-

CS162 Assignment 2: Classes Airplane Information Database

For this assignment, write a C++ program that keeps track of some single engine aircraft, but this time, use a set of two classes. To the user of the program, this version will seem just like the last version (from assignment 1), but it will work very differently. Modern programs should be portable, are written by teams, and need to be updated from time to time, so even though you will code this assignment by yourself, you will be implementing two key object-oriented programming concepts: **encapsulation** and **data hiding**. Encapsulation is the reason why we need two classes. The first class, Airplane, will be very similar to the struct you created from assignment 1, but will have encapsulated member functions. The other class you will create is for a collection of airplanes. We use collections so often for things that we have names for them. For example, a deck of cards or a flock of birds. A name for the class for a collection of airplanes could be Fleet. Or you could use a longer name such as AirplaneCollection. Either way, start the name of the class with a capital letter. See below for examples of what the classes might look like. Continue to use the same format from assignment 1 for loading from a file, presenting a menu driven loop, error checking etc. Load the input file at the start of the program, close it and only deal with data in memory, and then write the data back to the output file at program termination.

Requirements:

The Fleet class must encapsulate the array of Airplane class objects, and the array must be private. **You may not have a separate array of Airplane objects defined anywhere else, such as main() or as a global**. All functions that manipulate data for the Fleet class (loading from the file, searching for an airplane, etc.) must be encapsulated inside of the Fleet class. In other words, they must be member functions, commonly referred to as methods. All member variables (properties) of the Airplane class must be encapsulated and private inside of the Airplane class. These include make, model, maximum fuel, empty weight, engine horsepower, and cruise speed. All functions that manipulate data for the Airplane class must be encapsulated inside of the Airplane class, commonly referred to as getters and setters.

You must continue to use only c-strings (null-terminated char arrays) to store words and not use <string> or any container classes such as <vector>. You must use functions and have multiple code files and header files. You must follow the coding style listed in the C++ style guidelines.

After completing this assignment, you will be able to:

- Write code using classes and objects.
- Implement the default constructor for a user-defined class.
- Organize source code in multiple header and implementation files (.h and .cpp files).
- Implement and use accessors and mutators for a class.
- Use C-strings to represent character strings.

Tasks:

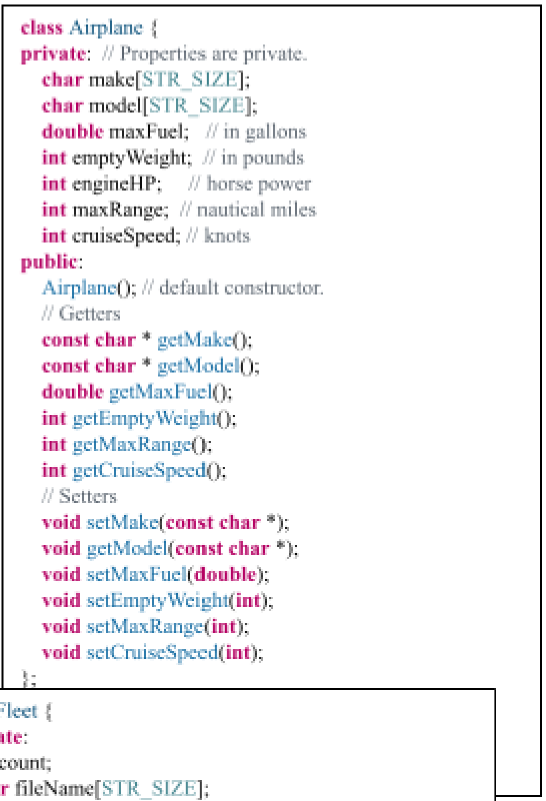
- Complete the zylabs from chapter 17 and 18. 17.11, Artwork Label and 18.4, Car Value. There are examples and a video in section 17.10, Nested Classes, that may help with the concepts in this assignment. It has a PlayList class, which is a collection of SongType objects. For chapters 17 and 18, please complete the zylabs with a score of 100%, 70% of the challenge activities, and 80% of the participation activities.
- Open the Algorithm Design Document or the UML based Algorithm Design Document, make a copy, and follow the steps to create your algorithm. Complete the algorithm design document to receive a grade.
- If you are tired of creating pseudocode and would like a graphical option, you can use this alternative UML based Algorithm Design Document. It has an example of a UML graph instead of pseudocode.

