CS161B - A06: Programming and Problem Solving I

Algorithm Design Document - Starter file

Note: This document already has some of the design - you need to add the rest at the appropriate places.

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:

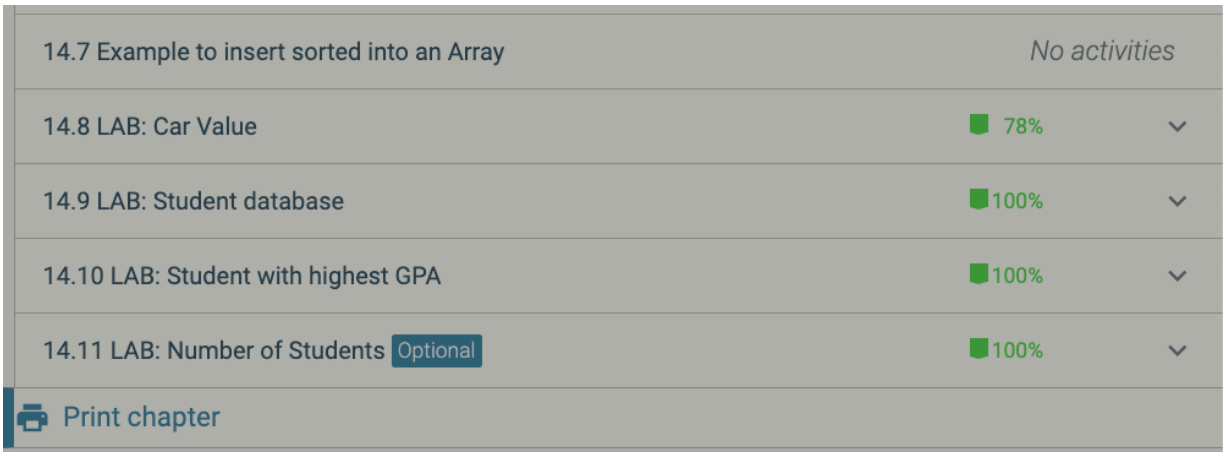
- ❏ Paste a screenshot of your zyBooks Challenge and Participation %
- ❏ Paste a screenshot of your assigned zyLabs completion
- ❏ Write a detailed description of your program, at least two complete sentences
- ❏ If applicable, design a sample run with test input and output
- ❏ Identify the program inputs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

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:



The Zylab 14.8 I emailed you about it giving me an environment specific error. When I reported it to Zybooks they told me to ask you since it was instructor created. But when I run the same exact code in Replit, I get zero errors. I'll screenshot my code here just to show you:

main.cpp

```
#include <iostream>
#include "car.h"
using namespace std;

int main() {
	Car myCar = InitCar();

	int userYear;
	int userPrice;
	int userCurrentYear;

	cin >> userYear;
	cin >> userPrice;
	cin >> userCurrentYear;

	SetModelYear(userYear, myCar);
	SetPurchasePrice(userPrice, myCar);
	myCar = CalcCurrentValue(userCurrentYear, myCar);

	PrintInfo(myCar);

	return 0;
}
```

car.h

```
#ifndef CAR_H_
#define CAR_H_

struct Car {
	int modelYear;
	int purchasePrice;
	int currentValue;
};

Car InitCar();
void SetModelYear(int userYear, Car& car);
int GetModelYear(Car car);
// TODO: Declare SetPurchasePrice() function
void SetPurchasePrice(int userPrice, Car& car);
// TODO: Declare GetPurchasePrice() function
int GetPurchasePrice(Car car);
Car CalcCurrentValue(int currentYear, Car car);
// TODO: Declare PrintInfo() function to output model_year, purchase_price, and
current_value
void PrintInfo(Car car);
#endif
```

