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:

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:

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 my Citizen's Database!


Here is your list so far:

Gayathri;USA;22
Stephanie;USA;27
Priya;India;34
Ahmed;Nigeria;52

Enter your name: Navid
Enter your citizenship: Ecuador
Enter your age (whole #s only): 34
Enter position number: 2

After adding a person, the list is:
Gayathri;USA;22
Stephanie;USA;27
Navid;Ecuador;34
Priya;India;34
Ahmed;Nigeria;52


*~*~*~ Thank you for using my Citizen Database!! ~*~*~*
```

```
Welcome to my Citizen's Database!
```

```
Here is your list so far:

Gayathri;USA;22
Stephanie;USA;27
Priya;India;34
Ahmed;Nigeria;52

Enter your name: Navid
Enter your citizenship: Ecuador
Enter your age (whole #s only): 34
Enter position number: 9

Error! Invalid position.

Failed to add a person.


*~*~*~ Thank you for using my Citizen Database!! ~*~*~*
```

```
Welcome to my Citizen's Database!


Here is your list so far:

Gayathri;USA;22
Stephanie;USA;27
Priya;India;34
Ahmed;Nigeria;52

Enter your name: Lucy
Enter your citizenship: Ecuador
Enter your age (whole #s only): 34
Enter position number: 0

After adding a person, the list is:

Lucy;Ecuador;34
Gayathri;USA;22
Stephanie;USA;27
Priya;India;34
Ahmed;Nigeria;52


*~*~*~ Thank you for using my Citizen Database!! ~*~*~*
```

```
Welcome to my Citizen's Database!
```

```
Here is your list so far:

Gayathri;USA;22
Stephanie;USA;27
Priya;India;34
Ahmed;Nigeria;52

Enter your name: Arely
Enter your citizenship: Mexico
Enter your age (whole #s only): 45
Enter position number: 4

After adding a person, the list is:

Gayathri;USA;22
Stephanie;USA;27
Priya;India;34
Ahmed;Nigeria;52
Arely;Mexico;45


*~*~*~ Thank you for using my Citizen Database!! ~*~*~*
```

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).
see comments in code

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).

see header comment and function comments

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.

none

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.

```
—---------- entire program flow, not sorted by file —----------------
// Main program flow
FUNCTION main RETURNS Integer
    DECLARE PersonType list[CAPACITY]
    DECLARE Integer count = 0
    DECLARE String fileName = "persons.txt"

    CALL welcome
    CALL populatePersons(list, count, fileName)
    DISPLAY "\nHere is your list so far: \n\n"
    CALL printPersons(list, count)
    IF CALL addPerson(list, count) THEN
        DISPLAY "\nAfter adding a person, the list is: \n\n"
        CALL printPersons(list, count)
    ELSE
        DISPLAY "\nFailed to add a person.\n"
    END IF
    CALL goodbye
    RETURN 0
END FUNCTION

// Function prototypes
```

```
FUNCTION void populatePersons(PersonType list[], Integer &count, String
fileName)
FUNCTION void printPersons(const PersonType list[], Integer count)
FUNCTION void welcome()
FUNCTION Boolean addPerson(PersonType list[], Integer &count)
FUNCTION void goodbye()
FUNCTION void readInt(String prompt, Integer &age)
FUNCTION Boolean containsDigit(String str)

// populatePersons function
FUNCTION populatePersons(PersonType list[], Integer &count, String
fileName)
    DECLARE ifstream inFile
    DECLARE String name, citizen
    DECLARE Integer age

    inFile.open(fileName)
    IF NOT inFile THEN
        DISPLAY "\nFail to open ", fileName, " to populate inventory!\n"
        EXIT PROGRAM
    END IF

    LOOP UNTIL inFile.eof()
        INPUT inFile >> name >> citizen >> age
        list[count].name = name
        list[count].citizenship = citizen
        list[count].age = age
        INCREMENT count
    END LOOP

    inFile.close()
END FUNCTION

// printPersons function
FUNCTION printPersons(const PersonType list[], Integer count)
    FOR Integer index = 0 TO count - 1
        DISPLAY list[index].name, ";", list[index].citizenship, ";",
list[index].age, "\n"
    END FOR
END FUNCTION

// welcome function
FUNCTION welcome()
    DISPLAY "\nWelcome to my Citizen's Database!\n\n"
END FUNCTION

// addPerson function
FUNCTION Boolean addPerson(PersonType list[], Integer &count) RETURNS
Boolean
```

```
    DECLARE Integer position, i, tempAge
    DECLARE PersonType aPerson

    IF count >= CAPACITY THEN
        DISPLAY "\nError! The list is full.\n"
        RETURN False
    END IF

    DISPLAY "\nEnter your name: "
    INPUT aPerson.name
    IF CALL containsDigit(aPerson.name) THEN
        DISPLAY "\nError! Name cannot contain numbers.\n"
        RETURN False
    END IF

    DISPLAY "Enter your citizenship: "
    INPUT aPerson.citizenship
    IF CALL containsDigit(aPerson.citizenship) THEN
        DISPLAY "\nError! Citizenship cannot contain numbers.\n"
        RETURN False
    END IF

    CALL readInt("Enter your age (whole #s only): ", tempAge)
    SET aPerson.age = tempAge

    IF aPerson.age < 1 OR aPerson.age > 100 THEN
        DISPLAY "\nError! Invalid age.\n"
        RETURN False
    END IF

    DISPLAY "Enter position number: "
    INPUT position
    IF position < 0 OR position > count THEN
        DISPLAY "\nError! Invalid position.\n"
        RETURN False
    END IF

    FOR i = count TO position STEP -1
        list[i] = list[i-1]
    END FOR

    list[position] = aPerson
    INCREMENT count
    RETURN True
END FUNCTION

// goodbye function
FUNCTION goodbye()
```

```
    DISPLAY "\n\n*~*~*~ Thank you for using my Citizen Database!!
~*~*~*\n\n"
END FUNCTION

// readInt function
FUNCTION readInt(String prompt, Integer &age)
    DISPLAY prompt
    WHILE NOT (INPUT age)
        DISPLAY "\nInvalid input. Please enter a valid number: "
        CLEAR INPUT BUFFER
    END WHILE
END FUNCTION

// containsDigit function
FUNCTION Boolean containsDigit(String str) RETURNS Boolean
    FOR EACH CHARACTER ch IN str
        IF IS DIGIT(ch) THEN
            RETURN True
        END IF
    END FOR
    RETURN False
END FUNCTION
```

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`