CS 161A/B: Programming and Problem Solving I

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.

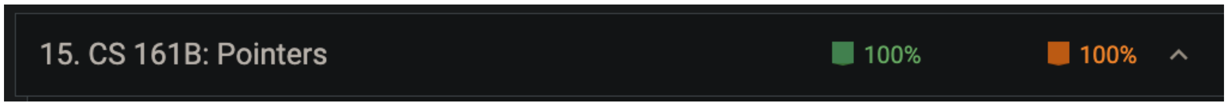
This document contains an interactive checklist. To mark an item as complete, click on the box (the entire list will be highlighted), then right click (the clicked box will only be highlighted), and choose the checkmark.

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

- ❏ 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:
 15. CS 161B: Pointers 🟩 100% 🟧 100%

Assigned zyLabs completion screenshot:

15.4 Receiving arrays as pointers in Functions — No activities

15.5 LAB: Fun with Pointers — 100%

15.6 LAB: Quarterly Sales — 100%

15.7 LAB: Find maximum — 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:

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:

```
Enter integer 1: 3
Enter integer 2: 17

Before call to swapArgs a: 3 b: 17
After call to swapArgs a: 17 b: 3
After call to divideArgs a: 5 b: 2
After call to powerArgs a: 25 b: 2

Goodbye!
```

```
Enter integer 1: 2
Enter integer 2: 10
```

```
Before call to swapArgs a: 2 b: 10
After call to swapArgs a: 10 b: 2
After call to divideArgs a: 5 b: 0
After call to powerArgs a: 1 b: 0

Goodbye!
```

```
Enter integer 1: -10
Enter integer 2: 3

Before call to swapArgs a: -10 b: 3
After call to swapArgs a: 3 b: -10
After call to divideArgs a: 0 b: 3
After call to powerArgs a: 0 b: 3

Goodbye!
```

```
Enter integer 1: 0
Enter integer 2: 0

No operations performed!
```

```
Enter integer 1: -2
Enter integer 2: -9

Before call to swapArgs a: -2 b: -9
After call to swapArgs a: -9 b: -2
After call to divideArgs a: 4 b: -1
After call to powerArgs a: 4 b: -1

Goodbye!
```

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).

a (int): The first integer input by the user. b (int): The second integer input by the user. swapArgs(int*, int*): Swaps the values of the two integer pointers. divideArgs(int*, int*): Calculates the quotient and remainder of the first integer divided by the second, updating the first integer to the quotient and the second to the remainder. powerArgs(int*, int*): Raises the first integer to the power of the second integer and stores the result in the first integer.

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).

*a and *b keep outputting in different formats using the 3 functions

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.

tempQuotient = *a / *b

tempRemainder = *a % *b (modulo)

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.

```
CREATE FUNCTION void swapArgs(int *a, int *b)
CREATE FUNCTION void divideArgs(int *a, int *b)
CREATE FUNCTION void powerArgs(int *a, int *b)
CREATE integer FUNCTION main()

   DECLARE integer a, SET to 0
   DECLARE integer b, SET to 0

   DISPLAY “Enter integer 1: “
   INPUT a
   DISPLAY “Enter integer 2: “
   INPUT b

   IF a AND b = 0
      DISPLAY “No operations performed!”
      end program
   END IF

   DISPLAY swapArgs prompt before + a + b
   CALL swapArgs (&a, &b)
   DISPLAY after swapArgs prompt + a + b

   CALL divideArgs(&a, &b)
   DISPLAY after divideArgs prompt + a + b

   CALL powerArgs (&a, &b)
   DISPLAY after powerArgs prompt + a + b

   DISPLAY goodbye!
   END PROGRAM

CALL FUNCTION void swapArgs( int *, int *)
   // DO NOT USE pow() function, use a loop to calculate the result
   // any number raised to the 0 power is 1
   // if the power is a negative number, do not calculate any result
   DECLARE int temp and SET to 0
   SET temp = a (dereferenced)
   SET a (dereferenced) to b (dereferenced)
   SET b (dereferenced) to temp
END swapArgs FUNCTION

CALL FUNCTION void divideArgs (int *a, int *b)
   DECLARE tempQuotient, SET to 0
   DECLARE tempRemainder, SET to 0

   IF b (deref’d) is NOT 0
      SET tempQuotient = a (deref’d) divided by b (deref’d)
      SET tempRemainder = a (deref’d) modulo b (deref’d)
      SET a (deref’d) to tempQuotient
      SET b (deref’d) to tempRemainder
   END IF
END divideArgs FUNCTION
```

```
CALL FUNCTION void powerArgs(int *a, int *b)
    DECLARE i, and SET to 0
    DECLARE result, and SET to 0
    IF b (deref'd) is less than 0
        return to program
    ELSE IF b (deref'd) is 0
        SET result = 1
        FOR LOOP (i = 0; i less than b (deref'd); i++)
            SET result = results * a (deref'd)
        END FOR LOOP
       SET a (deref'd) to result
    END IF
END FUNCTION powerArgs
```

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
		`SELECT num_dogs`
		`   CASE 0: DISPLAY "No dogs!"`
		`   CASE 1: DISPLAY "One dog.."`
		`   CASE 2: DISPLAY "Two dogs.."`
		`   CASE 3: DISPLAY "Three dogs.."`

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