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.

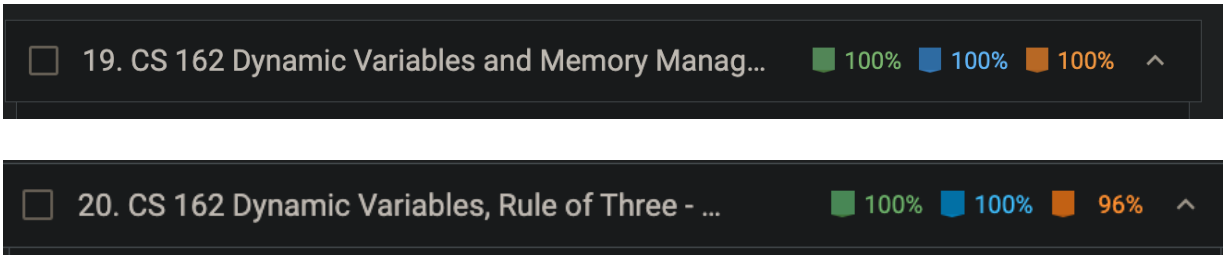
Planning your program before you start coding is part of the development process. In this document you will:

- ❏ 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
- ❏ Plan the algorithmic steps as pseudocode or a UML diagram

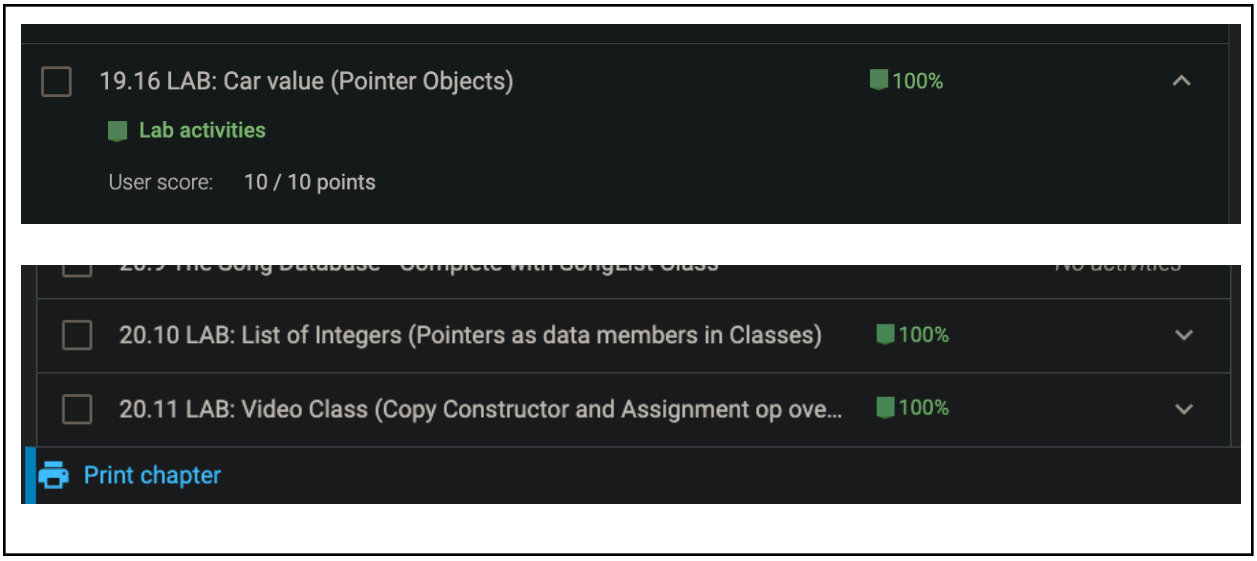
1. zyBooks

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:



Assigned zyLabs completion screenshot:



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 uses memory allocation for c-strings and dynamic arrays to have a user manipulate the Airplanes/Fleet database. The program will be menu-driven that the user controls. The code will focus on data encapsulation as well as saving memory (and creating and clearing memory). The user will be able to view the database in a formatted output and also manipulate the database (add, remove, search for planes).

