CS 162: Computer Science II

Algorithm Design Document

Make a copy before you begin (File -> Make a copy). Add the Assignment # above and complete the sections below BEFORE you begin to code. The sections will expand as you type. When you are finished, download this document as a PDF (File -> Download -> PDF) and submit to D2L.

This document contains an interactive checklist. To mark an item as complete, click on the box (the entire list will be highlighted), then right click (the clicked box will only be highlighted), and choose the checkmark.

Planning your program before you start coding is part of the development process. In this document you will:

❑ Paste a screenshot of your zyBooks Challenge and Participation %
❑ Paste a screenshot of your assigned zyLabs completion
❑ Write a detailed description of your program, at least two complete sentences
❑ If applicable, design a sample run with test input and output
❑ Identify the program inputs and their data types
❑ Identify the program outputs and their data types
❑ Identify any calculations or formulas needed
❑ Write the algorithmic steps as pseudocode

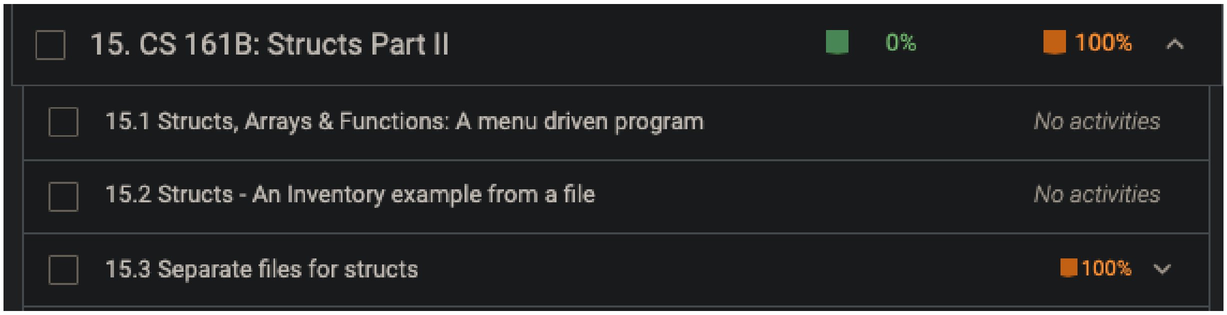
1. zyBooks

Add your zyBooks screenshots for the % and assigned zyLabs completions below. Required percentages: all **assigned** zyLabs, Challenge Activity with at least 70%, and Participation Activity with at least 80%.

Challenge and Participation % screenshot:

https://replit.com/@ginaferguson/CS162-A01-Structs-Planes#main.cpp

https://learn.zybooks.com/zybook/PCCCS162Summer24/chapter/15/section/1



□ 15. CS 161B: Structs Part II 🟩 0% 🟧 100%

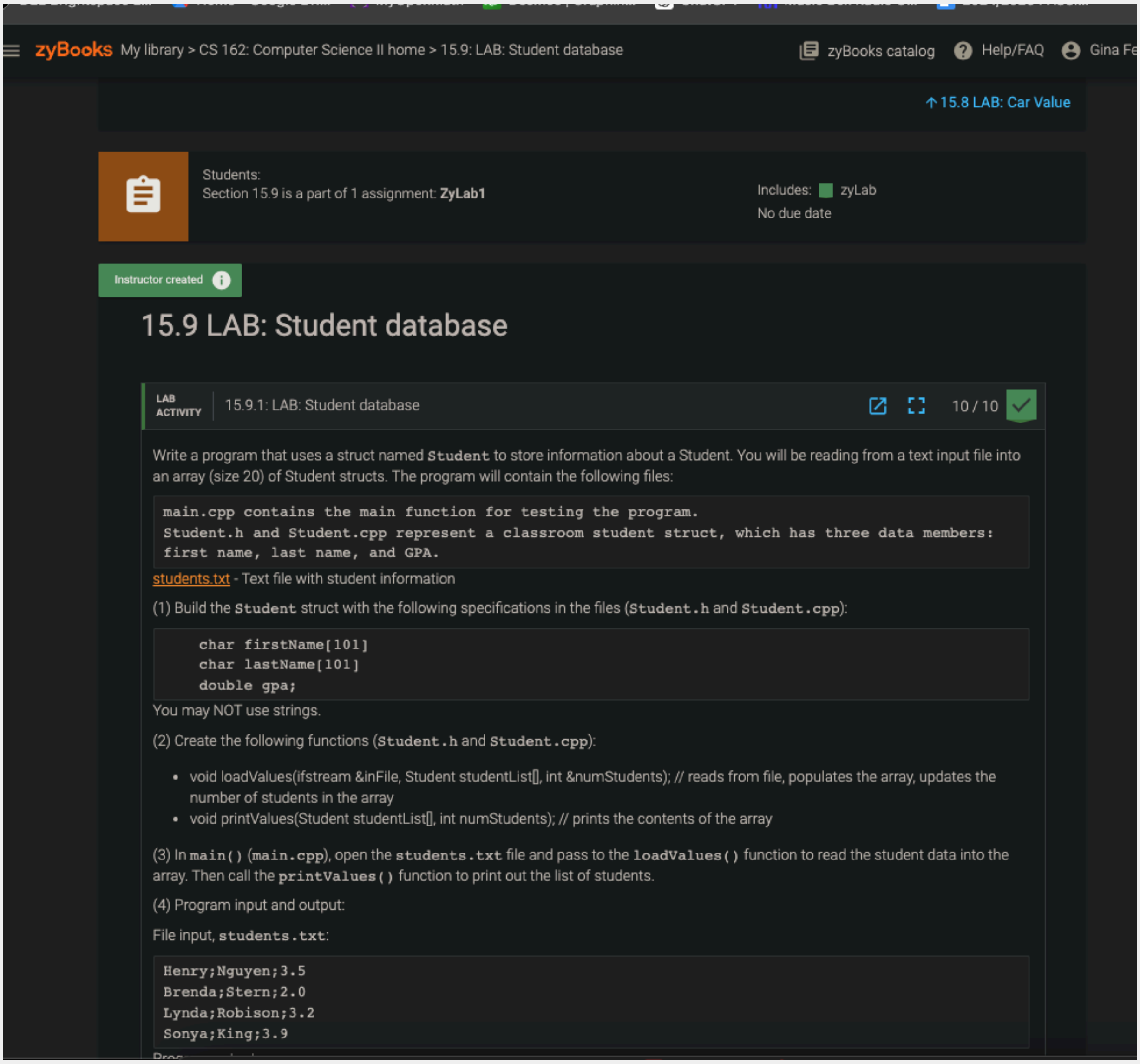
□ 15.1 Structs, Arrays & Functions: A menu driven program *No activities*

