CS162 Assignment 3: Dynamic Memory Airplane Information Database

For our airplane database program, we are going to take steps to make it more memory efficient. Consider the Airplane class from assignment 2 and the make and model c-string members. The constant STR_SIZE is at least 100, so that make and model can contain a relatively large amount of characters, just in case. However, one of the model names is 152, which is only 4 characters (3 characters and 1 null terminator). So, allocating 100 characters for such a small name is wasting 96 bytes. There is also a potential waste of memory – and a limitation - when creating an array of Airplanes in the Fleet class. The constant I chose for the size of the array is 30. If we have a list of only 10 airplanes, then we will have created memory for an extra 20 airplanes that won't be used. Conversely if we have a list of 40 airplanes, then we won't have enough room to load the last 10 from the file into memory.

To remedy this situation, we will be allocating memory for c-strings and the array of Airplanes dynamically. That means that we need char pointers for the c-strings and an Airplane pointer for the Airplane array. For the c-strings, the code will count the number of characters necessary for the make and model and then only allocate that many characters, plus one for the null-terminator. To store the Airplanes, the code will allocate a small Airplane array and then "grow" if more room is needed. See below for changes to the classes. The Airplane class now has char pointers for make and model, and the Fleet class has an Airplane pointer for a dynamic array.

Because the two classes (Airplane and Fleet) will have dynamic members, this assignment requires the "rule of 3" functions: the **destructor** (so there won't be memory leaks), the **copy constructor**, and the **overloaded assignment operator** (so there won't be shallow copies). There is also a new function called growArray() in the Fleet class, so that new memory can be allocated if the array of Airplanes becomes full.

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, insert the airplanes in sorted order by model name, close the input file and only deal with data in memory, and then write the data back to the output file at program termination.

Requirements:

- Continue to implement the encapsulation and data hiding requirements from assignment 2. The Fleet class must encapsulate the array of Airplane class objects, and the array must be private. 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 implement dynamic memory for make and model using a char pointer, and the array of Airplanes in the Fleet class must be dynamic. For the Airplane class, you must implement the destructor, copy constructor, and overloaded assignment operator. You must implement the destructor and the growArray() function for the Fleet class. You may implement the copy constructor and the assignment operator for Fleet or set them to delete. See below in the Strategies section for more details.
- You must not use <string> or any container classes such as <vector>. **Do not use the <sstream> class, as that class uses string objects**.
- You must use functions and have multiple code files and header files.

After completing this assignment, you will be able to:
- Write code using classes and objects that use dynamic memory.
- Create and use dynamic c-strings and dynamic arrays.
- Implement the default constructor, copy constructor, destructor, and overloaded assignment operator for a user-defined class.
- Avoid memory errors such as segmentation faults, memory leaks and dangling pointers.
- Implement and use accessors and mutators for a class.
- Organize source code in multiple header and implementation files.
- Read from input data files.

- You must follow the coding style listed in the C++ style guidelines.
- Use valgrind or some other memory leak checker to make sure that your program does not have any memory leaks. See the strategies section below for details.

Tasks (Same as Assignment 2):

- Complete the zylabs from chapters 19 and 20. 19.16, Car value (Pointer Objects) , 20.10, List of Integers and 20.11, Video Class. For chapters 19 and 20, 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 model. 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:

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 model (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:
    - Make sure the maximum fuel is less than 150 gallons.
    - Make sure the empty weight is less than 3000 pounds.
    - Make sure the engine horsepower is less than 400.

  - Make sure the range is less than 2000.
  - Make sure the maximum cruise is less than 250 knots.
- 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
```

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

  public:
    Fleet(); // Default constructor.
    ~Fleet(); // Destructor.
    Fleet(const Fleet &) = delete;
    Fleet & operator=(const Fleet &) = delete;
    int loadPlanes();
    void printPlanes();
    void listByMake();
    void search();
    bool addAPlane();
    bool removeAPlane();
    void writePlanes();
    void growArray();
};
```

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

- Just like assignment 2, you must have a main.cpp file, a plane.h and plane.cpp file, and a fleet.h and fleet.cpp file. All member function definitions must

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

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.

- Just like assignment 1 and 2, 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.
- Unlike assignment 2, you won't be checking the size of the Airplane array against a constant. This time you will be checking against a member variable, perhaps called capacity, that will be set in the constructor. If the size of the array is greater than or equal to capacity, then grow the array by calling the growArray() function.
- All of the remaining requirements from assignment 2 are also required for this assignment. Namely:
  1. make sure the file is open, ifstream.is_open()
  2. Check for negative values and bad data, depending on the grade version (C, B, or A).
  3. Use the file guards in your header files - `#ifndef, #define` and `#endif`
  4. Use #include to include the fleet.h and plane.h header files where they are needed.
  5. Use the Airplane class to represent airplane data and the Fleet class to represent a collection.

