ECS 40 **Program #3 (30 pts, < one hour)** **Spring 2016**

Due: Wednesday April 20, 11:59 PM Executable name: airline.out Handin to cs40a/p3

Filenames: authors.csv, main.cpp, utilities.cpp, utilities.h, plane.cpp, plane.h, flight.cpp, flight.h, Makefile.

You will convert the program #1 code to C++. You may wish to use my p1 source code (available in ~ssdavis/40/p1/SeansSrc Thursday morning) as a starting point for your p3. The following steps are based on my own program #1 source code. If you use your own p1 code, then you may need to look at my program source code to better understand the steps.

Specifications:

1. The typedef struct Plane must be replaced with a class Plane.
   1.1. All functions that deal with any data of the Plane class must be members of the Plane class.
   1.2. The constructor of Plane should read the information from the file.
   1.3. Member functions that are called from functions outside the Plane class should be public.
   1.4. Member functions that are only called from member functions of the Plane class should be private.
2. The typedef struct Flight must be replaced with a class Flight.
   2.1. All functions that deal with any data of the Flight class must be members of the Flight class.
   2.2. Member functions that are called from functions outside the Flight class should be public.
   2.3. Member functions that are only called from member functions of the Flight class should be private.
3. All functions dealing with the array of Flights should be in main.cpp.
4. All calls to functions in stdio.h should be replaced with calls from <iostream>, <fstream>, and <iomanip>
   4.1. stdio.h may not be included in any file.
   4.2. The reservations.txt file object must be named inf, and the reservations2.txt file object must be named outf.
5. All dynamic allocation should use "new," and all deallocation should use "delete.'
   5.1. The destructors for the Plane and Flight class should take care of all deallocation for their respective classes.
   5.2. The Flight array should be deallocated in main.cpp.
   5.3. stdlib.h may not be included in any file.
6. Use const for parameters, return types, and member functions wherever appropriate.
   6.1. Parameters passed by reference (either pointers or reference variables) that are not changed in a function must be const.
   6.2. Member functions that do not change any of the class' data, e.g. writePlane() should be made const functions.
   6.3. The released design checker will check for some consts. The actual design checker used for grading will check for additional consts, so do not rely on the released design checker to determine if something should be const!
7. Code must be submitted by exactly one member of each team. Double submissions, or errors in the authors.csv format will result in the team losing five points. Your handin command line will be:
   handin cs40a p3 authors.csv main.cpp plane.h plane.cpp flight.h flight.cpp utilities.h utilities.cpp Makefile

Suggested steps for development

1. (<10 minutes) Convert the Plane struct and the functions involving a Plane* to the Plane class. This assumes that all functions in plane.cpp have a Plane* as a parameter. If some do not, they will still need to be Plane methods to be able to access the private data of the Plane class.
   1.1. plane.h
      1.1.1. Make the typedef struct a Plane class with the variables in the private (default) section.
      1.1.2. Move the prototypes of all the functions that have a Plane* as a parameter, and are called from other files into a public section of the class declaration.
      1.1.3. Move the prototypes of all other functions into a private section of the class declaration.
      1.1.4. Remove the Plane* parameter from all the functions in the Plane class declaration.
      1.1.5. Move the prototypes of any other functions, to the private section of the Plane class declaration.
      1.1.6. Rename the function that frees the Plane's memory to be the destructor for the Plane class.
   1.2. plane.cpp. Make every function that has a Plane* a member of the Plane class.
      1.2.1. Use a global search and replace to remove the Plane* parameter from all the functions.
      1.2.2. Add "Plane::" in front of every function name.

1.2.3. Use a global search and replace to remove all of the "plane->" in the file, since the plane data is now implicitly available in the class methods.
1.2.4. Rename the function that reads the information from the file to become the constructor for the Plane class that takes the ifstream as its only parameter.
1.2.5. Rename the function that frees the Plane's memory to be the destructor for the Plane class.
1.2.6. Rename the function that reads the information from the file to become the constructor for the Plane class.
1.3. flight.cpp
1.3.1. Remove the flight->plane parameters used with Plane methods, and place "flight->plane->" in front of each such function call.
1.3.2. Change the dynamic allocation of a Plane to call that class' constructor.
1.3.3. Delete the call to the Plane function that freed the Plane's memory. The Plane destructor will be called automatically by the free(flight->plane) call.
1.4. Your program should now compile without warnings, and work perfectly.

2. (<10 minutes) Convert the Flight struct and the functions involving a Flight* to the Flight class. This assumes that all functions in flight.cpp have a Flight* parameter. If some do not, they will still need to be Flight methods to be able to access the private data of the Flight class. Follow the same pattern as you used for the Plane conversion for the flight.h, flight.cpp and main.cpp files. You should change the function that frees the plane to be the Flight destructor. In main, you would use "flights[i]." to call Flight methods. Again, your program should compile without warnings, and run perfectly when you are done.
3. (5 minutes) Change all calls using malloc to using new, and all frees to deletes. Remember to replace "free" with "delete []" when freeing an array, but only "delete" when freeing a single Plane. delete is not a function call, so remove the function parentheses used with "free". Remove the #include of <stdlib.h> from your program. Again, your program should run perfectly when you are done.
4. (30 minutes) Replace the all of the stdio.h functions with methods and overloaded operators from <iostream>, and <fstream>. Your program should run without warnings at the end of each of the six subparts.
4.1. #include the two headers at the top of main.cpp, plane.cpp, and flight.cpp. #include <fstream> at the top of the header files that mention FILE*. Add "using namespace std;" to each of these files just below the #include lists to avoid having to use "std::" with the fstream and iostream methods and objects.
4.2. Replace the reading FILE *fp with an ifstream object with the name inf.
4.2.1. Replace the fopen() with a declarations of the ifstream object.
4.2.2. Replace fclose() with a call to close().
4.2.3. Use &ifstream in place of FILE* that was passing a reading file pointer.
4.2.4. Replace fscanf() calls with statements that use "inf >> ".
4.2.5. inf.get() is a convenient way to read a single character, including space characters.
4.2.6. Replace fgets() with inf.getline(). Since getline does not copy the '\n' you no longer need to eliminate that from the string. (I have eliminated the messed up end of line characters from reservations.txt.)
4.2.7. The ignore() method of iostream (and thus ifstream), is a convenient way to eat up trailing '\n' in the file and the keyboard buffer.
4.3. Replace the writing FILE *fp with an ofstream object with the name outf.
4.3.1. Replace the fopen() with a declarations of the ofstream object.
4.3.2. Replace fclose() with a call to close().
4.3.3. Use &ofstream in place of FILE* that was passing a writing file pointer.
4.3.4. Replace fprintf() with statements that use "outf << ".
4.4. Replace scanf() with cin >>, getchar() with cin.get(), and fgets(…,stdin) with cin.getline(…).
4.5. Replace printf() with cout <<. You will need to #include <iomanip> to use setw().
4.6. Remove #include<stdio.h> from the top of all files.

5. (< 5 minutes) Add const where possible.
5.1. Make appropriate parameters const, by looking at every function, and determining if any variable passed by reference is not changed in the function.
5.2. Make appropriate member functions const (by putting "const" after their closing parameter parentheses). A const function does not change any data of the member object that called it.