□ 15.2 Structs - An Inventory example from a file *No activities*

□ 15.3 Separate files for structs 🟧 100%

Assigned zyLabs completion screenshot:

So, when I go to my Library for Zybooks, it says I have 0% on the Zylabs, which is NOT true. The one required is finished, and I will post proof below. I also noticed that it let me download my files after I was done, so I can attach those on the Assignment 1 submission page for further proof.



zyBooks My library > CS 162: Computer Science II home > 15.9: LAB: Student database
zyBooks catalog
Help/FAQ
Gina Fe
↑ 15.8 LAB: Car Value

Students:
Section 15.9 is a part of 1 assignment: **ZyLab1**

Includes: 🟩 zyLab
No due date

15.9 LAB: Student database

15.9.1: LAB: Student database 10 / 10 ✓

Write a program that uses a struct named `Student` to store information about a Student. You will be reading from a text input file into an array (size 20) of Student structs. The program will contain the following files:

```
main.cpp contains the main function for testing the program.
Student.h and Student.cpp represent a classroom student struct, which has three data members:
first name, last name, and GPA.
```

students.txt - Text file with student information

(1) Build the `Student` struct with the following specifications in the files (`Student.h` and `Student.cpp`):

```
    char firstName[101]
    char lastName[101]
    double gpa;
```

You may NOT use strings.

(2) Create the following functions (`Student.h` and `Student.cpp`):

