the user to the removing aircraft page.  On the ‘remove’ page, the system will ask the user to enter the call sign of the aircraft in which he/she wish to delete.  If the aircraft is found, the system will ask the user for confirmation. Enter ‘y’ to continue the removing or ‘n’ to cancel the process.  **Source:**  User | | **Major Output Description:**  The system will receive the call sign entered by the user and search the system thoroughly to find a matching call sign.  If aircraft is found, the system will ask the user if he/she really wants to delete the aircraft. If ‘yes’ then the system will delete the aircraft and return the user back to main menu.  If no matching aircraft is found, the system will show a message and send the user back to main menu.  **Destination:**  User |
| **Major Steps Performed:**   1. User enters ‘2’ on the main menu to access the remove aircraft page. 2. User enters a call sign into the system. 3. If a **matching aircraft is found**, the user can either enter ‘y’ to delete the aircraft or ‘n’ to stop the process. 4. If **no aircraft is found** the system will return the user back to main menu. | | **Information Required:**   1. User enters ‘2’ on the main menu to get to the ‘remove’ function. 2. User enters a ‘call sign’ to delete. 3. If the aircraft is found, user can either enter ‘y’ or ‘n’. 4. If no aircraft is found, the user will return the main menu. |
| **Use Case Name:** Edit Aircraft | | **ID Number:** 4 |
| **Short Description:** The user is on the main menu and wishes to edit the details of an aircraft. | | |
| **Trigger:** User wishes to edit the details of an aircraft in the database system. | | |
| **Type:** | External | |
| **Major Input Description:**  In this feature the user can edit the details of an existing aircraft in the system. To get to this feature, the user will need to enter ‘4’ on the main menu. Once the user is in the ‘Edit’ page, he/she will need to enter the call sign of the aircraft in which he/she wish to edit. If there is a matching aircraft, the user can change the aircraft type and enter new details for that aircraft.  If no aircraft is found, the system will take the user back to main menu. **Source:**  User | | **Major Output Description:**  The system will receive the call sign entered by the user and search the system thoroughly to find a matching call sign.  If aircraft is found, the system will prompt the user to select a new aircraft type and new details for that aircraft.    **Destination:**  User |
| **Major Steps Performed:**   1. User enters ‘4’ on the main menu to access the ‘edit’ aircraft page. 2. User enters a call sign into the system. 3. If a **matching aircraft is found**, the user have the options to change the aircraft type and enter new details for that aircraft. 4. If **no aircraft is found** the system will return the user back to main menu. | | **Information Required:**   1. User enters ‘2’ on the main menu to get to the ‘remove’ function. 2. User enters a ‘call sign’ to edit. 3. If the aircraft is found, user can follow on-screen instruction to change the details. 4. If no aircraft is found, the user will return the main menu. |

|  |  |  |
| --- | --- | --- |
| **Use Case Name:** Browse all Aircraft | | **ID Number:** 5 |
| **Short Description:** The user is on the main menu and wishes to browse all aircraft stored on the system. | | |
| **Trigger:** User wishes to browse all aircraft stored in the database system. | | |
| **Type:** | External | |
| **Major Input Description:**  To browse all the aircraft stored in the system, the user will need to enter ‘5’ on the main menu. Once the user is on this page, the system will automatically display all the aircrafts.  If no aircraft is stored on the system, the program will display a message (No aircrafts) on the screen and return the user back to main menu.  **Source:**  User | | **Major Output Description:**  Once the user gets to this page, the system will display all the aircrafts on the screen. If there are no aircrafts stored on the system, the program will redirect the user back to main menu.    **Destination:**  User |
| **Major Steps Performed:**   1. User enters ‘5’ on the main menu to access the ‘browse all aircraft’ page. 2. System display all aircrafts 3. If no aircraft stored on the system, user will be redirected to main menu. | | **Information Required:**   1. User enters ‘5’ on the main menu to get to the ‘browse all aircraft’ function. 2. Aircraft stored in the system. |

