CS 161A: Programming and Problem Solving I

Assignment A07 Algorithmic 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.

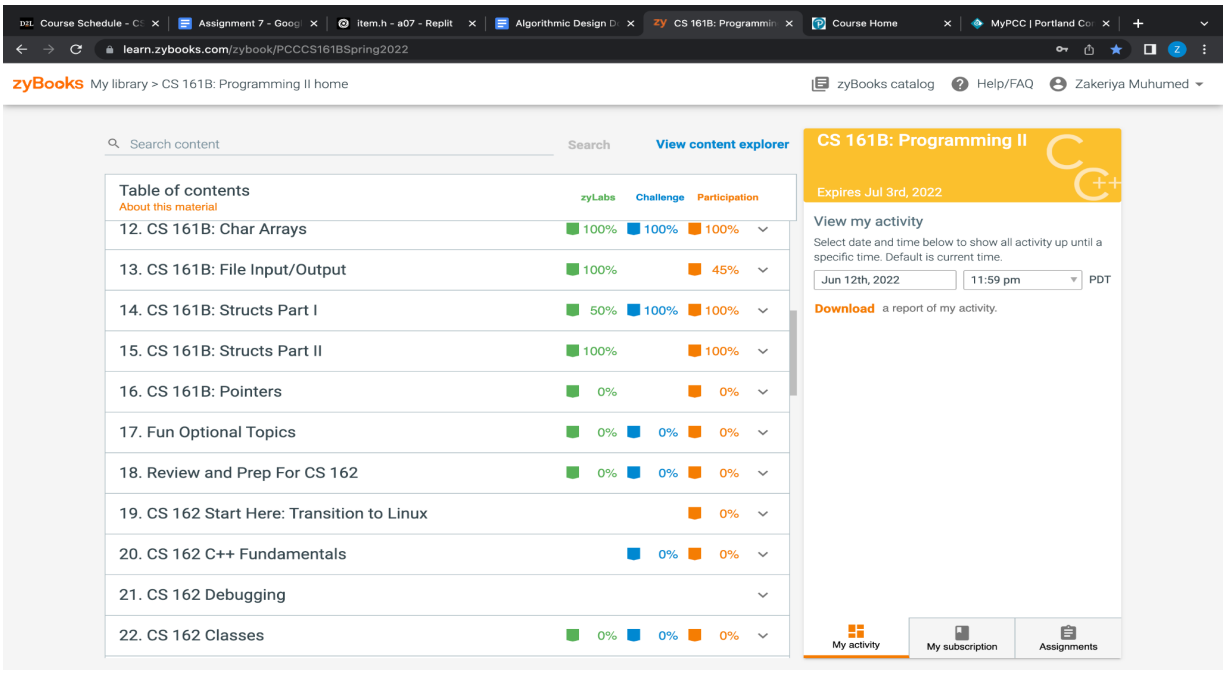
Planning your program before you start coding is part of the development process. In this document you will:

- ❏ 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 or a flowchart
- ❏ Tools for flowchart - Draw.io - Diagrams.net

