

CS 161A: Programming and Problem Solving I

Assignment A06 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](https://draw.io) - [Diagrams.net](https://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:

zyBooks My library > CS 161B: Programming II home

zyBooks catalog Help/FAQ Zakeriya Muhumed

Search content Search View content explorer

CS 161B: Programming II
Expires Jul 3rd, 2022

View my activity
Select date and time below to show all activity up until a specific time. Default is current time.
Jun 7th, 2022 11:59 pm PDT
Download a report of my activity.

Table of contents

	zyLabs	Challenge	Participation
9. CS 161B: Functions pass by value	100%		
10. CS 161B: Functions pass by reference	100%	100%	100%
11. CS 161B: Arrays	100%	100%	100%
12. CS 161B: Char Arrays	100%	100%	100%
13. CS 161B: File Input/Output	100%		45%
13.1 File input			0%
13.2 File output			100%
13.3 File Input/Output and Functions			No activities
13.4 Extra Sample Programs			No activities
13.5 File Input/Output and Parallel Arrays			No activities
13.6 Assignment Sample and Video			No activities
13.7 LAB: All-consuming (File Input)	100%		
13.8 LAB: Cereal Analysis	100%		
14. CS 161B: Structs Part I	50%	100%	100%
15. CS 161B: Structs Part II	0%		0%
16. CS 161B: Pointers	0%		0%

Print chapter

My activity My subscription Assignments

Assigned zyLabs completion screenshot:

zyBooks My library > CS 161B: Programming II home

zyBooks catalog Help/FAQ Zakeriya Muhumed

Search content Search View content explorer

CS 161B: Programming II
Expires Jul 3rd, 2022

View my activity
Select date and time below to show all activity up until a specific time. Default is current time.
Jun 7th, 2022 11:59 pm PDT
Download a report of my activity.

Table of contents

	zyLabs	Challenge	Participation
9. CS 161B: Functions pass by value	100%		
10. CS 161B: Functions pass by reference	100%	100%	100%
11. CS 161B: Arrays	100%	100%	100%
12. CS 161B: Char Arrays	100%	100%	100%
13. CS 161B: File Input/Output	100%		45%
13.1 File input			0%
13.2 File output			100%
13.3 File Input/Output and Functions			No activities
13.4 Extra Sample Programs			No activities
13.5 File Input/Output and Parallel Arrays			No activities
13.6 Assignment Sample and Video			No activities
13.7 LAB: All-consuming (File Input)	100%		
13.8 LAB: Cereal Analysis	100%		
14. CS 161B: Structs Part I	50%	100%	100%
15. CS 161B: Structs Part II	0%		0%
16. CS 161B: Pointers	0%		0%

Print chapter

My activity My subscription Assignments

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:

In this program, you will write a program that reads in information from a file that includes total employment numbers and the number of employees susceptible for automation for each major occupation as defined by the [Bureau of Labor Statistics](#). The program will output the occupations with the highest and lowest percentages of jobs [susceptible](#) to automation (likely or liable to be influenced or harmed by automation).

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:
<ul style="list-style-type: none"> Identify and list all of the user input and their data types. <ul style="list-style-type: none"> File data.txt
<ul style="list-style-type: none"> Identify and list all of the user output and their data types. <ul style="list-style-type: none"> highPercent as double - returns the highest percentage lowPercent as double- returns the lowest percentage Tier as string- returns the percent level each occupation is in 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%
<ul style="list-style-type: none"> 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. <ul style="list-style-type: none"> int percent= (nums[i][1]*100) /nums[i][0]; Double percent = (static_cast<double>(nums[i][1])/nums[i][0]) * 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.

1. **FUNCTION** bool openFile(ifstream &inFile)

- a. **Open** the text file using the file stream variable inFile
- b. **IF** file does not open THEN
 - i. **return false**
- c. **ELSE**
 - i. **Return true**
- d. **END IF**

END FUNCTION openFile()

2. **FUNCTION loadData** (takes, file stream var, a cstring iccu, a 2dim int array for quantity and calories)

- a. **DECLARE** count as int j, i, k and set to 0
- b. **Read** the first occupation from the text file into the cstring list
- c. **WHILE** not end of file
 - i. **Read** the employed into the j column of the 2 dim num array
 - ii. **read** the automation into the k column of the 2 dim num array
 - iii. **SET** rows to rows+1
 - iv. **Read** occupation array
 - v. **LOOP** until end of file
- d. **END WHILE LOOP**

END FUNCTION loadData()

3. **FUNCTION printData**(takes occupation, 2dim int num, int row, int low, int medium, high)

- a. **DECLARE** int j, i to 0
- b. **DECLARE** char tier with max of 30
- c. **DECLARE** double percent to 0
- d. **DISPLAY** 4 decimal point, left, setw 25- occupation name, (10) employed, (15) automation, (15) percent%, tier, newline
- e. **FOR** i = 0, til i < rows, increase i
 - i. **SET** present to (static_cast<double>(nums[i][1])/nums[i][0]) * 100;
 - ii. **DISPLAY** setw(25) occupation of i, (10) nums of i and j, (15) of nums of i and k.
 - 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**

f. **END FOR LOOP**

END FUNCTION printData()

