CS 162: Computer Science II

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.

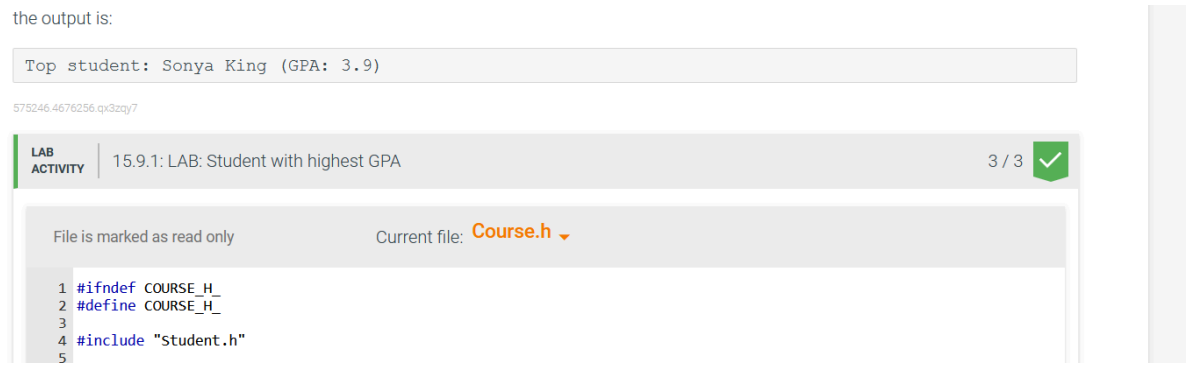
Planning your program before you start coding is part of the development process. In this document you will:

- ❏ 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

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 will load task data from a file and present a menu to the user. The user can then perform a number of actions on the data. When the user exits, the data will be saved to the file.

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.
Do not simply copy the sample run from the assignment instructions!

Sample run:

```
Welcome!
This program will help you manage your tasks for this Space
Station.

Pick an option from below:

(a)Add a new task
(b)List tasks by name
(c)Remove tasks by index
(q)Quit

b
1. 28;Replace ventilation filters;2;Carlos Johnston;Maintenance
2. 15;Data Transmission and Storage;4;Carlos Johnston;Communications
```

3. 9;Adjust fuel rods;3;Gail Lawhead;Operations

4. 14;Replace ventilation ion filters;2;Karma Thames;Maintenance

5. 7;Computer system diagnostic;2;Kelli Weeberly;Communications

6. 13;Hull debris sweep;4;Richard Moody;Operations

7. 7;Food system checkup;2;Richard Moody;Inventory

8. 5;Replace ventilation filters;2;Robbie Mitchell;Maintenance

```
Pick an option from below:

(a)Add a new task
(b)List tasks by name
(c)Remove tasks by index
(q)Quit

p
Invalid Option!

Pick an option from below:

(a)Add a new task
(b)List tasks by name
(c)Remove tasks by index
(q)Quit

a
Enter the day of the task (whole numbers between 1 and 30): 78
Invalid Entry! Must be between 1 and 30 inclusive!
Enter the day of the task (whole numbers between 1 and 30): 8
Enter the task name (50 characters or less): Surface static
discharge
Enter the person's name (50 characters or less): Steph Kalias
Enter the number of hours (whole numbers between 1 and 10): 3
Enter the task type 0-Operations, 1-Maintenance, 2-Inventory,
3-Communications, and 4-Others): 0

Task added!
```

1. 28;Replace ventilation filters;2;Carlos Johnston;Maintenance

2. 15;Data Transmission and Storage;4;Carlos Johnston;Communications

3. 9;Adjust fuel rods;3;Gail Lawhead;Operations

4. 14;Replace ventilation ion filters;2;Karma Thames;Maintenance

5. 7;Computer system diagnostic;2;Kelli Weeberly;Communications

6. 13;Hull debris sweep;4;Richard Moody;Operations

7. 7;Food system checkup;2;Richard Moody;Inventory

8. 5;Replace ventilation filters;2;Robbie Mitchell;Maintenance

9. 8;Surface static discharge;Steph Kalias;Operations

```
Pick an option from below:

(a)Add a new task
(b)List tasks by name
(c)Remove tasks by index
(q)Quit

c
```

