CS 161A: Programming and Problem Solving I

Discussion 3 Algorithmic Design Document

Corinne Fargo & Gina Ferguson

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

- 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

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 perform simple arithmetic, which can be used as a math tutor for a young student. It will take two random numbers, add them together, and display it in an intuitive math expression.

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 algorithm. Calculate and show the expected outputs. Use the sample run to test your algorithm.

Sample run:
What is the answer? Press enter to display! 345 + 543 --------

Press enter when ready for the answer!

<user presses enter>

345

+ 543

888

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.
An enter/procession
b. Identify and list all of the user output and their data types.
randNum1 (int) a random number generated by the program randNum2 (int) a second random number generated by the program sumNum (int) the sum of two randomly generated numbers
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.
sumNum = randNum1 + randNum2
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.
DECLARE/include <iostream> <cstdlib><cmath> <ctime> DECLARE randNum1 DECLARE randNum2 DECLARE sumNum DECLARE/SET random time statement INPUT/generate random number randNum1, using range of 0-999 INPUT/generate random number randNum2, using range of 0-999 SET sumNum = randNum1 + randNum2 DISPLAY “randNum1 + randNum2 =” formatted DISPLAY “Press enter when ready for the answer!” DISPLAY/declare cin.ignore/cin.get function to get user enter/procession DISPLAY “randNum1 + randNum2 = sumNum” formatted

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	INPUT	`INPUT num_dogs`

into a variable		
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`