car.cpp

```
#include <iostream>
#include <cmath>

#include "car.h"
using namespace std;


Car InitCar() {
	Car car;
	car.modelYear = 2020;
	car.purchasePrice = 0;
	car.currentValue = 0;
	return car;
}

void SetModelYear(int userYear, Car& car) {
	car.modelYear = userYear;
}

int GetModelYear(Car car) {
	return car.modelYear;
}

// TODO: Define SetPurchasePrice() function
void SetPurchasePrice(int userPrice, Car& car) {
```

```
car.purchasePrice = userPrice;
}

// TODO: Define GetPurchasePrice() function
int GetPurchasePrice(Car car) {
   return car.purchasePrice;
}

Car CalcCurrentValue(int currentYear, Car car) {
    double depreciationRate = 0.15;
    int carAge = currentYear - car.modelYear;

    //Car deprecation formula
    car.currentValue = static_cast<int>(round(car.purchasePrice * pow((1 -
depreciationRate), carAge)));
    return car;
}

// TODO: Define PrintInfo() function to output model_year, purchase_price, and
current_value
void PrintInfo(Car car) {
 cout << "Car's information:\n";
 cout << "   Model year: " << car.modelYear<<endl;
 cout << "   Purchase price: " << car.purchasePrice << endl;
 cout << "   Current value: " << car.currentValue;
}
```

Zybooks error:

```
main.cpp: In function 'int testPassed(std::ofstream&)':
main.cpp:17:41: error: no match for 'operator=' (operand types are
'Car' and 'void')
   17 |    myCar = SetPurchasePrice(18000, myCar);
      |                                         ^
In file included from main.cpp:3:
car.h:4:8: note: candidate: 'constexpr Car& Car::operator=(const
Car&)'
    4 | struct Car {
      |        ^~~
car.h:4:8: note:   no known conversion for argument 1 from 'void'
to 'const Car&'
car.h:4:8: note: candidate: 'constexpr Car& Car::operator=(Car&&)'
car.h:4:8: note:   no known conversion for argument 1 from 'void'
to 'Car&&'
```

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:
The purpose of this program is to keep track of a Course and its roster. The Course has a list of students and the number of students on the roster. Each student has a first name, last name and gpa for the attributes. The program loads from a file into the course roster - the functionalities are to add a student, remove a student, print the roster, print a single student and to find the student with the highest gpa.

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.

Sample run:

```
First sample run without modifying addStudent:

Welcome to my Course Roster Program

Here is the course roster:
Henry;Nguyen;3.5
Brenda;Stern;2
Lynda;Robison;3.2
Sonya;King;3.9
Gayathri;Iyer;3.5
Glen;Sasek;3.7
Priya;Goel;3.8

Enter the last name of the student to drop: King

Here is the course roster:
Henry;Nguyen;3.5
Brenda;Stern;2
```

```
Lynda;Robison;3.2
Gayathri;Iyer;3.5
Glen;Sasek;3.7
Priya;Goel;3.8

The student with the highest GPA:
Priya;Goel;3.8

Thank you for using my Student Roster program!!
```

```
Second sample run after modifying addStudent:

Welcome to my Course Roster Program

Here is the course roster:
Priya;Goel;3.8
Gayathri;Iyer;3.5
Sonya;King;3.9
Henry;Nguyen;3.5
Lynda;Robison;3.2
Glen;Sasek;3.7
Brenda;Stern;2

Enter the last name of the student to drop: Iyer

Here is the course roster:
Priya;Goel;3.8
Sonya;King;3.9
Henry;Nguyen;3.5
Lynda;Robison;3.2
Glen;Sasek;3.7
Brenda;Stern;2

The student with the highest GPA:
Sonya;King;3.9

Thank you for using my Student Roster 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 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 (for CS161B and up).
● lastNameToDrop: Char array. Stores the last name of the student to be dropped.
b. Identify and list all of the user output 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” (for CS161B and up).
● Student Information: String and Floating point; prints student's first name, last name, and GPA in the format "FirstName;LastName;GPA". ● Course Roster: Array of String; lists each student's information in the course roster. ● Highest GPA Student: String and Floating point; shows the first name, last name, and GPA of the student with the highest GPA.
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.
● Comparison to find the student with the highest gpa.
d. Design the logic of your program using pseudocode or flowcharts. 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 and plain English phrases. Do not include any implementation details (e.g. file names) or C++ specific syntax.**
Main.cpp: ○ DECLARE Course course

- CALL initCourse TO course
- DECLARE ifstream inFile
- OPEN inFile WITH "students.txt"
- IF inFile FAILS THEN
- DISPLAY "File did not open! Program Exiting!!"
- RETURN 1
- END IF

- CALL readStudent WITH inFile, course

- DISPLAY "Welcome to my Course Roster Program"
- DISPLAY "Here is the course roster: "
- CALL printRoster WITH course

- DECLARE char array lastNameToDrop[20]
- DISPLAY "Enter the last name of the student to drop: "
- INPUT lastNameToDrop
- CALL dropStudent WITH lastNameToDrop, course

- DISPLAY "Here is the course roster after dropping a student: "
- CALL printRoster WITH course
- DECLARE Student topStudent
- CALL findStudentHighestGPA WITH course TO topStudent
- DISPLAY "The student with the highest GPA is: "
- CALL printStd WITH topStudent

- CLOSE inFile
- DISPLAY "Thank you for using my Student Roster program!!"
- END

Student.h:

- // Prevent multiple inclusions of header
- IF NOT DEFINED STUDENT_H THEN
- DEFINE STUDENT_H
- // Define the Student structure
- STRUCT Student
- DECLARE char array first[20]
- DECLARE char array last[20]
- DECLARE double gpa
- END STRUCT
- // Function prototypes for Student-related operations
- FUNCTION Student initStudent(char *first, char *last, double gpa)
- FUNCTION VOID getLastName(char *studentName, Student student)
- FUNCTION double getGPA(Student student)
- FUNCTION VOID printStd(Student student)
- END IF

Student.cpp:

- FUNCTION Student initStudent(char *first, char *last, double gpa)

- DECLARE Student student
- SET student.first TO first
- SET student.last TO last
- SET student.gpa TO gpa
- RETURN student
- END FUNCTION
- FUNCTION VOID printStd(Student student)
- DISPLAY student.first, ";", student.last, ";", student.gpa
- END FUNCTION

Course.h:

```
// Prevent multiple inclusions of header
IF NOT DEFINED COURSE_H THEN
   DEFINE COURSE_H

   // Include the Student header for dependency
   INCLUDE "Student.h"

   // Define the Course structure
   STRUCT Course
      DECLARE Student array roster[20]
      DECLARE integer numStudents
   END STRUCT

   // Function prototypes for Course-related operations
   FUNCTION Course initCourse()
   FUNCTION VOID readStudent(ifstream &inFile, Course &course)
   FUNCTION VOID addStudent(Student student, Course &course)
   FUNCTION VOID dropStudent(char *lastname, Course &course)
   FUNCTION Student findStudentHighestGPA(Course course)
   FUNCTION VOID printRoster(Course course)

END IF
```

Course.cpp:

```
FUNCTION Course initCourse()
   DECLARE Course course
   SET course.numStudents = 0
   RETURN course
END FUNCTION

FUNCTION VOID readStudent(ifstream &inFile, Course &course)
   DECLARE Student student
   WHILE NOT inFile.eof() AND course.numStudents <= 19 DO
      READ student.first FROM inFile UNTIL ';'
      IGNORE inFile UNTIL ';'
      READ student.last FROM inFile UNTIL ';'
      IGNORE inFile UNTIL ';'
      INPUT inFile TO student.gpa
      IGNORE inFile UNTIL '\n'

      CALL addStudent WITH student, course
```

```
    END WHILE
END FUNCTION

FUNCTION VOID addStudent(Student student, Course &course)
    IF course.numStudents < 20 THEN
        SET course.roster[course.numStudents] TO student
        INCREMENT course.numStudents
    END IF
END FUNCTION

FUNCTION VOID dropStudent(char *lastname, Course &course)
    FOR EACH student IN course.roster DO
        IF student.last MATCHES lastname THEN
            REMOVE student FROM course.roster
            DECREMENT course.numStudents
            BREAK
        END IF
    END FOR
END FUNCTION

FUNCTION Student findStudentHighestGPA(Course course)
    IF course.numStudents == 0 THEN
        RETURN initStudent("None", "None", 0.0)
    END IF

    DECLARE Student topStudent SET TO course.roster[0]
    FOR EACH student IN course.roster DO
        IF student.gpa > topStudent.gpa THEN
            SET topStudent TO student
        END IF
    END FOR
    RETURN topStudent
END FUNCTION

FUNCTION VOID printRoster(Course course)
    FOR EACH student IN course.roster DO
        CALL printStd WITH student
    END FOR
END FUNCTION
```

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
	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	FOR *counter* = *start* TO *end*	`FOR count = 1 TO 10`

of times.	*statement* *statement* END FOR	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