CS 161A/B: Programming and Problem Solving I

Assignment 1 - 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:

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

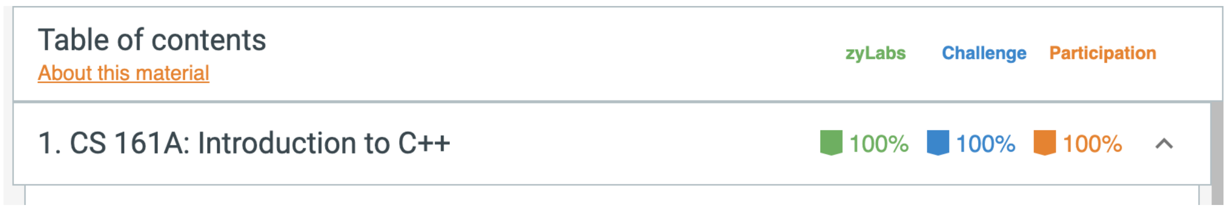
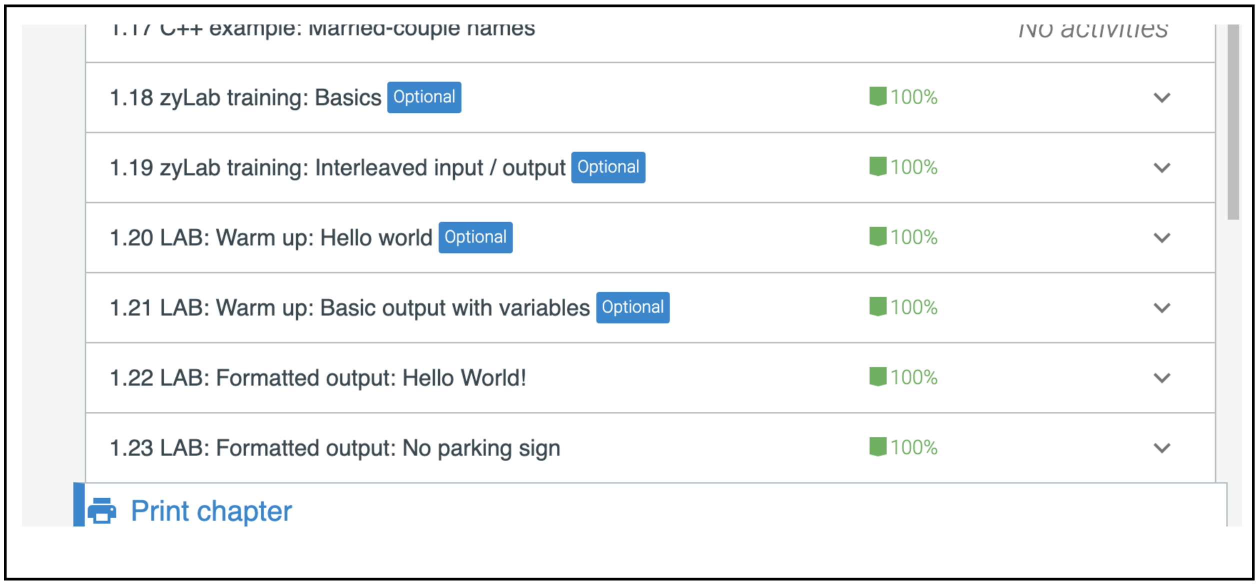
Challenge and Participation % screenshot:


Table of contents
About this material

	zyLabs	Challenge	Participation
1. CS 161A: Introduction to C++	🟩 100%	🟦 100%	🟧 100%

Assigned zyLabs completion screenshot:



1.17 C++ example: Married-couple names — No activities

1.18 zyLab training: Basics Optional — 🟩100%

1.19 zyLab training: Interleaved input / output Optional — 🟩100%

1.20 LAB: Warm up: Hello world Optional — 🟩100%

1.21 LAB: Warm up: Basic output with variables Optional — 🟩100%

1.22 LAB: Formatted output: Hello World! — 🟩100%

1.23 LAB: Formatted output: No parking sign — 🟩100%

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 calculate the number of pizzas needed and total cost for a Pizza Party. The user will tell the program how many people are expected at the pizza party, and the program will tell the user how many slices of pizza and what the cost will be.

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:
"Welcome to the Pizza Party Pizza Counting generator! Let's start with the number of people expected for the Pizza Party. Please input a number:" "Thanks for telling me you will have <integer> people at the pizza party!"

“The amount of pizzas you will need are <integer>, and the cost will be <integer>. Have a great party!”

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).
pizzaPartyHeadCount <input> INTEGER: ● The user’s number input for how many people are attending the party in integer format.
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).
DECLARE VARIABLE, integer: **pizzaPerPersonServing**: This is the estimated amount of 2 slices of pizza per person that was given in the assignment directions. **DECLARE** VARIABLE, double: **pizzaCostPerPerson**: This is the cost of slices per person.

<<user input & setting of pizzaPartyHeadCount variable happens here>>

SET VARIABLE, double: **pizzaPartyCost**: This is the variable for the cost of the pizza aka pizzaCostPerPerson, which has a data type as double because the answer will be in currency.)

SET VARIABLE, integer: **pizzaPartyPies**: (This is the variable for the number of pizzas, aka pizzaPies, which has a data type of integer because the answer will be rounded to a whole number

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.

pizzaPartyHeadCount = <user input integer>

OUTPUT: **pizzaPerPersonServing =** 12 (*slices in a whole pizza*) / 2 (*slices per person)*

OUTPUT: **pizzaCostPerPerson =** $14.95 / pizzaPerPersonServing

<<user input & setting of pizzaPartyHeadCount variable happens here>>

OUTPUT: **pizzaPartyCost =** pizzaPartyHeadCount * pizzaCostPerPerson

OUTPUT: **pizzaPartyPies =** pizzaPartyHeadCount * pizzaPerPersonServing

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.

1. DECLARE: The program starts and the variables for cost and slice amount per person on average are declared.

2. OUTPUT: Welcome the user and ask them how many people are expected at the pizza party. (String)
3. INPUT: The user inputs a number of people in integers. (Integer)
4. SET: The program sets the variable for head count (Integer)
5. DECLARE: The amount of slices of pizza needed for the party in the new variable (Integer), and the cost of the pizza party in the new variable (Double).
6. OUTPUT: The cost of the pizza party and slices/pies needed are output here. (String with variables storing integers)

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:*
		`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`

	statement
	    statement
	  CASE *value_2:*
	    statement
	    statement
	  DEFAULT:
	    statement
	    statement
	END SELECT
		`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`