|  |  |  |
| --- | --- | --- |
| **Use Case Name:** Clear Database | | **ID Number:** 6 |
| **Short Description:** The user is on the main menu and wishes to clear aircraft database. | | |
| **Trigger:** User wish to clear all aircrafts stored in the database system. | | |
| **Type:** | External | |
| **Major Input Description:**  Assuming that the user is on the main menu and wishes to clear all aircrafts stored in the database system, he/she can select ‘6’, and this will bring the user to ‘clear database’ function.  Before the user can clear the database, the system will ask the user for confirmation. If the user select ‘y’ which represent yes, then the system will continue its process and wipe out the database.  If the user wishes to cancel the procedure, he/she can enter ‘n’ to return to main menu.  **Source:** User | | **Major Output Description:**  One the user gets to this page, the system will ask for user confirmation. If user wishes to continue, the system will clear out the database completely. Then the system will display a successful message.  If no aircraft is stored on the database, the system will display a message(No aircraft on database)  **Destination:** User |
| **Major Steps Performed:**   1. User enters ‘6’ on the main menu. 2. User enters ‘y’ or ‘n’ for confirmation. 3. If user entered ‘y’, then the system will check if the database contain one or more aircrafts. If there are, then the system will clear it, otherwise a message (no aircraft on database) will be displayed. 4. If user entered ‘n’ then he/she will be redirect to main menu. | | **Information Required:**   1. User enters ‘5’ on the main menu. 2. User enters ‘y’ yes or ‘n’ no for confirmation 3. Aircraft database. |
| **Use Case Name:** Load data from text file | | **ID Number:** 7 |
| **Short Description:** The user is on the main menu and load data from an external text file. | | |
| **Trigger:** User wishes to load data from text file into the system. | | |
| **Type:** | External | |
| **Major Input Description:** When the user is on the main menu, and wishes to import data from a text file, he/she can enter ‘7’ to activate the loading function.  **Source:**  User | | **Major Output Description:**  Once the user selected this function, the system will fetch all the data from an external text file and then store them in the system database.  If text file not found, then it will display a error message.    **Destination:**  User |
| **Major Steps Performed:**   1. User enters ‘7’ on the main menu. 2. System quickly wipes out existing data stored in the database. 3. System load all data from text file into database. 4. If text file not found, then the system will display an error message. | | **Information Required:**   1. User enters ‘7’ on the main menu. 2. Aircraft stored in the system. 3. Data stored in the text file. |
| **Use Case Name:** Save Data to text file | | **ID Number:** 8 |
| **Short Description:** The user is on the main menu and wishes export data to a text file. | | |
| **Trigger:** User wish to save data stored in the database into an external text file. | | |
| **Type:** | External | |
| **Major Input Description:**  Assuming that the user is on the main menu and wishes to save export data stored in the system to an external text file. To do so, the user will need enter ‘8’ on the main menu.  The system will ask the user for confirmation, before it save the data to text file. To continue the process, the will need to enter ‘y’, and if the wish to cancel, he/she can enter ‘n’.  **Source:** User | | **Major Output Description:**  One the user gets to this page, the system will ask for user confirmation. If user wishes to continue, the system will save data to the text file. Then the system will display a successful message.  If user enters ‘n’ the system will redirect user back to main menu.  **Destination:** User |
| **Major Steps Performed:**   1. User enters ‘8’ on the main menu. 2. User enters ‘y’ or ‘n’ for confirmation. 3. If user entered ‘y’, then the system will export data to the text file. 4. If user entered ‘n’ then he/she will be redirect to main menu. | | **Information Required:**   1. User enters ‘8’ on the main menu. 2. User enters ‘y’ yes or ‘n’ no for confirmation 3. Aircraft database. 4. External text file |
| **Use Case Name:** Shutdown program | | **ID Number:** 9 |
| **Short Description:**  A process where user wants to exit the system | | |
| **Trigger:** User wish to exit the system. | | |
| **Type:** | External | |
| **Major Input Description:** When the user is on the main menu, he/she can press ‘9’ to quit the program.  **Source:**  User | | **Major Output Description:**  Once the user entered ‘9’the program will shut down.      **Destination:**  User |
| **Major Steps Performed:**   1. User enters ‘9’ on the main menu. 2. Shutdown system | | **Information Required:**   1. User enters ‘9’ on the main menu. 2. Quits the program. |