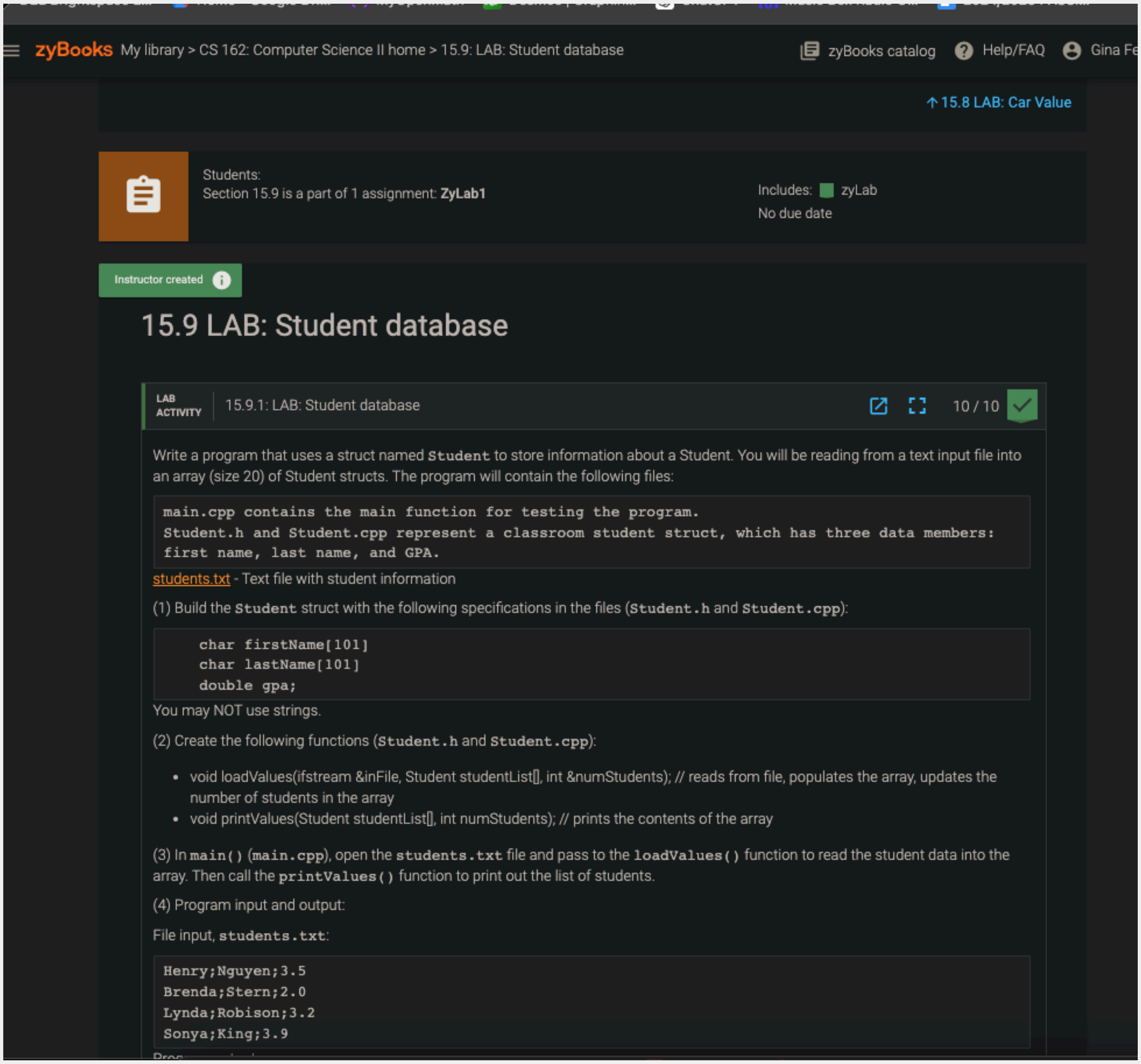
- void loadValues(ifstream &inFile, Student studentList[], int &numStudents); // reads from file, populates the array, updates the number of students in the array
- void printValues(Student studentList[], int numStudents); // prints the contents of the array

(3) In `main()` (`main.cpp`), open the `students.txt` file and pass to the `loadValues()` function to read the student data into the array. Then call the `printValues()` function to print out the list of students.

(4) Program input and output:

File input, `students.txt`:

```
Henry;Nguyen;3.5
Brenda;Stern;2.0
Lynda;Robison;3.2
Sonya;King;3.9
```


zyBooks My library > CS 162: Computer Science II home > 15.9: LAB: Student database
zyBooks catalog
Help/FAQ
Gina Fe
↑ 15.8 LAB: Car Value

Students:
Section 15.9 is a part of 1 assignment: **ZyLab1**

Includes: 🟩 zyLab
No due date

15.9 LAB: Student database

LAB ACTIVITY
15.9.1: LAB: Student database
10 / 10 ✓

Write a program that uses a struct named `Student` to store information about a Student. You will be reading from a text input file into an array (size 20) of Student structs. The program will contain the following files:

```
main.cpp contains the main function for testing the program.
Student.h and Student.cpp represent a classroom student struct, which has three data members:
first name, last name, and GPA.
```

students.txt - Text file with student information

(1) Build the `Student` struct with the following specifications in the files (`Student.h` and `Student.cpp`):

```
    char firstName[101]
    char lastName[101]
    double gpa;
```

You may NOT use strings.

(2) Create the following functions (`Student.h` and `Student.cpp`):