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:
Welcome to the airplane collection program. What is the name of the airplane collection file? notAFile.txt notAFile.txt was not found. Try again or type 'quit' to exit the program: planes.txt Model Make Fuel Weight HP Range Speed --- 1. 360 Lancair 43 1090 180 990 208 2. Skyhawk 172 Cessna 53 1663 180 515 123

```
 3. K35 Bonanza         Beechcraft       70    1832  250   534     168
 4. RangeMaster H       Navion           40    1945  330   1381    160
 5. Tomahawk            Piper            30    1128  112   383     107
 6. M20R Ovation        Mooney           89    2205  280   969     189
 7. C23 Sundowner       Beechcraft       57    1494  180   564     115
 8. RV-12               Vans Aircraft    20     750  100   451     119
 9. TB-21 GT Trinidad   Socata           88    1911  250   1025    168
10. RV-9                Vans Aircraft    36    1057  160   616     163
11. 152                 Cessna           26    1081  110   414     106
12. Tiger               Grumman          51    1360  180   529     139
13. Super Cub           Piper            36     845  125   449      96


13 planes were loaded from the file.

   Model            Make          Fuel  Weight  HP  Range   Speed
-------------------------------------------------------------------------------
 1. 152             Cessna         26    1081  110   414     106
 2. 360             Lancair        43    1090  180   990     208
 3. C23 Sundowner   Beechcraft     57    1494  180   564     115
 4. K35 Bonanza     Beechcraft     70    1832  250   534     168
 5. M20R Ovation    Mooney         89    2205  280   969     189
 6. RV-12           Vans Aircraft  20     750  100   451     119
 7. RV-9            Vans Aircraft  36    1057  160   616     163
 8. RangeMaster H   Navion         40    1945  330   1381    160
 9. Skyhawk 172     Cessna         53    1663  180   515     123
10. Super Cub       Piper          36     845  125   449      96
11. TB-21 GT Trinidad  Socata      88    1911  250   1025    168
12. Tiger           Grumman        51    1360  180   529     139
13. Tomahawk        Piper          30    1128  112   383     107

Choose an option:
L: List All Planes
M: List Planes by Make
A: Add a New Plane
R: Remove a Plane
Q: Save and Quit

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
OOPS! Enter a decimal number for fuel capacity between 1.00 and 150.00.
Please try again here: 120.00
What is the empty weight? 20500
OOPS! The weight must be a whole number between 1 and 3000 pounds:
Please try again here: 2435
What is the horsepower of the engine? 550
The horsepower must be a whole number between 1 and 400.
Please try again here: 350
What is the range? 1342
What is the cruise speed? 212

→ Malibu Mirage plane data was successfully inserted.

Choose an option:
L: List All Planes
M: List Planes by Make
A: Add a New Plane
R: Remove a Plane
Q: Save and Quit
```

S

For what airplane would you like to search? Boeing 777
The Boeing 777 was not found in the database.

Choose an option:
L: List All Planes
M: List Planes by Make
A: Add a New Plane
R: Remove a Plane
Q: Save and Quit

S

For what airplane would you like to search? Tiger

Information on the Tiger is as follows:
Make: Grumman, Fuel Capacity: 51.00, Empty weight: 1360, Horsepower: 180, Range: 529, Cruise speed: 139

Choose an option:
L: List All Planes
M: List Planes by Make
A: Add a New Plane
R: Remove a Plane
Q: Save and Quit

L

Model	Make	Fuel	Weight	HP	Range	Speed
1. 152	Cessna	26	1081	110	414	106
2. 360	Lancair	43	1090	180	990	208
3. C23 Sundowner	Beechcraft	57	1494	180	564	115
4. K35 Bonanza	Beechcraft	70	1832	250	534	168
5. M20R Ovation	Mooney	89	2205	280	969	189
6. Malibu Mirage	Piper	120	2435	350	1342	212
7. RV-12	Vans Aircraft	20	750	100	451	119
8. RV-9	Vans Aircraft	36	1057	160	616	163
9. RangeMaster H	Navion	40	1945	330	1381	160
10. Skyhawk 172	Cessna	53	1663	180	515	123
11. Super Cub	Piper	36	845	125	449	96
12. TB-21 GT Trinidad	Socata	88	1911	250	1025	168
13. Tiger	Grumman	51	1360	180	529	139
14. Tomahawk	Piper	30	1128	112	383	107

Choose an option:
L: List All Planes
M: List Planes by Make
A: Add a New Plane
R: Remove a Plane
Q: Save and Quit

M

Please type the make of the airplanes you would like to list: Jabiru
There are no airplanes made by Jabiru in the database.