Programming Strategies

Now that we have dynamic c-strings and a dynamic array, the constructor for Airplane will be responsible for allocating dynamic memory, or for setting the char pointers to nullptr. You can either allocate memory for something like "none," or you can just set the pointers equal to nullptr.
The constructor for Fleet will need to allocate an array for fleetAirplanes. Set the array capacity small, such as 3, so that it will be easy to test your `growArray()` function:

```
count = 0;

capacity = 3;

fleetAirplanes = new Airplane[capacity];
```

We will need the "rule of 3" as described in the Zybook for the Airplane class. The rule of 3 refers to the 3 member functions (at a minimum) that we must implement. The 3 functions are: the assignment operator (also called the copy assignment operator), the copy constructor, and the destructor. Those 3 functions are provided by the compiler automatically if we don't provide code for them. However, we can't rely on the versions of the copy constructor and assignment operator that are provided automatically anymore, because our classes have dynamically allocated properties. If we don't provide the overloaded code for them, then we will have a serious problem called "shallow copy." This is because our objects contain dynamic variables, which are simply pointers. Let's look at examples of how these functions are invoked:

```
Airplane a1, a2;

a1 = a2;          // invoke the assignment operator.

Airplane a3(a2); // invoke the copy constructor. a3 will be a copy of a2.
```

Suppose the model of a2 is set to the string "Skyhawk 172". The auto-generated copy constructor and assignment operator will make a copy of the pointer for model. Therefore a1.model and a2.model will be pointing at the same memory location. In effect, they will be sharing the memory for model, and so if the model for a1 is modified, it will modify the model for a2, or more likely, create errors for a2. To fix this problem, we need a deep copy. So, in the code for the overloaded copy constructor and assignment operator, we will first find out how many characters are in a2.model, allocate just the right amount of characters for a1.model, and then use: `strcpy(a1.model, a2.model);`

We will need a similar strategy for the setters for the c-strings. Suppose the user changes the model to "TB-21 GT Trinidad." This is longer than the string "Skyhawk 172," so we can't just overwrite the memory that contains "Skyhawk 172," because we haven't allocated enough memory and will have buffer overflow. So instead, we will have to delete the currently assigned memory and then allocate new memory for the new model name.

We need this same strategy for the growArray() function. Once we run out of room in the fleetAirplanes array, we will need to allocate a larger array. Once we create the new array, we will copy each Airplane from the old array to the new array, delete the old array, and then set fleetAirplanes to the new array.

The destructor is invoked when an object is no longer accessible, which is when an automatic object (on the stack) goes out of scope or when a dynamically allocated object (on the heap) is deleted. The only thing the destructors will need to do is call `delete` or `delete[]` for all of the dynamically allocated properties.

If all dynamic memory is deleted before program termination, then there can't be any memory leaks, but we need to make sure. Memory leaks are notoriously hard to find. So, we can use a tool called valgrind. Valgrind is a program on the Linux server that checks your program for memory leaks and other memory errors. Here is a sample run: `valgrind --leak-check=full ./programName`

The textbox on the left shows that valgrind found some memory leaks, and the textbox on the right shows no memory leaks:

```
LEAK SUMMARY:
definitely lost: 1,104 bytes in 3 blocks
indirectly lost: 1,266 bytes in 81 blocks
possibly lost: 0 bytes in 0 blocks
still reachable: 0 bytes in 0 blocks
suppressed: 0 bytes in 0 blocks
```

```
HEAP SUMMARY:
in use at exit: 0 bytes in 0 blocks
total heap usage: 282 allocs, 282 frees, 97,634 bytes allocated
All heap blocks were freed -- no leaks are possible
```

You may notice that there are prototypes for the copy constructor and overloaded assignment operator in the Fleet class, but both are set to delete. If you wish to implement them, then remove the equal sign and delete keyword from the prototype. We will only need one Fleet object, so we won't need a copy, so we won't write the code for those two functions. Remember though that if we don't provide these two functions, then the compiler will, so if the client tries to make a copy of a Fleet object, it will produce a shallow copy. To stop the client from making a copy, we can set those two functions to delete, and they will get a compilation error.

List of Items to Complete:

- Complete zylabs 19.16, Car value (Pointer Objects) , 20.10, List of Integers and 20.11, Video Class. 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, you may upload your files individually or as a compressed zip file. Please upload all the assignment files including the data text file.
- 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.