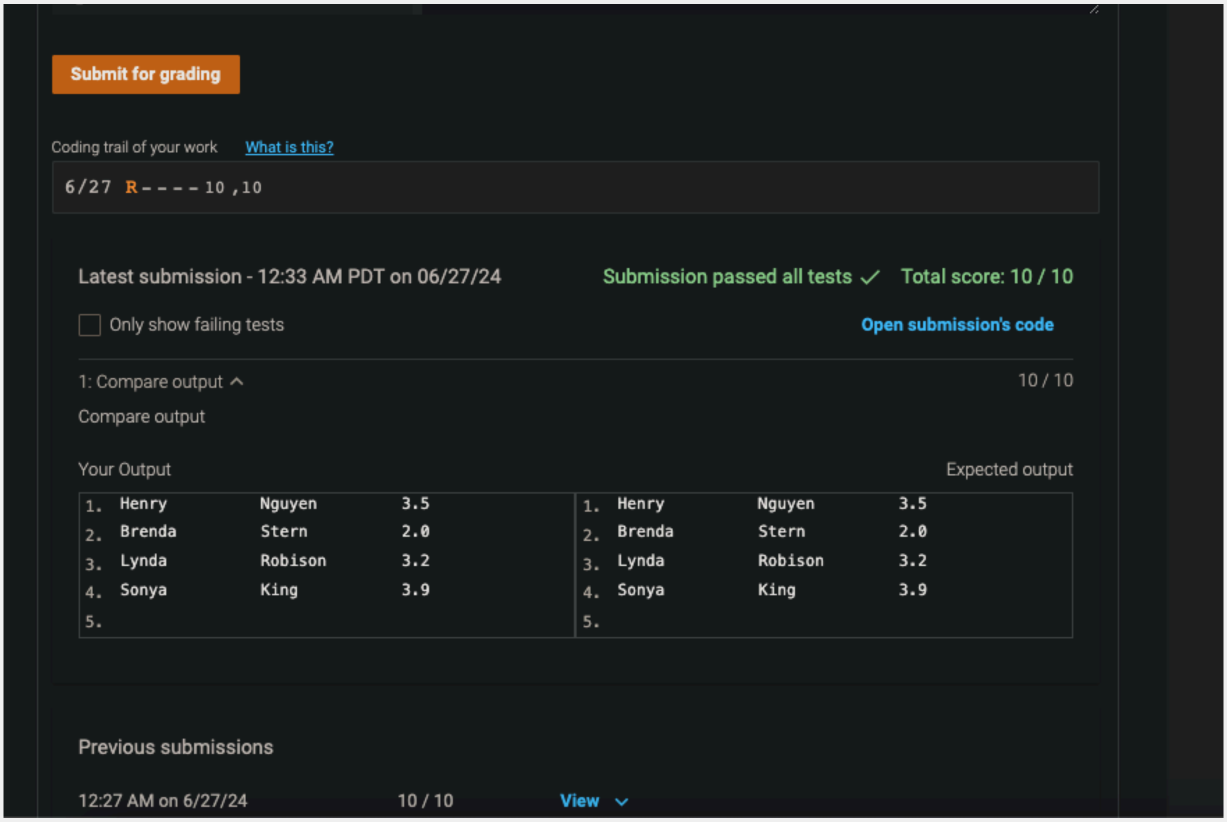
- void loadValues(ifstream &inFile, Student studentList[], int &numStudents); // reads from file, populates the array, updates the number of students in the array
- void printValues(Student studentList[], int numStudents); // prints the contents of the array

(3) In `main()` (`main.cpp`), open the `students.txt` file and pass to the `loadValues()` function to read the student data into the array. Then call the `printValues()` function to print out the list of students.

(4) Program input and output:

File input, `students.txt`:

```
Henry;Nguyen;3.5
Brenda;Stern;2.0
Lynda;Robison;3.2
Sonya;King;3.9
```



Coding trail of your work

```
6/27 R----10,10
```

Latest submission - 12:33 AM PDT on 06/27/24 Submission passed all tests ✓ Total score: 10 / 10

□ Only show failing tests

1: Compare output 10 / 10

Compare output

Your Output

```
1. Henry      Nguyen     3.5
2. Brenda     Stern      2.0
3. Lynda      Robison    3.2
4. Sonya      King       3.9
5.
```

Expected output

```
1. Henry      Nguyen     3.5
2. Brenda     Stern      2.0
3. Lynda      Robison    3.2
4. Sonya      King       3.9
5.
```

Previous submissions

12:27 AM on 6/27/24 10 / 10

2. Program Description

In the box below, describe the purpose of the program. You must include a detailed description with at least two complete sentences.

Program description:
This program will read from a list of airplanes in a .txt file and create a menu for the user to be able to view, search, write new, remove, and export the airplane data into an output .txt file. The program will allow the user to view the airplane's make, model, maximum fuel needed, weight, horsepower, maximum range, and cruising speed.

3. Sample Run

If you are designing your own program, you will start with a sample run. **Imagine** a user is running your program - what will they see? What inputs do you expect, and what will be the outputs from the given inputs? Choose test data you will use to test your program. Calculate

and show the expected outputs. Use the sample run to test your program.
Do not simply copy the sample run from the assignment instructions!