1. 28;Replace ventilation filters;2;Carlos Johnston;Maintenance

2. 15;Data Transmission and Storage;4;Carlos Johnston;Communications

3. 9;Adjust fuel rods;3;Gail Lawhead;Operations

4. 14;Replace ventilation ion filters;2;Karma Thames;Maintenance

5. 7;Computer system diagnostic;2;Kelli Weeberly;Communications

6. 13;Hull debris sweep;4;Richard Moody;Operations

7. 7;Food system checkup;2;Richard Moody;Inventory

8. 5;Replace ventilation filters;2;Robbie Mitchell;Maintenance

9. 8;Surface static discharge;Steph Kalias;Operations

```
Enter index of task to remove: 9
Task Removed!
```

1. 28;Replace ventilation filters;2;Carlos Johnston;Maintenance

2. 15;Data Transmission and Storage;4;Carlos Johnston;Communications

3. 9;Adjust fuel rods;3;Gail Lawhead;Operations

4. 14;Replace ventilation ion filters;2;Karma Thames;Maintenance

5. 7;Computer system diagnostic;2;Kelli Weeberly;Communications

6. 13;Hull debris sweep;4;Richard Moody;Operations

```
7. 7;Food system checkup;2;Richard Moody;Inventory
8. 5;Replace ventilation filters;2;Robbie Mitchell;Maintenance

Pick an option from below:

(a)Add a new task
(b)List tasks by name
(c)Remove tasks by index
(q)Quit

q

Tasks written to file! Thank you for using my program!!
```

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 variables 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”.
● char Selection - The menu selection the user chooses ● // Add task ○ int day - The day for the task, 1 - 30 ○ string name - The task name, 50 Char max ○ string personName - The person assigned to the task, 50 char max ○ int duration - The duration of the task, 1 - 10 ○ int category - The category of the task, 0 - 4

- // Remove Task
    - int index - The index to be removed
- // List tasks by type
    - int category - The category to filter by, 0 - 4
- // List tasks by name
    - string filter - The string to filter the task names by

b. Identify and list all of the user output variables 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”.

- int day - The day of the task
- int duration - The duration of the task
- string name - The name of the task
- string personName - The person assigned to the task
- string category - The category name of the task

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.

No calculations

d. Design the logic of your program using pseudocode. 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. Do not include any implementation details (e.g. file names) or C++ specific syntax.

```
DECLARE MAX_LIST_SIZE = 30
DECLARE MAX_CHAR = 51 // 50 +1 for null byte
DECLARE Task Array taskList
DECLARE int size = 0

// Load File
DECLARE Task newTask
DO
   INPUT INTO newTask from task.txt
```

```
   DECLARE int index = 0
   WHILE index < MAX_LIST_SIZE AND taskList[index].personName < task.personName
      SET index = index + 1
   END WHILE
   FOR int i = 0, i > index; i–
      taskList[i] = taskList[i - 1]
   END FOR
   taskList[index] = newTask
   SET size = size + 1
WHILE file is not eof and index < MAX_LIST_SIZE

// Menu Loop
bool exitSig = false
DECLARE char selection
DO
   DISPLAY menu
   INPUT into selection
   SWITCH (selection)
      case ‘a’ // Add Task
         DECLARE Task newTask
         INPUT INTO newTask
         DECLARE int index = 0
         WHILE index < MAX_LIST_SIZE AND taskList[index].personName <
task.personName
            SET index = index + 1
         END WHILE
         FOR int i = 0, i > index; i–
            taskList[i] = taskList[i - 1]
         END FOR
         taskList[index] = newTask
         SET size = size + 1
      case ‘b’ // List Task
         FOR int i = 0; i < size; i++
            DISPLAY taskList[i]
         END FOR
      case ‘c’ // Remove Task
         DECLARE int index
         INPUT INTO index
         FOR int i = index; i < MAX_LIST_SIZE - 1; i++
            SET taskList[i] = taskList[i + 1];
         ENDFOR
         SET taskList[MAX_LIST_SIZE - 1] = empty struct;
         SET size = size - 1;
```

```
        case ‘d’ // List Task By Type
            DECLARE TaskType filter
            INPUT into filter
            FOR int i = 0; i < size; i++
                IF taskList[i] == filter
                    DISPLAY taskList[i]
                ENDIF
            END FOR
        case ‘e’ // List Task By Name
            DECLARE string filter
            INPUT into filter
            FOR int i = 0; i < size; i++
                IF filter in taskList[i].name
                    DISPLAY taskList[i]
                ENDIF
            END FOR
        case ‘q’ // quit
            exitSig = true
WHILE !exitSig

// Load File
FOR int i = 0; i < size; i++
    SAVE taskList[i] TO FILE
ENDFOR
```

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	IF *condition* THEN	`IF num_dogs > 10 THEN`

conditional	*statement* *statement* END IF	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`