Choose an option:
L: List All Planes
M: List Planes by Make
A: Add a New Plane

```
R: Remove a Plane
Q: Save and Quit

M

Please type the make of the airplanes you would like to list: Piper

The airplanes in the list made by Piper are:



   Model           Make         Fuel   Weight   HP  Range   Speed
-------------------------------------------------------------------------------
 6. Malibu Mirage      Piper          120    2435  350   1342     212
11. Super Cub          Piper          36     845  125    449      96
14. Tomahawk           Piper          30    1128  112    383     107

Choose an option:
L: List All Planes
M: List Planes by Make
A: Add a New Plane
R: Remove a Plane
Q: Save and Quit

R

Which index would you like to remove (1 – 14)? 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
Index 9 has been removed.

Choose an option:
L: List All Planes
M: List Planes by Make
A: Add a New Plane
R: Remove a Plane
Q: Save and Quit

L

   Model           Make         Fuel   Weight   HP  Range   Speed
-------------------------------------------------------------------------------
 1. 152              Cessna          26    1081  110    414     106
 2. 360              Lancair         43    1090  180    990     208
 3. C23 Sundowner       Beechcraft       57    1494  180    564     115
 4. K35 Bonanza         Beechcraft       70    1832  250    534     168
 5. M20R Ovation        Mooney           89    2205  280    969     189
 6. Malibu Mirage       Piper          120    2435  350   1342     212
 7. RV-12             Vans Aircraft     20     750  100    451     119
 8. RV-9              Vans Aircraft     36    1057  160    616     163
 9. Skyhawk 172         Cessna           53    1663  180    515     123
10. Super Cub           Piper          36     845  125    449      96
11. TB-21 GT Trinidad    Socata          88    1911  250   1025     168
12. Tiger              Grumman          51    1360  180    529     139
13. Tomahawk           Piper          30    1128  112    383     107


Choose an option:
L: List All Planes
M: List Planes by Make
A: Add a New Plane
```

```
R: Remove a Plane
Q: Save and Quit

Q

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”.

Variable Name: fileName
Data Type: string
Description: The name of the airplane collection file provided by the user.

Variable Name: option
Data Type: character
Description: The menu option selected by the user to perform different operations.

Variable Name: model
Data Type: string
Description: The model name of the airplane provided by the user.

Variable Name: make
Data Type: string
Description: The make (manufacturer) of the airplane provided by the user.

Variable Name: maxFuel
Data Type: floating point
Description: The maximum fuel capacity of the airplane provided by the user.

Variable Name: emptyWeight
Data Type: integer
Description: The empty weight of the airplane provided by the user.

Variable Name: engineHP

Data Type: integer
Description: The engine horsepower of the airplane provided by the user.

Variable Name: maxRange
Data Type: integer
Description: The maximum range of the airplane provided by the user.

Variable Name: cruiseSpeed
Data Type: integer
Description: The cruise speed of the airplane provided by the user.

Variable Name: index
Data Type: integer
Description: The index of the airplane to be removed, provided 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”.

Variable Name: welcomeMessage
Data Type: string
Description: The welcome message displayed to the user.

Variable Name: menuOptions
Data Type: string
Description: The menu options displayed to the user.

Variable Name: airplaneList
Data Type: array of strings
Description: The list of airplanes in the fleet, including their details.

Variable Name: successMessage
Data Type: string
Description: The success message displayed when an airplane is added or removed.

Variable Name: errorMessage
Data Type: string
Description: The error message displayed for invalid inputs or unsuccessful operations.

Variable Name: searchResults
Data Type: string
Description: The search results displaying the details of the searched airplane.

Variable Name: updatedFileMessage
Data Type: string
Description: The message displayed when the database file is updated.

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.

There are no complex calculations for this algorithm. The primary operations involve reading user inputs, validating them, manipulating arrays, and displaying outputs. The key operations include:

1. Insertion of a new airplane in lexicographical order by model:
   - Finding the correct index to insert the new airplane.
   - Shifting elements to make space for the new airplane.
   - Inserting the new airplane at the correct index.

2. Removal of an airplane by index:
   - Shifting elements to remove the airplane from the array.

3. Validation of user inputs:
   - Checking if the input values are within specified ranges.

These operations primarily involve comparisons, indexing, and basic array manipulations, without complex mathematical calculations.

d. Design the logic of your program by describing the classes, data members, and functions using pseudocode or a UML diagram. If using pseudocode, here is where you would use conditionals, loops or functions (if applicable) and list the steps in transforming inputs into outputs. If creating a UML diagram, use the example below as a model for your own. Step through your logic with the test data from the assignment document or the sample run above.

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

tools.h

```
# Define constants and functions
DECLARE CONSTANT integer STR_SIZE = 100
DECLARE FUNCTION void welcome()
DECLARE FUNCTION void displayMenu()
DECLARE FUNCTION bool validateInput(double &value, double min, double max)
DECLARE FUNCTION bool validateInput(int &value, int min, int max)
```

tools.cpp

```
# Display welcome message
FUNCTION void welcome()
   DISPLAY "Welcome to the airplane collection program!"
   DISPLAY "What is the name of the airplane collection file? "
END FUNCTION

# Display menu options
FUNCTION void displayMenu()
   DISPLAY "Choose an option:"
   DISPLAY "L: List All Planes"
   DISPLAY "M: List Planes by Make"
   DISPLAY "A: Add a New Plane"
```

```
    DISPLAY "R: Remove a Plane"
    DISPLAY "Q: Save and Quit"
END FUNCTION

# Validate double input within range
FUNCTION bool validateInput(double &value, double min, double max)
    INPUT value
    IF cin.fail() OR value < min OR value > max THEN
        CALL cin.clear()
        CALL cin.ignore(10000, '\n')
        RETURN false
    END IF
    RETURN true
END FUNCTION

