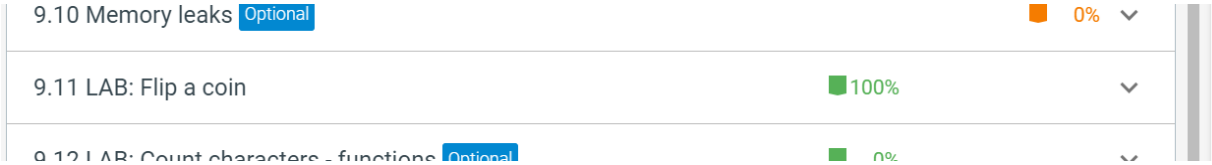
Assignment xx Algorithmic Design Document

Make a copy before you begin (File -> Make a copy). Add the Assignment # above and complete the sections below BEFORE you begin to code and submit with your Assignment to D2L (File -> Download -> PDF). The sections will expand as you type.

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%.

zyLabs, Challenge, and Participation % Screenshot:


Assigned zyLabs completion Screenshot:


Assignment

Program description:
This program will take any two numbers you enter, swap the values, and then display them.

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 all of the user input. What are the data types of the inputs? Define the input variables.
Input num1 as Int Input num2 as Int
b. Describe the program output. What is displayed to the user? What are the data types of the output? Define the output variables.

The progrma will output strings:
- - Welcome!
- - Please enter 2 numbers
- - You entered num1, num2
- - Swapped, these numbers are now num1, num2
- - Thank you for using my program!

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 calculations are performed.

d. Design the logic of your program using pseudocode or flowcharts. See pseudocode syntax at the bottom of this document. Here is where you would use conditionals, loops, functions or array constructs (if applicable) and list the steps in transforming inputs into outputs. Walk through your logic steps with the test data from the assignment document.

```
START

Struct Number_struct as Number
- - DECLARE Int num

Function Swap(Number* numPtr1, Number* numPtr2)
- - DECLARE Int tempData1
- - SET tempData1 = numPtr1->num
- - DECLARE Int tempData2
- - SET tempData2 = numPtr2->num
- -
- - SET numPtr1->num = tempData2
- - SET numPtr2->num = tempData1

Function main(void)
- - DECLARE Number* num1
- - DECLARE Number* num2
- -
- - DISPLAY "Welcome to my Swapping Numbers Program!\n"
- -
- - DISPLAY "\nEnter 2 numbers and I will swap them for you!!\n"
- - INPUT num1, num2
- -
- - DISPLAY "\nYou entered:    num1 = %d and num2 = %d\n", num1, num2
- -
- - CALL Swap(&num1, &num2)
- -
- - DISPLAY "After swapping: num1 = %d and num2 = %d\n", num1, num2
- -
- - DISPLAY "\nThank you for using my program!\n"
- -
```

- - RETURN 0

END

e. Include 2 Sample Program Runs for your program using your own set of data. This data set must be different from my Sample Runs in the Assignment document. This process is similar to Unit Testing and will help you test your program better.

Sample Program Run 1:

```
Microsoft Visual Studio Debu

Welcome to my Swapping Numbers Program!

Enter 2 numbers and I will swap them for you!!
5 10

You entered:    num1 = 5 and num2 = 10
After swapping: num1 = 10 and num2 = 5

Thank you for using my program!
```

Sample Program Run 2:

```
Microsoft Visual Studio Debu

Welcome to my Swapping Numbers Program!

Enter 2 numbers and I will swap them for you!!
6 14

You entered:    num1 = 6 and num2 = 14
After swapping: num1 = 14 and num2 = 6

Thank you for using my program!
```

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`