Sample run:

```
(User Input in Bold):

Welcome to the airplane collection program!

What is the name of the airplane collection file? notAFile.txt

*** The file notAFile.txt did not open. Type 'Q' to quit, or try again now: planes.txt

13 planes were loaded from the file.


     Model               Make           Fuel Capacity   Empty Weight     Horsepower          Range   Cruise Speed

-----------------------------------------------------------------------------------------------------------------
 1.  152                 Cessna                 26.00           1081            110            414            106
 2.  360                 Lancair                43.00           1090            180            990            208
 3.  C23 Sundowner       Beechcraft             57.00           1494            180            564            115
 4.  K35 Bonanza         Beechcraft             70.00           1832            250            534            168
 5.  M20R Ovation        Mooney                 89.00           2205            280            969            189
 6.  RV-12               Vans Aircraft          20.00            750            100            451            119
 7.  RV-9                Vans Aircraft          36.00           1057            160            616            163
 8.  RangeMaster H       Navion                 40.00           1945            330           1381            160
 9.  Skyhawk 172         Cessna                 53.00           1663            180            515            123
10.  Super Cub           Piper                  36.00            845            125            449             96
11.  TB-21 GT Trinidad   Socata                 88.00           1911            250           1025            168
12.  Tiger               Grumman                51.00           1360            180            529            139
13.  Tomahawk            Piper                  30.00           1128            112            383            107


What would you like to do?
(A)dd a plane,
(L)ist all planes,
(R)emove a plane by index
(Q)uit?
A

What is the model (name) of the airplane? Malibu Mirage
What is the make (manufacturer) of the airplane? Piper
What is the fuel capacity in gallons? One hundred and twenty
Invalid input. Please enter a valid fuel capacity between 1.00 and 150.00: 120.00
What is the empty weight (in pounds)? 20500
The weight must be a whole number between 1 and 3000 pounds: 2435
What is the horsepower of the engine? 550
Invalid input. Please enter a valid horsepower between 1 and 400: 350
What is the range? 1342
What is the cruise speed? 212
```

```
Successfully added Malibu Mirage to the database.

What would you like to do?
(A)dd a plane,
(L)ist all planes,
(R)emove a plane by index
(Q)uit?
L

    Model             Make     Fuel Capacity   Empty Weight   Horsepower   Range     Cruise speed
--------------------------------------------------------------------------------------------
1 . 152               Cessna         26.00        1081        110         414         106
2 . 360               Lancair        43.00        1090        180         990         208
3 . C23 Sundowner     Beechcraft     57.00        1494        180         564         115
4 . K35 Bonanza       Beechcraft     70.00        1832        250         534         168
5 . M20R Ovation      Mooney         89.00        2205        280         969         189
6 . Malibu Mirage     Piper         120.00        2435        350        1342         212
7 . RV-12             Vans Aircraft  20.00         750        100         451         119
8 . RV-9              Vans Aircraft  36.00        1057        160         616         163
9 . RangeMaster H     Navion         40.00        1945        330        1381         160
10. Skyhawk 172       Cessna         53.00        1663        180         515         123
11. Super Cub         Piper          36.00         845        125         449          96
12. TB-21 GT Trinidad Socata         88.00        1911        250        1025         168
13. Tiger             Grumman        51.00        1360        180         529         139
14. Tomahawk          Piper          30.00        1128        112         383         107

What would you like to do?
(A)dd a plane,
(L)ist all planes,
(R)emove a plane by index
(Q)uit?
R
Enter the index of the plane to remove: 0
Invalid Index. Please type an index between 1 and 14: 15
Invalid Index. Please type an index between 1 and 14: four
Invalid Index. Please type an index between 1 and 14: 9

Successfully removed plane at index 9.

What would you like to do?
(A)dd a plane,
(L)ist all planes,
(R)emove a plane by index
(Q)uit?
L

    Model             Make     Fuel Capacity   Empty Weight   Horsepower   Range     Cruise speed
--------------------------------------------------------------------------------------------
1 . 152               Cessna         26.00        1081        110         414         106
2 . 360               Lancair        43.00        1090        180         990         208
3 . C23 Sundowner     Beechcraft     57.00        1494        180         564         115
```

```
4 .  K35 Bonanza        Beechcraft       70.00          1832          250          534          168
5 .  M20R Ovation       Mooney           89.00          2205          280          969          189
6 .  Malibu Mirage      Piper           120.00          2435          350         1342          212
7 .  RV-12              Vans Aircraft    20.00           750          100          451          119
8 .  RV-9               Vans Aircraft    36.00          1057          160          616          163
9 .  Skyhawk 172        Cessna           53.00          1663          180          515          123
10.  Super Cub          Piper            36.00           845          125          449           96
11.  TB-21 GT Trinidad  Socata           88.00          1911          250         1025          168
12.  Tiger              Grumman          51.00          1360          180          529          139
13.  Tomahawk           Piper            30.00          1128          112          383          107

What would you like to do?
(A)dd a plane,
(L)ist all planes,
(R)emove a plane by index
(Q)uit?
Q

What is the name of the file to write to? out.txt

Database file updated. Terminating Program.
```

4. Algorithmic Design

Before you begin coding, **you must first plan out the logic** and think about what data you will use to test your program for correctness. All programmers plan before coding - this saves a lot of time and frustration! Use the steps below to identify the inputs and outputs, calculations, and steps needed to solve the problem.

Use the pseudocode syntax shown in the document, supplemented with English phrases if necessary. **Do not include any implementation details (e.g. source code file names, class or struct definitions, or language syntax)**. Do not include any C++ specific syntax or data types.