# Validate integer input within range
FUNCTION bool validateInput(int &value, int min, int max)
    INPUT value
    IF cin.fail() OR value < min OR value > max THEN
        CALL cin.clear()
        CALL cin.ignore(10000, '\n')
        RETURN false
    END IF
    RETURN true
END FUNCTION

# airplane.h

# Define Airplane class with attributes and methods
DECLARE CLASS Airplane
    DECLARE PRIVATE char* make
    DECLARE PRIVATE char* model
    DECLARE PRIVATE double maxFuel
    DECLARE PRIVATE int emptyWeight
    DECLARE PRIVATE int engineHP
    DECLARE PRIVATE int maxRange
    DECLARE PRIVATE int cruiseSpeed

    DECLARE PUBLIC FUNCTION Airplane()  # Default constructor
    DECLARE PUBLIC FUNCTION Airplane(const Airplane &)  # Copy constructor
    DECLARE PUBLIC FUNCTION ~Airplane()  # Destructor
    DECLARE PUBLIC FUNCTION Airplane &operator=(const Airplane &)  # Copy assignment
operator

    # Getters
    DECLARE PUBLIC FUNCTION const char* getMake()
    DECLARE PUBLIC FUNCTION const char* getModel()
    DECLARE PUBLIC FUNCTION double getMaxFuel()
    DECLARE PUBLIC FUNCTION int getEmptyWeight()
    DECLARE PUBLIC FUNCTION int getEngineHP()
    DECLARE PUBLIC FUNCTION int getMaxRange()
    DECLARE PUBLIC FUNCTION int getCruiseSpeed()

    # Setters
    DECLARE PUBLIC FUNCTION void setMake(const char *)
    DECLARE PUBLIC FUNCTION void setModel(const char *)
    DECLARE PUBLIC FUNCTION void setMaxFuel(double)
    DECLARE PUBLIC FUNCTION void setEmptyWeight(int)
```

```
   DECLARE PUBLIC FUNCTION void setEngineHP(int)
   DECLARE PUBLIC FUNCTION void setMaxRange(int)
   DECLARE PUBLIC FUNCTION void setCruiseSpeed(int)
END CLASS

# airplane.cpp

# Default constructor
FUNCTION Airplane::Airplane()
   DECLARE make = new char[20]
   SET make = "None"
   DECLARE model = new char[20]
   SET model = "None"
   SET maxFuel = 0.0
   SET emptyWeight = 0
   SET engineHP = 0
   SET maxRange = 0
   SET cruiseSpeed = 0
END FUNCTION

# Copy constructor
FUNCTION Airplane::Airplane(const Airplane &other)
   DECLARE make = new char[strlen(other.make) + 1]
   SET make = other.make
   DECLARE model = new char[strlen(other.model) + 1]
   SET model = other.model
   SET maxFuel = other.maxFuel
   SET emptyWeight = other.emptyWeight
   SET engineHP = other.engineHP
   SET maxRange = other.maxRange
   SET cruiseSpeed = other.cruiseSpeed
END FUNCTION

# Copy assignment operator
FUNCTION Airplane& Airplane::operator=(const Airplane &other)
   IF this == &other THEN
      RETURN *this
   END IF
   CALL delete[] make
   CALL delete[] model
   DECLARE make = new char[strlen(other.make) + 1]
   SET make = other.make
   DECLARE model = new char[strlen(other.model) + 1]
   SET model = other.model
   SET maxFuel = other.maxFuel
   SET emptyWeight = other.emptyWeight
   SET engineHP = other.engineHP
   SET maxRange = other.maxRange
   SET cruiseSpeed = other.cruiseSpeed
   RETURN *this
END FUNCTION

# Destructor
FUNCTION Airplane::~Airplane()
   CALL delete[] make
   CALL delete[] model
END FUNCTION
```

```
# Getters
FUNCTION const char* Airplane::getMake()
  RETURN make
END FUNCTION

FUNCTION const char* Airplane::getModel()
  RETURN model
END FUNCTION

FUNCTION double Airplane::getMaxFuel()
  RETURN maxFuel
END FUNCTION

FUNCTION int Airplane::getEmptyWeight()
  RETURN emptyWeight
END FUNCTION

FUNCTION int Airplane::getEngineHP()
  RETURN engineHP
END FUNCTION

FUNCTION int Airplane::getMaxRange()
  RETURN maxRange
END FUNCTION

FUNCTION int Airplane::getCruiseSpeed()
  RETURN cruiseSpeed
END FUNCTION

# Setters
FUNCTION void Airplane::setMake(const char* make)
  CALL delete[] this->make
  DECLARE this->make = new char[strlen(make) + 1]
  SET this->make = make
END FUNCTION

FUNCTION void Airplane::setModel(const char* model)
  CALL delete[] this->model
  DECLARE this->model = new char[strlen(model) + 1]
  SET this->model = model
END FUNCTION