• Load the data from the database file in sorted order by name. You may not use a user-defined sorting function or sort function from a library - this means that each airplane must be inserted into the array in sorted order.

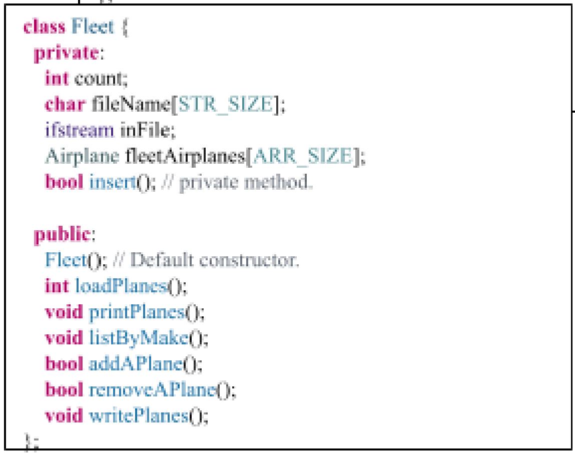
• Load the data when the program is started and write the data back to the file at program termination. Only load from the file at the start of the program. Only write to the file at program termination.

• After the file is loaded, the program will enter a user-controlled loop. Inside of the loop, the program will ask the user what they want to do.

• The rest of the tasks are split into 3 different versions based on the type of grade you want, just like assignment 1. There are 3 versions for this assignment, and each version has the minimum requirements for a particular grade – A, B or C. The B version builds on the requirements of the C version, and the A version builds on the requirements of the A version. Here are the requirements for each version:



```
class Airplane {
private:  // Properties are private.
   char make[STR_SIZE];
   char model[STR_SIZE];
   double maxFuel;   // in gallons
   int emptyWeight;  // in pounds
   int engineHP;     // horse power
   int maxRange;  // nautical miles
   int cruiseSpeed; // knots
public:
   Airplane(); // default constructor.
   // Getters
   const char * getMake();
   const char * getModel();
   double getMaxFuel();
   int getEmptyWeight();
   int getMaxRange();
   int getCruiseSpeed();
   // Setters
   void setMake(const char *);
   void getModel(const char *);
   void setMaxFuel(double);
   void setEmptyWeight(int);
   void setMaxRange(int);
   void setCruiseSpeed(int);
};
```



```
class Fleet {
 private:
   int count;
   char fileName[STR_SIZE];
   ifstream inFile;
   Airplane fleetAirplanes[ARR_SIZE];
   bool insert(); // private method.

 public:
   Fleet(); // Default constructor.
   int loadPlanes();
   void printPlanes();
   void listByMake();
   bool addAPlane();
   bool removeAPlane();
   void writePlanes();
};
```

Version C:

- List all the airplanes, one per line, using line numbers.
- Add a new airplane to the database in sorted order.
- Quit.

Version B:

- List all the airplanes, one per line, using line numbers.
- Add a new airplane to the database in sorted order.
- Remove an airplane by index (new for B version).
- Quit.

Version A:

- List all the airplanes, one per line, using line numbers.
- Add a new airplane to the database in sorted order.
- Remove an airplane by index.
- Search for an airplane by name (new for A version).
- List all airplanes by make (new for A version).
- Quit.

NOTE: There are different error checking requirements when adding a new airplane or removing an airplane for each one of the grade versions above:

- C version – check to make sure all number values are positive.
- B version – check to make sure all number values are positive and check for proper index to remove (not out of bounds). Also:
  - o Make sure the maximum fuel is less than 150 gallons.
  - o Make sure the empty weight is less than 3000 pounds.
  - o Make sure the engine horsepower is less than 400.
  - o Make sure the range is less than 2000.
- A version – All the error checks for version B and check for input failure (don't allow the program to crash or go into an infinite loop when letters are typed for input to numbers).

The file format lists all the information for one airplane on one line, and includes:

1. model (name)	2. make (manufacturer)
3. fuel capacity	4. empty weight

5. engine horsepower	6. range (nautical miles)
7. cruise speed (knots)	

Make sure that there is a newline (return) at the end of every line, including the last line. See the assignment 1 document for handling the newline at the end of the file.