Algorithmic design:

a. Identify and list all of the user input variables and their data types. Include a variable name, data type, and description. Data types include string, integer, floating point, (single) character, and boolean. Data structures should be referenced by name, e.g. "array of integer" or "array of string".

```
1. fileName
     a. Data Type: string
     b. Description: Stores the name of the airplane collection
        file entered by the user.
```

choice

2. choice
    a. Data Type: character
    b. Description: Stores the user's menu choice (A, L, R, Q).
3. newPlane.model
    a. Data Type: string
    b. Description: Stores the model name of the airplane entered by the user.
4. newPlane.make
    a. Data Type: string
    b. Description: Stores the make (manufacturer) of the airplane entered by the user.
5. newPlane.maxFuel
    a. Data Type: floating point
    b. Description: Stores the fuel capacity in gallons of the airplane entered by the user.
6. newPlane.emptyWeight
    a. Data Type: integer
    b. Description: Stores the empty weight in pounds of the airplane entered by the user.
7. newPlane.engineHP
    a. Data Type: integer
    b. Description: Stores the horsepower of the airplane engine entered by the user.
8. newPlane.maxRange
    a. Data Type: integer
    b. Description: Stores the range in nautical miles of the airplane entered by the user.
9. newPlane.cruiseSpeed
    a. Data Type: integer
    b. Description: Stores the cruise speed in knots of the airplane entered by the user.
10. index
    a. Data Type: integer
    b. Description: Stores the index of the airplane to be removed, as entered by the user.

b. Identify and list all of the user output variables and their data types. Include a variable name, data type, and description. Data types include string, integer, floating point, (single) character, and boolean. Data structures should be referenced by name, e.g. “array of integer” or “array of string”.

1. planes

- Data Type: array of Airplane structs
- Description: Stores the collection of airplane structs, each containing details about a plane (model, make, maxFuel, emptyWeight, engineHP, maxRange, cruiseSpeed). This is the main data structure used to store and display the airplanes.

2. count
   - Data Type: integer
   - Description: Stores the number of airplanes currently loaded in the array. This is used to determine how many airplanes are displayed and managed within the program.

3. success
   - Data Type: boolean
   - Description: Indicates whether an operation (such as adding a plane) was successful. This is used to provide feedback to the user about the success or failure of their actions.

4. result
   - Data Type: boolean
   - Description: Indicates whether the file was successfully opened. This is used to inform the user if the program was able to load the airplane data from the specified file.

5. index
   - Data Type: integer
   - Description: Stores the adjusted index of the airplane to be removed. This is used to identify and remove a specific airplane from the array based on user input.

6. fileName
   - Data Type: string
   - Description: The name of the output file to write the airplane data to. This is used to save the current state of the airplane collection when the program terminates.

c. What calculations do you need to do to transform inputs into outputs? List all formulas needed, if applicable. If there are no calculations needed, state there are no calculations for this algorithm. Formulae should reference the variable names from step a and step b as applicable.

No calculations, only functions.

d. Design the logic of your program using pseudocode. Here is where you would use conditionals, loops or functions (if applicable) and list the steps in transforming inputs into outputs. Walk through your logic steps with the test data from the assignment document or the sample run above.

