CS 161A: Programming and Problem Solving I

Assignment A05 Sample 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.

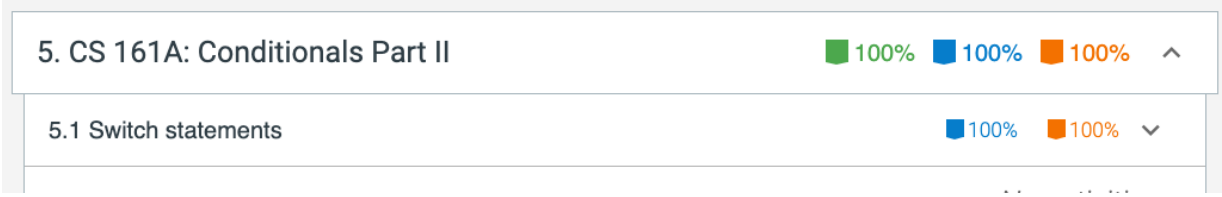
Planning your program before you start coding is part of the development process. In this document you will:

- [x] ~~Paste a screenshot of your zyBooks Challenge and Participation %~~
- [x] ~~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:

This program reads two phrases on separate lines, and outputs one of these 4 responses:

- Phrase one is found within phrase two
- Phrase two is found within phrase one
- Both phrases match
- No matches

The user will input the two phrases upon request from the program, and will exercise the find() and substr() functions, and string::npos constant.

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 Gina's Phrase Scrambler!

We are going to scramble 2 phrases for you! Please enter phrase 1:
truck
```

```
You entered: truck

Please enter phrase 2:
firetruck is here
You entered: firetruck is here

truck is found in firetruck is here
truck is here

Thank you for using Gina's Phrase Scrambler! Have a great day!
```

```
Welcome to Gina's Phrase Scrambler!
We are going to scramble 2 phrases for you! Please enter phrase 1:
the green grass grows
You entered: the green grass grows

Please enter phrase 2:
green grass
You entered: green grass

green grass is found in the green grass grows
green grass grows
Thank you for using Gina's Phrase Scrambler! Have a great day!
```

```
Welcome to Gina's Phrase Scrambler!

We are going to scramble 2 phrases for you! Please enter phrase 1:
He was between a rock and a hard place
You entered: He was between a rock and a hard place

Please enter phrase 2:
rock
You entered: rock

rock is found in He was between a rock and a hard place
rock and a hard place

Thank you for using Gina's Phrase Scrambler! Have a great day!
```

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:

a. Identify and list all of the user input and their data types.

- phrase1 (string)
- phrase2 (string)

b. Identify and list all of the user output and their data types.

- position (size_t)

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.

- no math calculations
- Listing the if else statements in the pseudocode

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.

1. DECLARE variable phrase1 as string
2. DECLARE variable phrase2 as string
3. DECLARE variable position as size_t
4. DISPLAY welcome message
5. DISPLAY a new line to separate text
6. DISPLAY prompt to enter phrase 1
7. INPUT phrase1 (using getline)
8. DISPLAY the You Entered: message
9. DISPLAY a new line to separate text
10. DISPLAY prompt to enter phrase 2
11. INPUT phrase2 (using getline)
12. DISPLAY the You Entered: message
13. DISPLAY a new line to separate text
14. SET position variable to phrase2.find(phrase1) → searches Phrase2 for the beginning of phrase1
15. IF position is NOT string::npos
    a. THEN
        i. DISPLAY phrase1 + " is found in " + phrase2
        ii. DISPLAY phrase2.subtr(position) → creating a substring from phrase2
    b. ELSE
        i. position = phrase1.find(phrase2)

ii. IF position is NOT string::npos
    1. DISPLAY phrase2 + " is found in " + phrase1
    2. DISPLAY phrase1.substr(position)
    3. ELSE
        a. IF phrase1 == phrase2
            i. DISPLAY "Both phrases match"
        b. ELSE
            i. DISLAY "No match"
        c. END IF
    4. END IF
iii. END IF

16. DISPLAY a new line to separate text
17. DISPLAY thank you/exit message
18. END program

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
	  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`