FUNCTION void Airplane::setMaxFuel(double maxFuel)
  SET this->maxFuel = maxFuel
END FUNCTION

FUNCTION void Airplane::setEmptyWeight(int emptyWeight)
  SET this->emptyWeight = emptyWeight
END FUNCTION

FUNCTION void Airplane::setEngineHP(int engineHP)
  SET this->engineHP = engineHP
END FUNCTION

FUNCTION void Airplane::setMaxRange(int maxRange)
  SET this->maxRange = maxRange
END FUNCTION
```

```
FUNCTION void Airplane::setCruiseSpeed(int cruiseSpeed)
   SET this->cruiseSpeed = cruiseSpeed
END FUNCTION

# fleet.h

# Define Fleet class with attributes and methods
DECLARE CLASS Fleet
   DECLARE PRIVATE char fileName[STR_SIZE]
   DECLARE PRIVATE ifstream inFile
   DECLARE PRIVATE int count
   DECLARE PRIVATE int capacity
   DECLARE PRIVATE Airplane* fleetAirplanes
   DECLARE PRIVATE bool insert()
   DECLARE PUBLIC FUNCTION Fleet()  # Default constructor
   DECLARE PUBLIC FUNCTION ~Fleet()  # Destructor
   DECLARE PUBLIC FUNCTION int loadPlanes()
   DECLARE PUBLIC FUNCTION void printPlanes()
   DECLARE PUBLIC FUNCTION void listByMake()
   DECLARE PUBLIC FUNCTION void search()
   DECLARE PUBLIC FUNCTION bool addAPlane()
   DECLARE PUBLIC FUNCTION bool removeAPlane()
   DECLARE PUBLIC FUNCTION void writePlanes()
   DECLARE PUBLIC FUNCTION void growArray()
   DECLARE PUBLIC FUNCTION bool openTheFile()
END CLASS

# fleet.cpp

# Default constructor
FUNCTION Fleet::Fleet()
   SET count = 0
   SET capacity = 3
   DECLARE fleetAirplanes = new Airplane[capacity]
END FUNCTION

# Destructor
FUNCTION Fleet::~Fleet()
   CALL delete[] fleetAirplanes
END FUNCTION

# Grow array capacity
FUNCTION void Fleet::growArray()
   SET capacity = capacity * 2
   DECLARE newArray = new Airplane[capacity]
   FOR integer i = 0 TO count - 1
      SET newArray[i] = fleetAirplanes[i]
   END FOR
   CALL delete[] fleetAirplanes
   SET fleetAirplanes = newArray
END FUNCTION

# Insert airplane in lexicographical order
FUNCTION bool Fleet::insert()
   DECLARE tempPlane = fleetAirplanes[count]
   DECLARE bool result = false
   DECLARE integer index = 0
```

```
  IF count >= capacity THEN
    CALL growArray()
  END IF

  FOR index = 0 TO count - 1
    IF strcmp(fleetAirplanes[index].getModel(), tempPlane.getModel()) > 0 THEN
      BREAK
    END IF
  END FOR

  FOR integer i = count TO index + 1 STEP -1
    SET fleetAirplanes[i] = fleetAirplanes[i - 1]
  END FOR

  SET fleetAirplanes[index] = tempPlane
  SET count = count + 1
  SET result = true
  RETURN result
END FUNCTION

# Load airplanes from file
FUNCTION int Fleet::loadPlanes()
  DECLARE char model[STR_SIZE]
  DECLARE char make[STR_SIZE]
  DECLARE double maxFuel = 0.0
  DECLARE int emptyWeight = 0
  DECLARE int engineHP = 0
  DECLARE int maxRange = 0
  DECLARE int cruiseSpeed = 0

  DECLARE ifstream inFile(fileName)
  IF NOT inFile.is_open() THEN
    DISPLAY "Failed to open file."
    RETURN -1
  END IF

  WHILE inFile.getline(model, STR_SIZE, ';') DO
    CALL inFile.getline(make, STR_SIZE, ';')
    INPUT maxFuel
    CALL inFile.ignore()
    INPUT emptyWeight
    CALL inFile.ignore()
    INPUT engineHP
    CALL inFile.ignore()
    INPUT maxRange
    CALL inFile.ignore()
    INPUT cruiseSpeed
    CALL inFile.ignore()

    DECLARE Airplane plane
    CALL plane.setModel(model)
    CALL plane.setMake(make)
    CALL plane.setMaxFuel(maxFuel)
    CALL plane.setEmptyWeight(emptyWeight)
    CALL plane.setEngineHP(engineHP)
    CALL plane.setMaxRange(maxRange)
    CALL plane.setCruiseSpeed(cruiseSpeed)
```

```
      IF count >= capacity THEN
         CALL growArray()
      END IF

      SET fleetAirplanes[count] = plane
      CALL insert()
   END WHILE

   CALL inFile.close()
   RETURN count
END FUNCTION

# Print airplanes
FUNCTION void Fleet::printPlanes()
   DISPLAY ""

   DISPLAY ""
   DISPLAY "Model              Make          Fuel Weight HP Range Speed"
   DISPLAY "-------------------------------------------------------------"