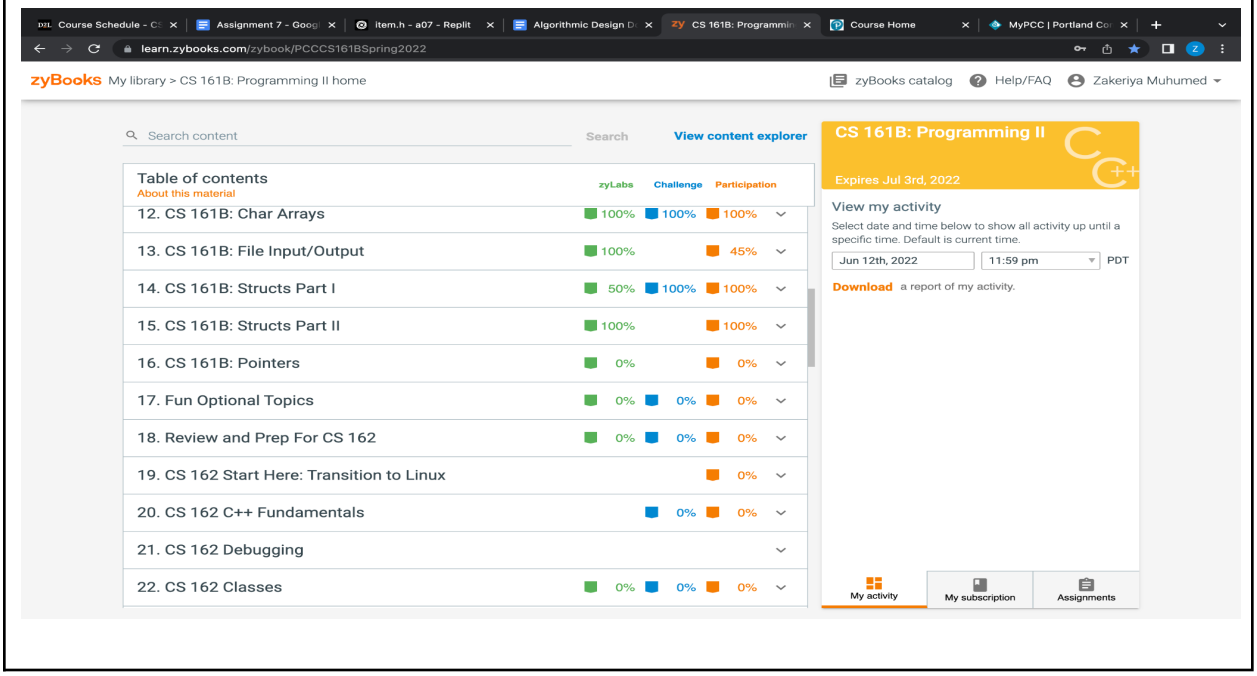
1. zyBooks

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:



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:
You will modify the program you wrote for Assignment 6 to use an array of structs. Each struct will hold the occupation name, the total number employed, and the number of employed susceptible to automation in that occupation.

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:

```
Occupation Name          Employed  Automation  Percent %
_______________          ________  __________  _________
Administrative           23081     13849       60.0017%
Agriculture              1060      594         56.0377%
Arts Entertainment       2773      555         20.0144%
Business                 8067      1129        13.9953%
Computer                 4419      1635        36.9993%
Construction             6813      3407        50.0073%
Education                9427      1697        18.0015%
Engineering              2601      494         18.9927%
Facilities Care          5905      2893        48.9924%
Food Service             13206     10697       81.0011%
Health Practitioner      8752      2888        32.9982%
Health Support           4316      1726        39.9907%
Legal                    1283      488         38.0359%
Maintenance              5654      1187        20.994%
Management               9533      2193        23.0043%
Personal Care            6420      2183        34.0031%
Production               9357      7592        81.1371%
Protective               3506      1262        35.9954%
Sales                    15748     6772        43.0023%
Science                  1300      416         32    %
Social Service           2571      566         22.0148%
```

```
Transportation           10274     5651        55.0029%
Highest/Lowest Occupations Susceptible to Automation:
Production has the highest share (81%)
Business has the lowest share (14%)
```

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.

Algorithmic design:
● Identify and list all of the user input and their data types.
○ File data.txt
● Identify and list all of the user output and their data types.
○ low as int- count the number of occupation between 0-30% ○ medium as int- count the number of occupation between 30-70% ○ high as int- count the number of occupation between 70-100%
● 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.
○ percent = static_cast<double>(ccupation[i].automation) / static_cast<double>(ccupation[i].employed) *100; ○ Number of occupation in this file total to low + medium + high ○ Number of occupation > 70% (high*100)/total ○ Number of occupation > 30%(medium * 100)/total ○ Number of occupation < 30%(low*100)/total
● 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.