Use the syntax shown at the bottom of this document. Do not include any implementation details (e.g. file names) or C++ specific syntax.

```
—-------------------------------------------------- airplane.h
—--------------------------------------------------
DECLARE constant integer STR_SIZE = 128
DECLARE constant integer ARR_SIZE = 20
DECLARE constant integer LARGE_NUMBER = 1000
DECLARE constant integer ERROR = -1
DECLARE constant integer MAX_COUNT = 200

DECLARE STRUCT Airplane
    DECLARE char model[STR_SIZE]
    DECLARE char make[STR_SIZE]
    DECLARE double maxFuel
    DECLARE integer emptyWeight
    DECLARE integer engineHP
    DECLARE integer maxRange
    DECLARE integer cruiseSpeed
END STRUCT

DECLARE FUNCTION integer loadPlanes(Airplane planes[], ifstream & inFile)
DECLARE FUNCTION boolean addPlane(Airplane planes[], integer & count)
DECLARE FUNCTION boolean insertPlane(Airplane planes[], Airplane newPlane,
integer & count)
DECLARE FUNCTION void printPlanes(Airplane planes[], integer count)
DECLARE FUNCTION boolean openTheFile(ifstream & inFile)
DECLARE FUNCTION void writePlane(Airplane planes[], integer count)
DECLARE FUNCTION void search(Airplane planes[], integer count, const char
searchName[])

—-------------------------------------------------- airplane.cpp
—--------------------------------------------------
FUNCTION boolean openTheFile(ifstream & inFile)
    DECLARE boolean success = false
    DECLARE char fileName[STR_SIZE]
    DISPLAY "Welcome to the airplane collection program!"
    DISPLAY "What is the name of the airplane collection file? "
    INPUT fileName
    CALL inFile.open(fileName)

    WHILE NOT inFile.is_open() AND fileName NOT EQUAL "Q"
        DISPLAY "The file " + fileName + " did not open. Type 'Q' to quit,
or try again now: "
        INPUT fileName
        CALL inFile.open(fileName)
    END WHILE
    IF inFile.is_open() THEN
        SET success = true
    END IF

    RETURN success
END FUNCTION

FUNCTION integer loadPlanes(Airplane planes[], ifstream & inFile)
    DECLARE Airplane newPlane
    DECLARE integer count = 0
    DECLARE boolean success = true

    CALL inFile.getline(newPlane.model, STR_SIZE, ';')

    WHILE NOT inFile.eof() AND success
        CALL inFile.getline(newPlane.make, STR_SIZE, ';')
        CALL inFile >> newPlane.maxFuel
        CALL inFile.ignore()
        CALL inFile >> newPlane.emptyWeight
        CALL inFile.ignore()
        CALL inFile >> newPlane.engineHP
        CALL inFile.ignore()
        CALL inFile >> newPlane.maxRange
        CALL inFile.ignore()
        CALL inFile >> newPlane.cruiseSpeed
        CALL inFile.ignore()

        SET success = insertPlane(planes, newPlane, count)

        IF NOT inFile.eof() AND NOT success THEN
            DISPLAY "Not all planes were loaded from the file, out of room!
Please quit the program, and try again."
        END IF

        CALL inFile.getline(newPlane.model, STR_SIZE, ';')
    END WHILE

    RETURN count
END FUNCTION

FUNCTION boolean addPlane(Airplane planes[], integer & count)
    DECLARE boolean result = false

    IF count < ARR_SIZE THEN
        DECLARE Airplane newPlane
        DISPLAY "What is the model (name) of the airplane? "
        INPUT newPlane.model
        DISPLAY "What is the make (manufacturer) of the airplane? "
        INPUT newPlane.make

        DISPLAY "What is the fuel capacity in gallons? "
        WHILE NOT (cin >> newPlane.maxFuel) OR newPlane.maxFuel < 1.0 OR newPlane.maxFuel > 150.0
            DISPLAY "Invalid input. Please enter a valid fuel capacity between 1.00 and 150.00: "
            CALL cin.clear()
            CALL cin.ignore(LARGE_NUMBER, '\n')
        END WHILE

        DISPLAY "What is the empty weight (in pounds)? "
        WHILE NOT (cin >> newPlane.emptyWeight) OR newPlane.emptyWeight < 1 OR newPlane.emptyWeight > 3000
            DISPLAY "Invalid input. Please enter a valid weight between 1 and 3000 pounds: "
            CALL cin.clear()
            CALL cin.ignore(LARGE_NUMBER, '\n')
        END WHILE

        DISPLAY "What is the horsepower of the engine? "
        WHILE NOT (cin >> newPlane.engineHP) OR newPlane.engineHP < 1 OR newPlane.engineHP > 400
            DISPLAY "Invalid input. Please enter a valid horsepower between 1 and 400: "
            CALL cin.clear()
            CALL cin.ignore(LARGE_NUMBER, '\n')
        END WHILE

        DISPLAY "What is the range? "
        INPUT newPlane.maxRange

        DISPLAY "What is the cruise speed? "
        INPUT newPlane.cruiseSpeed

        SET result = insertPlane(planes, newPlane, count)
    END IF

    RETURN result
END FUNCTION

FUNCTION boolean insertPlane(Airplane planes[], Airplane newPlane, integer & count)
    DECLARE boolean result = false
    DECLARE integer index = 0

    IF count < ARR_SIZE THEN
        IF count == 0 THEN
            SET planes[0] = newPlane
        ELSE IF strcmp(planes[count - 1].model, newPlane.model) <= 0 THEN
            SET planes[count] = newPlane
        ELSE
            WHILE strcmp(planes[index].model, newPlane.model) <= 0
                SET index = index + 1
            END WHILE

            FOR integer i = count DOWNTO index + 1
                SET planes[i] = planes[i - 1]
            END FOR

            SET planes[index] = newPlane
        END IF

        SET count = count + 1
        SET result = true
    END IF

    RETURN result
END FUNCTION

FUNCTION void printPlanes(Airplane planes[], integer count)
    DECLARE char separator[114]

    DISPLAY left SETW(25) "     Model" + left SETW(15) "Make" + right
SETW(13) "Fuel Capacity" + right SETW(15) "Empty Weight" + right SETW(15)
"Horsepower" + right SETW(15) "Range" + right SETW(16) "Cruise Speed\n"

    FOR integer i = 0 TO 114
        SET separator[i] = '-'
    END FOR
    SET separator[113] = '\0'
    DISPLAY separator

    FOR integer i = 0 TO count - 1
        DISPLAY right SETW(2) i + 1 + left SETW(3) ". " + left SETW(20)
planes[i].model + left SETW(18) planes[i].make + right SETW(10) fixed
SETPRECISION(2) planes[i].maxFuel + right SETW(15) planes[i].emptyWeight +
right SETW(15) planes[i].engineHP + right SETW(15) planes[i].maxRange +
right SETW(15) planes[i].cruiseSpeed
    END FOR
END FUNCTION

FUNCTION void writePlane(Airplane planes[], integer count)
    DECLARE char fileName[STR_SIZE]
    DECLARE ofstream outFile
    DISPLAY "What is the name of the file to write to? "
    INPUT fileName
    CALL outFile.open(fileName)
    FOR integer i = 0 TO count - 1
        CALL outFile << planes[i].model << ';' << planes[i].make << ';' <<
planes[i].maxFuel << ';' << planes[i].emptyWeight << ';' <<
planes[i].engineHP << ';' << planes[i].maxRange << ';' <<
planes[i].cruiseSpeed << endl
    END FOR

    CALL outFile.close()
END FUNCTION

—-------------------------------------------------- main.cpp
—--------------------------------------------------

DECLARE FUNCTION integer main()
    DECLARE char choice = ' '
    DECLARE Airplane planes[ARR_SIZE]
    DECLARE ifstream inFile
    DECLARE boolean result = false
    DECLARE boolean success = false
    DECLARE integer count = 0

    SET result = openTheFile(inFile)

    IF result THEN
        SET count = loadPlanes(planes, inFile)
        CALL inFile.close()
        DISPLAY count + " planes were loaded from the file.\n"
        CALL printPlanes(planes, count)
    END IF

    DO
        DISPLAY "\nWhat would you like to do?\n(A)dd a plane\n(L)ist all
planes\n(R)emove a plane by index\n(Q)uit?\n"
        INPUT choice
        CALL cin.ignore()
        SET choice = toupper(choice)

        SELECT choice
            CASE 'A':
                DECLARE integer prevCount = count
                SET success = addPlane(planes, count)
                IF success THEN
                    DECLARE integer newIndex = -1
                    FOR integer i = 0 TO count - 1
                        IF strcmp(planes[i].model, planes[count-1].model)
== 0 AND strcmp(planes[i].make, planes[count-1].make) == 0 THEN
                            SET newIndex = i
                            BREAK
                        END IF
                    END FOR
                    IF newIndex != -1 THEN
                        DISPLAY "Successfully added " +
planes[newIndex].model + " " + planes[newIndex].make + " plane to the
database."
                    ELSE
                        DISPLAY "Successfully added new plane to the
database."
                    END IF
                ELSE
                    DISPLAY "Not added, the array is out of room."
                END IF
                BREAK
            CASE 'L':
                CALL printPlanes(planes,

END PROGRAM
```