|  |  |  |
| --- | --- | --- |
| **Use Case Name:** Search Aircraft | | **ID Number:** 10 |
| **Short Description:** The user is on the main menu and wishes to search an aircraft. | | |
| **Trigger:** User wishes search for an aircraft in the database system. | | |
| **Type:** | External | |
| **Major Input Description:**  In this feature the user can search for an existing aircraft in the system. To get to this feature, the user will need to enter ‘3’ on the main menu. Once the user is in the ‘Search’ page, he/she will need to enter the call sign of the aircraft. If there is a matching aircraft, the system will display the details of that aircraft.  If no aircraft is found, the system will take the user back to main menu.  **Source:** User | | **Major Output Description:**  The system will receive the call sign entered by the user and search the system thoroughly to find a matching call sign.  If aircraft is found, the system will display the details of that aircraft.  Otherwise, the system will display an error message (No aircraft is found) on the screen.    **Destination:**  User |
| **Major Steps Performed:**   1. User enters ‘3’ on the main menu to access the ‘search aircraft page. 2. User enters a call sign into the system. 3. If a **matching aircraft is found**, the system will display the details of that aircraft. 4. If **no aircraft is found** the system will return the user back to main menu. | | **Information Required:**   1. User enters ‘3’ on the main menu to get to the ‘remove’ function. 2. User enters a ‘call sign’. 3. System Database |

# Sequence Diagram

# G:\Aircraft Project - Final\aircraftSSD.pngClass Diagram

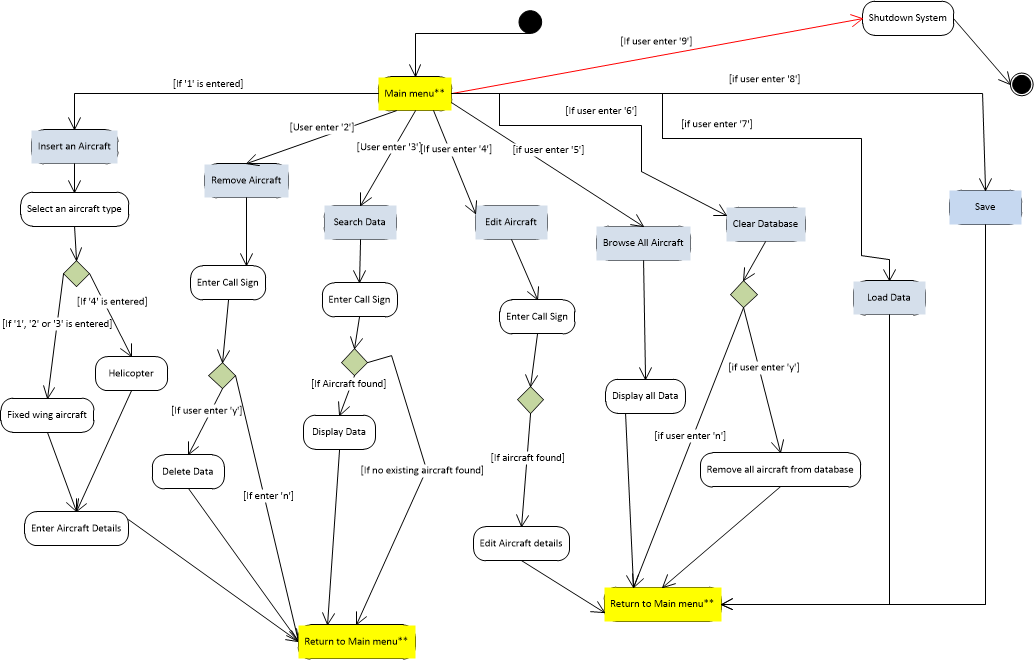
# C:\Users\Asus\Dropbox\SDI 2 Coursework\Report\aircraftCD.png