Occupation.H

1. **STRUCT Item** with 3 data members -
    a. occupation as a char array
    b. employed as an integer
    c. automation as an integer
2. **Function prototypes** for -
    a. **loadData** - to load data from file into the array of structs
    b. **printData** - to print from the array of structs
    c. **FindIow**- to find the item with lowest percent
    d. **findhigh** - to find the occupation with the highest percent

Occupation.CPP

3. **FUNCTION loadData** (takes, file stream var, int rows)
    a. **Read** the first occupation from the text file into the struct occupation. name list
    b. **WHILE** not end of file
        i. **Read** the employed into the occupation. employed array
        ii. **read** the automation into the occupation. automated array
        iii. **SET** rows to rows+1
        iv. **Read** occupation array
        v. **LOOP** until end of file
    c. **END WHILE LOOP**

    END FUNCTION loadData()

4. **FUNCTION printData**(takes int row, int low, int medium,high)
    a. **DECLARE** double percent to 0
    b. **DISPLAY** 4 decimal point, left, setw 25- occupation name,(10) employed, (15) automation, (15) percent% , tier, newline
    c. **FOR** i = 0, til i < rows, increase i
        i. **SET** present to static_cast<double>(occupation[i].automation) / static_cast<double>(occupation[i].employed) *100;
        ii. **DISPLAY** setw(25) struct occupation.employed of i, struct of occupation.automation of i,(15))
        iii. **DISPLAY** percent, setw to (8) %.
        iv. **IF** percent >= 70
            1. **DISPLAY** High
            2. **SET** high to high +1
            3. **END IF**
        v. **ELSE IF** percent > = 30 and percent is <= 70
            1. **DISPLAY** medium
            2. **SET** medium to medium +1
            3. **END IF**
        vi. **ELSE IF** percent < 30

1. **DISPLAY** Low
2. **SET** low to low +1
3. **END IF**

d. **END FOR LOOP**

END FUNCTION printData()

1. **FUNCTION findHigh**
    a. **FOR LOOP** from k = 0 till rows, increase k++
        i. **DECLARE**present to static_cast<double>(occupation[i].automation) / static_cast<double>(occupation[i].employed) *100;
        ii. **IF** k = 0
            1. **SET** highPercent to the first percent (largest so far)
            2. **END IF**
        iii. **IF** highPercent is less or equal to percent
            a. **SET** highPercent to percent
            b. **SET i = k**
            2. **END IF**
    b. **RETURN OCCUPATION[I]**
    c. **END FOR LOOP**

END FUNCTION findHigh()

2. **FUNCTION findLow**
    a. **FOR LOOP** from k = 0 till rows, increase k++
        i. **DECLARE** percent = (nums[i][1]*100) /nums[i][0];
        ii. **IF** i = **0**
            1. **SET** lowPercent to the first percent (largest so far)
            2. **END IF**
        iii. **IF** lowPercent is greater or equal to percent
            a. **SET** lowPercent to percent
            b. **SET** i = k
            2. **END IF**
    b. **END FOR LOOP**
    c. **RETURN OCCUPATION[I]**

END FUNCTION findLow()

MAIN.CPP

5. **FUNCTION** bool openFile(ifstream &inFile)
    a. Open the text file using the file stream variable inFile
    b. IF file does not open THEN
        i. r**eturn** false
    c. ELSE

i. **Return** true

d. **END IF**

END FUNCTION openFile()

1. FUNCTION main()
   a. **DECLARE** file stream variable inFile
   b. **DECLARE** occupations as list strings
   c. **DECLARE** rows, low, medium, high, total to 0 and as integer
   d. **CALL** openFile function and output error message if it returns false
   e. **CALL** loadData()
   f. **CALL** printData ()
   g. **DISPLAY** Highest/Lowest Occupations Susceptible to Automation
   h. **SET/ CALL** highPercent = findHigh()
   i. **DISPLAY** high percent.occupation had the high share, highPercent
   j. **SET/ CALL** lowpercent = findLow()
   k. **DISPLAY** low percent.occupation, “had the lowest share” , lowPercent
   l. ////***CHALLENGE***
   m. **SET** total to low + medium + high
   n. **DISPLAY** “#Highly susceptible:”, high, (high*100)/total, “%”
   o. **DISPLAY** “#Medium susceptible: ", medium, (medium * 100)/total, “%”
   p. **DISPLAY** “#Low susceptible: " , low, (low*100)/total, “%”
   q. **RETURN** 0

END FUNCTION main()

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 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*	`FUNCTION Integer add(Integer num1, Integer num2)`

	statement END FUNCTION	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