   FOR integer i = 0 TO count - 1
      DISPLAY (i + 1) ". " fleetAirplanes[i].getModel() fleetAirplanes[i].getMake()
      DISPLAY fleetAirplanes[i].getMaxFuel() fleetAirplanes[i].getEmptyWeight()
      DISPLAY fleetAirplanes[i].getEngineHP() fleetAirplanes[i].getMaxRange()
      DISPLAY fleetAirplanes[i].getCruiseSpeed()
   END FOR
END FUNCTION

# List airplanes by make
FUNCTION void Fleet::listByMake()
   DECLARE char make[STR_SIZE]
   DECLARE bool found = false
   DISPLAY "Please type the make of the airplanes you would like to list: "
   INPUT make
   DISPLAY "The airplanes in the list made by " make " are:"

   DISPLAY ""
   DISPLAY "Model              Make          Fuel Weight HP Range Speed"
   DISPLAY "-------------------------------------------------------------"

   FOR integer i = 0 TO count - 1
      IF strcmp(fleetAirplanes[i].getMake(), make) == 0 THEN
         DISPLAY (i + 1) ". " fleetAirplanes[i].getModel() fleetAirplanes[i].getMake()
         DISPLAY fleetAirplanes[i].getMaxFuel() fleetAirplanes[i].getEmptyWeight()
         DISPLAY fleetAirplanes[i].getEngineHP() fleetAirplanes[i].getMaxRange()
         DISPLAY fleetAirplanes[i].getCruiseSpeed()
         SET found = true
      END IF
   END FOR

   IF NOT found THEN
      DISPLAY "There are no airplanes made by " make " in the database."
   END IF
END FUNCTION

# Add a new airplane
FUNCTION bool Fleet::addAPlane()
   DECLARE char model[STR_SIZE]
```

```
   DECLARE char make[STR_SIZE]
   DECLARE double maxFuel = 0.0
   DECLARE int emptyWeight = 0
   DECLARE int engineHP = 0
   DECLARE int maxRange = 0
   DECLARE int cruiseSpeed = 0

   DISPLAY "What is the model (name) of the airplane? "
   INPUT model
   DISPLAY "What is the make (manufacturer) of the airplane? "
   INPUT make

   DISPLAY "What is the fuel capacity in gallons? "
   WHILE NOT validateInput(maxFuel, 1.0, 150.0) DO
      CALL cin.clear()
      DISPLAY "You must enter a decimal number for fuel capacity between 1.00 and 150.00.
Please try again: "
   END WHILE

   DISPLAY "What is the empty weight? "
   WHILE NOT validateInput(emptyWeight, 1, 3000) DO
      CALL cin.clear()
      DISPLAY "The weight must be a whole number between 1 and 3000 pounds. Please try
again: "
   END WHILE

   DISPLAY "What is the horsepower of the engine? "
   WHILE NOT validateInput(engineHP, 1, 400) DO
      CALL cin.clear()
      DISPLAY "The horsepower must be a whole number between 1 and 400. Please try
again: "
   END WHILE

   DISPLAY "What is the max range? "
   WHILE NOT validateInput(maxRange, 1, 2000) DO
      CALL cin.clear()
      DISPLAY "The range must be a whole number between 1 and 2000 nautical miles. Please
try again: "
   END WHILE

   DISPLAY "What is the cruise speed? "
   WHILE NOT validateInput(cruiseSpeed, 1, 250) DO
      CALL cin.clear()
      DISPLAY "The cruise speed must be a whole number between 1 and 250 knots. Please
try again: "
   END WHILE

   DECLARE Airplane airplane
   CALL airplane.setModel(model)
   CALL airplane.setMake(make)
   CALL airplane.setMaxFuel(maxFuel)
   CALL airplane.setEmptyWeight(emptyWeight)
   CALL airplane.setEngineHP(engineHP)
   CALL airplane.setMaxRange(maxRange)
   CALL airplane.setCruiseSpeed(cruiseSpeed)

   IF count >= capacity THEN
      CALL growArray()
```

```
    END IF

    SET fleetAirplanes[count] = airplane
    CALL insert()
    DISPLAY "--> " airplane.getModel() " plane data was successfully inserted."
    RETURN true
END FUNCTION

# Remove an airplane
FUNCTION bool Fleet::removeAPlane()
    DECLARE integer index = 0
    DISPLAY "Which index would you like to remove (1 – " count ")? "
    WHILE NOT validateInput(index, 1, count) DO
        DISPLAY "Invalid Index. Please type an index between 1 and " count ": "
    END WHILE

    DISPLAY "--> " fleetAirplanes[index - 1].getModel() " at index " index " has been removed."