4. **FUNCTION findHigh**(takes the cstring array for occupation, a 2dim int array num, int rows, another occupation array, and double high percent)
 - a. **DECLARE** int j to 0 , k to 1 and i to 0
 - b. **FOR LOOP** from i = 0 till rows, increase i++
 - i. **DECLARE** percent = (nums[i][1]*100) /nums[i][0];
 - ii. **IF** i = 0
 1. **SET** highPercent to the first percent (largest so far)
 2. **COPY** occupations of i to occupation
 3. **END IF**
 - iii. **IF** highPercent is less or equal to percent
 - a. **SET** highPercent to percent
 - b. **COPY** occupations of i to occupation
 2. **END IF**
- c. **END FOR LOOP**

END FUNCTION findHigh()

5. **FUNCTION findLow**(takes the cstring array for occupation, a 2dim int array num, int rows, another occupation array, and double lowPercent)
 - a. **DECLARE** int j to 0 , k to 1 and i to 0
 - b. **FOR LOOP** from i = 0 till rows, increase i++
 - 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. **COPY** occupations of i to occupation
 3. **END IF**
 - iii. **IF** lowPercent is greater or equal to percent
 - a. **SET** lowPercent to percent

<ul style="list-style-type: none"> b. COPY occupations of i to occupation 2. END IF c. END FOR LOOP
END FUNCTION findHigh()
6. FUNCTION main() <ul style="list-style-type: none"> a. DECLARE file stream variable inFile b. DECLARE occupations as a list of strings (a 2dim char array) c. DECLARE occupations as list strings d. DECLARE int num as 2 dim array to store employed and automated e. DECLARE rows, low, medium, high, total to 0 and as integer f. CALL openFile function and output error message if it returns false g. CALL loadData() h. CALL printData () i. DISPLAY Highest/Lowest Occupations Susceptible to Automation j. CALL findHigh() k. DISPLAY char array occupation had the high share, int highPercent l. CALL findLow() m. DISPLAY char array occupation, "had the lowest share" , int lowPercent n. ////CHALLENGE o. SET total to low + medium + high p. DISPLAY "#Highly susceptible:", high, (high*100)/total, "%" q. DISPLAY "#Medium susceptible: ", medium, (medium * 100)/total, "%" r. DISPLAY "#Low susceptible: " , low, (low*100)/total, "%" s. 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	<pre>IF <i>condition</i> THEN <i>statement</i> <i>statement</i> END IF</pre>	<pre>IF num_dogs > 10 THEN DISPLAY "That is a lot of dogs!" END IF</pre>
Use a dual alternative conditional	<pre>IF <i>condition</i> THEN <i>statement</i> <i>statement</i> ELSE <i>statement</i> <i>statement</i> END IF</pre>	<pre>IF num_dogs > 10 THEN DISPLAY "You have more than 10 dogs!" ELSE DISPLAY "You have ten or fewer dogs!" END IF</pre>
Use a switch/case statement	<pre>SELECT <i>variable or expression</i> CASE <i>value_1</i>: <i>statement</i> <i>statement</i> CASE <i>value_2</i>: <i>statement</i> <i>statement</i> CASE <i>value_2</i>: <i>statement</i> <i>statement</i> DEFAULT: <i>statement</i> <i>statement</i> END SELECT</pre>	<pre>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</pre>
Loops		
Loop while a condition is true - the loop body will execute 0 or more times.	<pre>WHILE <i>condition</i> <i>statement</i> <i>statement</i> END WHILE</pre>	<pre>SET num_dogs = 1 WHILE num_dogs < 10 DISPLAY num_dogs, " dogs!" SET num_dogs = num_dogs + 1 END WHILE</pre>
Loop while a condition is true - the loop body will execute 1 or more times.	<pre>DO <i>statement</i> <i>statement</i> WHILE <i>condition</i></pre>	<pre>SET num_dogs = 1 DO DISPLAY num_dogs, " dogs!" SET num_dogs = num_dogs + 1 WHILE num_dogs < 10</pre>
Loop a specific number of times.	<pre>FOR <i>counter</i> = <i>start</i> TO <i>end</i> <i>statement</i> <i>statement</i> END FOR</pre>	<pre>FOR count = 1 TO 10 DISPLAY num_dogs, " dogs!" END FOR</pre>
Functions		
Create a function	<pre>FUNCTION <i>return_type</i> <i>name (parameters)</i> <i>statement</i></pre>	<pre>FUNCTION Integer add(Integer num1, Integer num2)</pre>

	<i>statement</i> END FUNCTION	DECLARE Integer sum SET sum = num1 + num2 RETURN sum END FUNCTION
Call a function	CALL <i>function_name</i>	CALL add(2, 3)
Return data from a function	RETURN <i>value</i>	RETURN 2 + 3