5. Pseudocode Syntax

Think about each step in your algorithm as an action and use the verbs below:

To do this:	Use this verb:	Example:
Create a variable	DECLARE	`DECLARE integer num_dogs`
Print to the console window	DISPLAY	`DISPLAY “Hello!”`
Read input from the user into a variable	INPUT	`INPUT num_dogs`
Update the contents of a variable	SET	`SET num_dogs = num_dogs + 1`
Conditionals		

Use a single alternative conditional	IF *condition* THEN
	statement
	statement
	END IF
		`IF num_dogs > 10 THEN`
		`DISPLAY “That is a lot of dogs!”`
		`END IF`

Use a dual alternative conditional	IF *condition* THEN
	statement
	statement
	ELSE
	statement
	statement
		`IF num_dogs > 10 THEN`
		`DISPLAY “You have more than 10 dogs!”`
		`ELSE`
		`DISPLAY “You have ten or fewer dogs!”`

	END IF	`END IF`

Use a switch/case statement	SELECT *variable or expression*
	CASE *value_1:*
	statement
	statement
	CASE *value_2:*
	statement
	statement
	CASE *value_2:*
	statement
	statement
	DEFAULT:
	statement
	statement
	END SELECT
		`SELECT num_dogs`
		`   CASE 0: DISPLAY “No dogs!”`
		`   CASE 1: DISPLAY “One dog..”`
		`   CASE 2: DISPLAY “Two dogs..”`
		`   CASE 3: DISPLAY “Three dogs..”`
		`   DEFAULT: DISPLAY “Lots of dogs!”`
		`END SELECT`

Loops		

Loop while a condition is true - the loop body will execute 0 or more times.	WHILE *condition*
	statement
	statement
	END WHILE
		`SET num_dogs = 1`
		`WHILE num_dogs < 10`
		`   DISPLAY num_dogs, “ dogs!”`
		`   SET num_dogs = num_dogs + 1`
		`END WHILE`

Loop while a condition is true - the loop body will execute 1 or more times.	DO
	statement
	statement
	WHILE *condition*
		`SET num_dogs = 1`
		`DO`
		`   DISPLAY num_dogs, “ dogs!”`
		`   SET num_dogs = num_dogs + 1`
		`WHILE num_dogs < 10`

Loop a specific number of times.	FOR *counter* = *start* TO *end*
	statement
	statement
	END FOR
		`FOR count = 1 TO 10`
		`   DISPLAY num_dogs, “ dogs!”`
		`END FOR`

Functions		

Create a function	FUNCTION *return_type name (parameters)*
	statement
	statement
	END FUNCTION
		`FUNCTION Integer add(Integer num1, Integer num2)`
		`   DECLARE Integer sum`
		`   SET sum = num1 + num2`
		`   RETURN sum`
		`END FUNCTION`

Call a function	CALL *function_name*	`CALL add(2, 3)`
Return data from a function	RETURN *value*	`RETURN 2 + 3`