# System menus and Navigation

When the user starts up the program, there will be 9 options available, these includes:

* 1 - Insert aircraft
* 2 – Remove an aircraft
* 3 - Search an aircraft
* 4 - Edit an aircraft’s details
* 5 – Browse all aircraft
* 6 – Clear Database
* 7 – load Data from text file
* 8 – Save data to text file
* 9 – Quit program

Below is a flowchart illustrating the system functionality and a sense of how it navigates within the system:



# Internal data structures

List of data structures used in the software are as follows:

### Aircraft

Aircraft stores information about aircraft. This is the root in the inheritance hierarchy, in other words this is a base class. The class also has operator over loaders which are used extensively to compare call signs, this had to be implemented into the class because C++ by default don’t know how to compare it. Below is a screen shot of Aircraft class, it shows all the variables and functions that are used.

classAircraft

{

public:

Aircraft(void);//constructor

~Aircraft(void);//destructor

Aircraft(intcallSign);

//operator overloaders (for Aircraft callSign comparison)

booloperator<(AircraftcP);

booloperator>(AircraftcP);

booloperator!=(AircraftcP);

booloperator==(AircraftcP);

booloperator<=(AircraftcP);

booloperator>=(AircraftcP);

//Setters

voidsetCallSign(int);

voidsetSerialNumber(int);

voidsetOwnerName(string);

voidsetProductionDate(string);

voidsetManufacturer(string);

voidsetMinSpeed(double);

voidsetMaxSpeed(double);

voidsetMaxClimbRate(double);

voidsetType(string);

//Getters

intgetCallSign(void);

intgetSerialNumber(void);

stringgetOwnerName(void);

stringgetProductionDate(void);

stringgetManufacturer(void);

doublegetMinSpeed(void);

doublegetMaxSpeed(void);

doublegetMaxClimbRate(void);

stringgetType(void);

private:

string\_type;

int\_callSign;//stores unique number

int\_serialNumber;

string\_ownerName;

string\_productionDate;

string\_manufacturer;

int\_minSpeed;

int\_maxSpeed;

int\_maxClimbRate;

};

### FixedWing

Fixed wing is a derived class that derives from aircraft, so this is level 2 in the inheritance hierarchy. This has one extra variable that Aircraft doesn’t and that is wingSpan.

classFixedWing:publicAircraft

{

public:

FixedWing(void);//constructor

~FixedWing(void);//destructor

//Setters

voidsetWingSpan(int);

//Getters

intgetWingSpan(void);

private:

int\_wingSpan;

};

**Jet**

The Jet class is the leaf in other words the inheritance terminates in this level. This is the third level of inheritance. Jet derives from FixedWing.

classJet:publicFixedWing

{

public:

Jet(void);

~Jet(void);

private:

};

### Glider

The Glider is another leaf, Glider is also derived from the FixedWing aircraft so it has all FixedWing variables as well as Aircraft variables.

classGlider:publicFixedWing

{

public:

Glider(void);

~Glider(void);

};

### PropellerDrivenAircraft

The PropellerDrivenAircraft is another leaf, it is derived from FixedWing aircraft so it has all FixedWing’s variables as well as Aircraft’s variables.

classPropellerDrivenAircraft:publicFixedWing

{

public:

PropellerDrivenAircraft(void);

~PropellerDrivenAircraft(void);

private:

};

### Helicopter

The Helicopter class is derived from the Aircraft class. It has all the Aircraft’s variables and in addition it has two new variables as displayed in the picture below.

classHelicopter:publicAircraft

{

public:

Helicopter(void);//constructor

~Helicopter(void);//destructor

//Setters

voidsetRotorDiameter(int);

voidsetNoOfRotorBlades(int);

//Getters

intgetRotorDiameter(void);

intgetNoOfRotorBlades(void);

private:

int\_rotorDiameter;

int\_noOfRotorBlades;

};

## Node

The Node class is extensively used in the binary tree. An instance of a Node class can store an instance of Aircraft class and then the instance of the Node class (with instance of Aircraft class) can be stored in an instance of a BinaryTree class.   
So logically thinking each Node really represents an instance of Aircraft. Also dude to use of polymorphism instance of Jet or any class that is derived from Aircraft can be stored in a Node and the BinaryTree.