Along with this document there will be an example text file that contains the following airplanes in this format:

```
360;Lancair;43.00;1090;180;990;208
Skyhawk 172;Cessna;53.00;1663;180;515;123
K35 Bonanza;Beechcraft;70.00;1832;250;534;168
RangeMaster H;Navion;40.00;1945;330;1381;160
Tomahawk;Piper;30.00;1128;112;383;107
M20R Ovation;Mooney;89.00;2205;280;969;189
C23 Sundowner;Beechcraft;57.00;1494;180;564;115
RV-12;Vans Aircraft;20.00;750;100;451;119
TB-21 GT Trinidad;Socata;88.00;1911;250;1025;168
RV-9;Vans Aircraft;36.00;1057;160;616;163
152;Cessna;26.00;1081;110;414;106
Tiger;Grumman;51.00;1360;180;529;139
Super Cub;Piper;36.00;845;125;449;96
```

The sample run / sample output is the same as the first assignment, so it has been removed from this version. Open the first assignment document to see the sample output.

Academic Integrity

You may NOT, under any circumstances, begin a programming assignment by looking for completed code on StackOverflow or Chegg or any such website, which you can't claim as your own. Please check out the Student Code of Conduct at PCC.

If you start with code from another source and just change the variable names or other content to make it look original, you will receive a zero on the assignment. I may ask you to explain your assignment verbally. If you cannot satisfactorily explain what your code does and why you wrote it in a particular way, then you should not expect to receive credit. Do not share your code with any other students and do not post it on a public website.

Programming Requirements

- Now that we have two classes for this assignment, we need a few more files. Just like assignment 1, you must have a main.cpp file and a plane.h file, but we also need an implementation file called plane.cpp. Place the Airplane class definition in the plane.h file and the member function definitions in the plane.cpp file. We need two additional files for the fleet class as well: fleet.h and fleet.cpp. Place the Fleet class definition in the fleet.h file and the member function definitions in the fleet.cpp file. To help me streamline my grading, please use these standard names. All member function definitions must be part of the class implementation file for both classes. You may also have `tools.h` and `tools.cpp` or `functions.h` and `functions.cpp`, but these files should only contain code for stand-alone functions (such as `welcome()`) which are not part of a class. You can also add a `main.h` file if you want. Please don't use any other names for your code and header files, and don't change the capitalization. So here is the list of files that may be part of your assignment:

Airplane class files:	plane.h	plane.cpp
Fleet class files:	fleet.h	fleet.cpp
Main function files:	main.cpp	main.h (optional)
Tools or functions files:	tools.h or functions.h	tools.cpp or functions.cpp

- Just like assignment 1, Don't use the string library or any of the STL containers such as vectors.
- Encapsulate your array of Airplane objects in the Fleet class. Do not use any global variables.

- Just like assignment 1, you must create several functions, but they must be part of the Fleet class, and therefore, in the Fleet header and implementation files. There are at least four member functions that you must create for the C version, including: `loadPlanes()`, `writePlanes()`, `addAPlane()`, and `listPlanes()`. For the B version, you must add `remove()`, and for the A version, `listByMake()` and `search()`. Since these functions are member functions, they will have the class name and the scope resolution operator in the function header. **Notice that because these functions use the member properties, they don't have arguments:**
  `bool Fleet::addAPlane();`
  If you add other stand-alone functions, such as `welcome()` place them in a tools or functions header and implementation file. You may change the name of these functions, such as `removeByIndex()` instead of just `remove()`.
- Just like assignment 1, make sure you check for index out of bounds when adding to the array. Make sure you check for index out of bounds before adding new data to the array.
- Notice that the `insert()` function is in the private section of the Fleet class. Only the `addAPlane()` and `loadPlanes()` functions will be calling it so it should be private (not available to client code).
- Check to make sure that the input file is open before reading from it, using `ifstream.is_open()`. Ask the user to try a different file name if the file does not open or have them type "quit" to quit the program at that point (if they don't have a collection file).
- Make sure that you check for negative numbers and bad data when reading number data from the user.
- For version C, make sure the number values are positive (greater than zero).
- For version B, check the values for version C, but also make sure the values are less than the values spelled out in Tasks - B version. For removal in version B, make sure that the index is not out of bounds.
- For version A, do all the error checking for version B but also make sure your program doesn't crash or go into an infinite loop for input failure.
- Use the file guards for your header files: `#ifndef`, `#define` and `#endif`. These are the preprocessor directives that should be placed around the contents of a header file.
- The Fleet class will make use of the Airplane class, so use `#include "plane.h"` in the fleet.h file. The main function will make use of the Fleet class, so use `"#include "fleet.h"` in main.cpp or main.h (if you have a main.h file). That way, the header file for Airplane will be included in both the fleet implementation and in the main function file.
- You must represent a single airplane using an airplane class object. Represent a fleet of airplanes using an array of airplane objects inside of a fleet class object. You may change the name of the classes and member variables, such as, instead of `class Airplane`, you could use `class planeInfo`, and perhaps just `weight` instead of `emptyWeight`.