    FOR integer i = index - 1 TO count - 2
        SET fleetAirplanes[i] = fleetAirplanes[i + 1]
    END FOR

    SET count = count - 1
    RETURN true
END FUNCTION

# Write airplanes to file
FUNCTION void Fleet::writePlanes()
    DISPLAY "... file is writing to file output.txt..."

    DECLARE ofstream outFile("output.txt")
    IF NOT outFile.is_open() THEN
        DISPLAY "Failed to open file for writing."
        RETURN
    END IF

    FOR integer i = 0 TO count - 1
        CALL outFile << fleetAirplanes[i].getModel() << ";"
        CALL outFile << fleetAirplanes[i].getMake() << ";"
        CALL outFile << fleetAirplanes[i].getMaxFuel() << ";"
        CALL outFile << fleetAirplanes[i].getEmptyWeight() << ";"
        CALL outFile << fleetAirplanes[i].getEngineHP() << ";"
        CALL outFile << fleetAirplanes[i].getMaxRange() << ";"
        CALL outFile << fleetAirplanes[i].getCruiseSpeed() << endl
    END FOR

    CALL outFile.close()
    DISPLAY "Database file updated. Terminating Program."
END FUNCTION

# Search for an airplane
FUNCTION void Fleet::search()
    DECLARE char model[STR_SIZE]

    DISPLAY "For what airplane would you like to search? "
    INPUT model

    FOR integer i = 0 TO count - 1
```

```
      IF strcmp(fleetAirplanes[i].getModel(), model) == 0 THEN
         DISPLAY "Information on the " model " is as follows:"
         DISPLAY "Make: " fleetAirplanes[i].getMake()
         DISPLAY "Fuel Capacity: " fleetAirplanes[i].getMaxFuel()
         DISPLAY "Empty weight: " fleetAirplanes[i].getEmptyWeight()
         DISPLAY "Horsepower: " fleetAirplanes[i].getEngineHP()
         DISPLAY "Range: " fleetAirplanes[i].getMaxRange()
         DISPLAY "Cruise speed: " fleetAirplanes[i].getCruiseSpeed()
         RETURN
      END IF
   END FOR

   DISPLAY "The " model " was not found in the database."
END FUNCTION

# Open the file
FUNCTION bool Fleet::openTheFile()
   DECLARE bool success = false
   INPUT fileName
   CALL inFile.open(fileName)

   WHILE NOT inFile.is_open() AND strcmp(fileName, "Q") != 0 DO
      DISPLAY "*** The file " fileName " was not found. 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

# main.cpp

# Main function
DECLARE FUNCTION int main()
   DECLARE Fleet fleet
   DECLARE bool success = false
   DECLARE integer count = 0
   DECLARE char option = ' '

   CALL welcome()

   IF NOT fleet.openTheFile() THEN
      DISPLAY "Quitting the program because the file didn't open."
   ELSE
      SET count = fleet.loadPlanes()
      DISPLAY count " planes were loaded from the file."
      CALL fleet.printPlanes()
   END IF

   DO
      CALL displayMenu()
      INPUT option
      SET option = toupper(option)
      CALL cin.ignore()
```

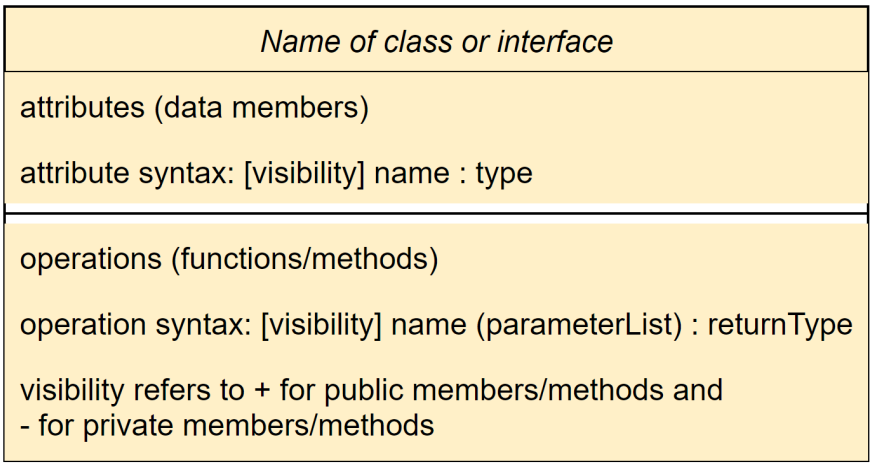
```
    SELECT option
      CASE 'L':
        CALL fleet.printPlanes()
      CASE 'M':
        CALL fleet.listByMake()
      CASE 'A':
        SET success = fleet.addAPlane()
        IF NOT success THEN
          DISPLAY "The new plane was not added. Out of room."
        END IF
      CASE 'R':
        CALL fleet.removeAPlane()
      CASE 'S':
        CALL fleet.search()
      CASE 'Q':
        CALL fleet.writePlanes()
      DEFAULT:
        DISPLAY "Invalid choice. Please try again."
    END SELECT
  WHILE option != 'Q'