The Node has a Node pointer to left and right and there is \_keyValue of type T. The Node class uses template so that any data type can be stored in it. The \_keyValue stores instance of Aircraft class. Left and right are mostly used for navigation inside the binary tree.   
The Node has a remove function that finds the node to remove (based on the key that is passed in) and also re-arranges the whole binary tree and returns the node to remove to the remove function in the binary tree and then it just simply gets deleted.

The destroy function is designed to destroy the whole binary tree. The way it works is it basically gets called in the destroy function of the binary tree and then it iterates through each node by going left and right and check if it is not equal to null, and finally deletes the node. It does it multiple times until the tree is empty.

The Node consists of a minValue function that returns the minimum value in the tree.   
This gets called in the remove function. There is also a getSuccessor function with return type of vector. What this function does is it gets all the nodes by checking if left != null and right \_= null and then inserts everything in a vector which later gets used in the global class.

So below is the definition of the Node class with all the member functions and variables.

template<classT>

structNode

{

public:

//setters and getters

//set \_keyValue

voidsetKey(T\*key);

//returns \_keyValue

T\*getKey();

//set left node

voidsetNodeLeft(Node<T>\*left);

//get left node

Node<T>\*getNodeLeft();

//set right node

voidsetNodeRight(Node<T>\*right);

//get right node

Node<T>\*getNodeRight();

//delete a particular node

Node<T>\*remove(Tkey,Node<T>\*parent);

//gets the min value in tree

T\*minValue();

//destroy node

voiddestroy();

//returns all successors

vector<T\*>Node<T>::getSuccessors();

private:

//stores T pointer variable ... \_keyValue

T\*\_keyValue;

Node<T>\*\_left;

Node<T>\*\_right;

void\_destroy();

T\*\_minValue();

};

## BinaryTree

The BinaryTree class is the most important class in the program. This is where information is temporarily stored. Without the binary tree the program would have been useless. The binary tree has a range of useful functions.

The add function of the binary tree lets the user add instance of any data structure including any user defined class types.   
This is possible because template was used in the binary tree. In the binary tree there are two insert functions, firstly there is a private insert function which simply checks if the root is empty if it is empty then the data gets added into it but it is not empty then the private version (\_insert()) gets called and this handles anything other than the root.   
So the \_insert() function checks if the key is less than key of current node it is checking, if key is less than current node’s key then \_insert() function is called with node->getNodeleft() as a parameter in \_insert(). The same process occurs for right hand side until a free space is found. If a free space is found then a new node is created on the space which holds an aircraft.

Along with the insert the binary tree has a search function. This is a binary search, hence it is a very efficient and inexpensive searching method that is extremely fast. It checks if current node’s key is equal to the key that has been passed in by the user.   
If the keys doesn’t match then it follows similar pattern to the add, a search function is called by passing node->getNodeLeft(). This is recursion and it will continue this process until it finds the key. The same process happens for the right until the key is found.

The binary tree also has a remove function which removes a node from the tree by searching the key. In this function the user passes in the key and then the remove function finds the key, rearranges all the nodes around it and then finally deletes the node itself.   
This function took a long time to code and throughout development it was very buggy and gave us many errors, but finally as of now the remove function works perfectly and this can be confirmed after a lot of testing on remove() function.

Below is the code for the binary tree class.