Programming Strategies

- You may assume that the input file is formatted correctly, so no error checking is necessary when reading from the file. See the assignment 1 document for data verification strategies for data from the keyboard.
- You can use the example database file, or you can create your own input file using your favorite text editor on the Linux system: vim, emacs, nano, etc. If you use a Windows or Mac program like Notepad++ or TextEdit, make sure that the last character in the file is a newline (return character), and make sure you save the input file in text file format with Unix line termination, rather than rich text, Word, or some other format. Use the <fstream> library to load the file. You must use C++-style input and output objects and manipulators; do not use any C-style functions for input and/or output, such as fprintf, fscanf, etc.
- When doing list output, you may use setw() and left/right from the <iomanip> library to align the data into columns, but it is not required. Use fixed, showpoint, and setprecision(2) for floating point number output.

- Do not assume that the input file contains a certain number of airplanes. Read from the file using an EOF-controlled loop (loop to the end-of-file marker. See the assignment 1 document for more details).
- Do not use extraction (>>) with cin or ifstream to read data into any c-strings, because this is a potential security issue called buffer overflow. Extraction also stops at whitespace, so you won't be able to take multiple-word input. Use `cin.getline()` or `inFile.getline()` instead. For more information, check out the cplusplus.com page.
- Notice that there is a prototype for the default constructor in both classes. Default constructors are invoked when a new object is created. Anything about an object that should be set up automatically should happen in the default constructor. For example, when a new Fleet object is created, the array is empty, so the count property should be set to zero.
- Notice that none of the member functions (methods) of the Fleet class have arguments. That's because all the arguments that were used in assignment 1 have been encapsulated. Let's compare one of the functions for assignment 1 to the prototypes in the Fleet class – `loadPlanes()` for example:
  - Assignment 1: `int loadPlanes(planeInfo planes[], ifstream & planeFile);`
  - Assignment 2: `int loadPlanes();`

  Why doesn't the version from assignment 2 need the arguments? The answer is because all the arguments – the array of Airplanes, the input file, and/or the file name – have all been incapsulated and are members of the class. So, they can be directly accessed by the `loadPlanes()` function.
- You can encapsulate as many properties as you need for the classes, including the file name, input file, and possibly the output file for the Fleet class. I've included the file name and input file as part of the Fleet class in the textbox above. If I forgot some properties or methods that should be part of either class, go ahead and add them. If you want to add something about individual airplanes, make it part of the Airplane class. If you want to add something about the collection, add it to the Fleet class. You can ask for the file name and open it in the constructor or in the `loadPlanes()` function. Don't forget to close the file at the appropriate time. Use the same name for an ofstream file for `writePlanes()`.

List of Items to Complete:

- Complete zyBooks **zylab 17.11, Artwork Label** and **zylab 18.4, Car Value**
- Open the Algorithm Design Document or the UML based design document, make a copy, and follow the steps to create your algorithm.
- **Be sure to include your screenshot of the zyBooks completion in the algorithm document. The screenshot must be present for your assignment to be graded.**
- Create your source code files, build and test your program.
- Code may be written using any development environment but must use Standard C++.
- PSU-bound students are strongly encouraged to do all development on the PCC Linux server.
- Please open and compare your work with the grading rubric before submitting.
- Remember to follow all C++ style guidelines.
- After you complete the code and the algorithm document, compress the files together along with your text file, and submit the zip file to the D2L assignment 2 submission area.
- You may express your algorithm as pseudocode or a flowchart. Please note that your pseudocode must follow the syntax requirements shown in the document - **you may not use C++ syntax as a substitute for pseudocode**.

Additional Support

- Post a question for the instructor in the Ask Questions! area of the Course Lobby.
- Set up a Zoom session with the instructor or the tutor.
- Come to class on M/W from 10:00 -1:00, or office hour after class, 1:00 - 2:00. Sylvania TCB 311.
- Go to the Discord server and post a question there.