  RETURN 0
END FUNCTION
```

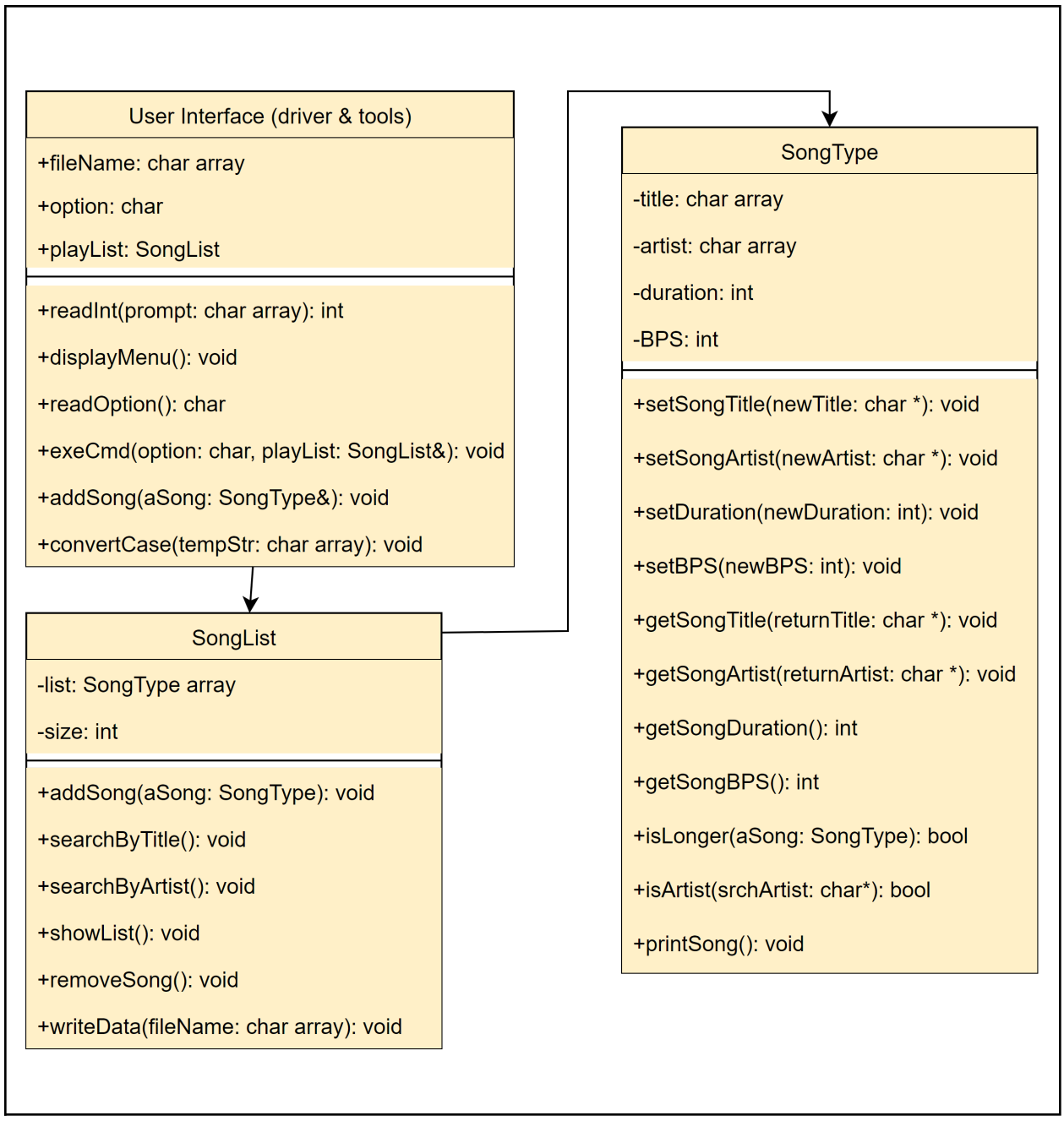
5. UML Syntax Example:

The following diagram uses the Unified Modeling Language, or UML, syntax. This example was made using draw.io and is based on the program found in ZyBooks 18.2: CS162 Classes Part 2, 18.2 C++ example: SongType Class and SongList Class

How to structure your UML diagram:



UML example based on ZyBooks 18.2 SongType Class and SongList Class:



6. 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	INPUT	`INPUT num_dogs`

into a variable		
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
	END IF
		`IF num_dogs > 10 THEN`
		`  DISPLAY "You have more than 10 dogs!"`
		`ELSE`
		`  DISPLAY "You have ten or fewer dogs!"`
		`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`