//use of template means this binary tree can store any datatypes (including user defined classes)

template<classT>

classBinaryTree

{

public:

//constructor

BinaryTree(void);

//destructor

~BinaryTree(void);

//remove (this calls \_remove)

voidremove(Tkey);

//insert - this insert only checks root, if empty it fills it else it directs to \_insert() (it check all other roots as it iterates through the tree)

voidinsert(T\*key);

//search

T\*search(Tkey);

//destroy tree!

voiddestroy();

//convert tree to vector

vector<T\*>BinaryTree<T>::vectorConversion();

private:

//make root

Node<T>\*\_root;

//gets to this insert when root is not empty

void\_insert(T\*key,Node<T>\*node);

//binary search

Node<T>\*\_search(Tkey,Node<T>\*node);

//\_destroy

void\_destroy();

//\_remove

void\_remove(Tkey);

};

## Global

Throughout the project object oriented techniques were significantly followed. It was decided that a class called Global would be created to keep the main() function nice and tidy. The Global class can be called an interface because it is the main media through which user interacts with the system. Most functions in Global class is private. There are few public functions but they just calls the private versions of the function, this is again to protect code from other developers who might try to add code to the system.

In Global class definition there are only two private member variables. Firstly there is \_isRunning, this is used to control the while loop and secondly an instance of BinaryTree is created by creating BinaryTree<Aircraft> \_database.

The constructor of Global sets \_isRunning to true so that the main loop starts straight away. Global’s destructor destroys the \_database, i.e. the destroy() function gets called. There is a \_setupConsole() function that resizes the console window size (although this is only applicable if the current console window size is actually bigger than the window size mentioned in the code), so in most cases it will not work but it was worth an effort.

The \_load function opens up the text file aircraft.txt and reads everything from it. Similarly there is a \_save function that saves all information to the same text file. So this means for instance in aircraft.txt there is ‘x’ amount of aircrafts, after user inserts few more aircrafts (‘y’) to the binary tree (i.e. the \_database) the user has the option to save and once they do that the new aircrafts will be saved into the text file i.e. ‘x + y’.

If the user decides to add user to the database then \_add() function will get called. Once the add is called the user will get an option to choose between Jet, Propeller driven aircraft, Glider and Helicopter. Whichever option the user selects will be the type variable of Aircraft. And then the user will have to insert all other data such as callSing, manufacturer, etc. Finally inside the \_add() function the BinaryTree’s insert() function gets called and this adds everything to the tree.

User has the option to edit an aircraft.   
The \_edit() function does not lets the user change an aircraft’s callSign however all other information can be changed it behaves somewhat similar to the \_add() function. Along with \_edit() the user has the option to browse, the \_browse just displays everything that is stored in the BinaryTree.

The \_search() function allows the user to search for a specific aircraft based on the callSign they enter. The search function in Global calls BinaryTree’ssearch() function. When the delete function of the BinaryTree gets called the aircraft with the callSign entered gets deleted instantly.

Finally there is the \_run() function. This function is the core interface where the main user interaction takes place. The run function handles user very well. Basically all the functions mentioned above in under the Global class section gets called in the \_run() function and finally the run() gets called in the main() function. The run() is basically a public void function that calls \_run() (a private void function). So this is the Global class and just a brief explanation on how it operates in the system and what its roles are.

Below is the code in the Global class’s definition.

#pragmaonce

//define all libraries required

#include"Main.h"

#include"BinaryTree.h"

#include"Aircraft.h"

#include"Helicopter.h"

#include"FixedWing.h"

#include"Jet.h"

#include"Glider.h"

#include"PropellerDrivenAircraft.h"

usingnamespacestd;

class Global

{

public:

//constructor

Global(void);

//destructor (destroy database i.e. the tree)

~Global(void);

//this function calls the private \_setupConsole() function

voidsetupConsole();

//this calls the private \_run() function

void run();

private:

//sets up console's window size

void \_setupConsole();

//this function tackles user interaction

void \_run();

//use this function for saving to text file

void \_save(string fileName);

//use this function for loading from text file

void \_load(string fileName);

//make instance of binary tree (so taht it can store aircraft)

BinaryTree<Aircraft> \_database;

//keeps track in the main while loop, if this changes to false the program will close

bool \_isRunning;

//add function

void \_add(intuserInput);

//search function

void \_search(intenteredCallSign);

//display all aircrafts

void \_browse();

//edit aircraft

void \_edit(intenteredCallSign);

};

//Client side

void main()

{

Global global;

global.setupConsole();

global.run();

}

# File Format Used

The file format used for storing the data is called Comma Separated Values (CSV). CSV is a type of data format in which each part of data is separated by a comma.  
Since the program has exporting and importing features, the team decided that the best way to achieve it is by using a file format that can be easily loaded to different file edit applications.   
Furthermore, the CSV can be saved in ASCII(American Standard Code for Information Interchange)standard. ASCII is a character encoding scheme based on the English alphabet which assigns [letters](http://www.computerhope.com/jargon/l/letter.htm), [numbers](http://www.computerhope.com/jargon/n/number.htm), and other [characters](http://www.computerhope.com/jargon/c/charact.htm) within the 256 slots available in the [8-bit](http://www.computerhope.com/jargon/num/8bit.htm) code. In short term, it is a standard language that can be displayed on computers.  
By using the CSV stored in this standard, the editing of the data can be done effortlessly and efficiently.

# Conclusion

To sum up creating this software was a fun experience, at times it was extremely hard since pointers were used throughout the project and Visual Studio did not specify the root cause of many errors that occurred, and it was very difficult identifying the errors.   
A lot of new techniques have been learnt while developing the software. It can also now be officially confirmed that this software was strictly hard coded by Arka Mitra and Thomas Cheong and hence they are the owner of this software.

# Test plan

|  |  |
| --- | --- |
| **Test criteria** | **Outcome** |
| Does the program start up and is the main menu displaying correctly? (1-9 options) | **Successful**.  The program run without showing any error. The main menu options are also displaying correctly. Below is a screen shot of the program just loaded up. |
| Can user insert one or more aircraft into the system?  Test object –  Aircraft CallSign:882  Aircraft type: Propeller driven aircraft | **Successfully** inserted a Propeller driven aircraft into the system.  The program allow user to enter an aircraft by going into the ‘1’ insert feature.  Below is a screenshot illustrating an aircraft is added to the system. |
| What if the user tries to enter an aircraft to the system with a call sign that already exists in the database?  Does the program prevent duplicate call signs?  Aircraft  Test Object: CallSign:882 | **Successfully** prevented the user trying to insert an aircraft with the same call sign. |

|  |  |
| --- | --- |
| Does the system allow user to search for an aircraft by its call sign?  And does the system show a message when the call sign entered doesn’t match any aircraft?    Test Object –  CallSign:882  And Call sign: 288 | **Successful for both test.**  When entered call sign 882 into the system for searching, the program reply and display with all the aircraft details.    When entered call sign 288 for searching, the program show an error message(Aircraft was not found) |
| Does the system allow user to remove an aircraft? And does the system ask for confirmation before deleting it?  Test Object –  Call sign: 882 | **Successfully removed the aircraft from database** |
| Can user edit the details of an existing aircraft?  Test Object – | **Successfully edited the aircraft details.**  Below is a screen shot of the process: |

|  |  |
| --- | --- |
| Can user browse all the aircraft stored on the system? | **Successful.**  By using ‘5 - Browse’ feature, the system has retrieve all the aircraft stored on the system and display them on screen. |
| Does the system clear out the database when user selected ‘6 – Clear All Aircraft’?  If it does, will there still be any aircraft left? | **Successful on both occasions.**  Below is a screen shot of the process (Clearing the database) Notice that the system will ask for confirmation too.    When trying to browse all aircraft after clearing the database, the system displayed the following message. |

|  |  |
| --- | --- |
| Does the system load the data from a text file when a user used the ‘7 – Load’ feature? | **Successful.**  The database is cleared following by the previous test. After loading the data from an external text file, the data returned back to the system. |
| Does the system allow user to save data stored on the system to an external data?  Test Object –  Try inserting a new aircraft with the call sign ‘777’ and then save. Check the text file | **Successful**  A new aircraft with a call sign ‘777’ is inserted into the database.  When trying to save the data to a text file, the system will ask for a confirmation.    Below is a screen shot of the newly added aircraft saved to a text file successfully: |
| Does the system allow user to quit the program? | **Successful.**  When entered ‘9’ on the main menu, the program will shut down. |