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:
This program entails a queue of 5 numbers in an array using a FIFO structure. The front of the queue starts with just 3 numbers, and as we add a number to the rear, one-at-a-time, the values in the array shift toward the front of the queue.

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 the FIFO Queue Program!

Enter option: +
Integer: 9
[9]

Enter option: +
Integer: 3
[9, 3]

Enter option: +
Integer: -2
[9, 3, -2]

Enter option: +
Integer: 10
Error: Queue Overflow!
[9, 3, -2]

Enter option: -
Integer: 5
5 is not in the queue.
[9, 3, -2]

Enter option: -
Integer: 3
[-2]
```

```
Enter option: @
Invalid option.

Enter option: p
[-2]

Enter option: -
Integer: -2
-2 is not in the queue.
[]

Enter option: -
Integer: 0
Queue Empty.
[]

Enter option: +
Integer: 23
[23]

Enter option: q

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).
option (char): one character that stores user choice (+, -, p, or q) **val** (int): represents the value to move throughout the queue

queue[]: array that stores the queue data structure w/ a MAX size of 5

size (int): this stores the num of elements in the array (queue in this case) and passed by reference. (0 - MAX in size)

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).

cout string for messages throughout the code

queue[i] (int): this stores the index of the queue in the printQueue()

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.

queue[size] = val; size++; → when a value enters the queue, it is assigned to the element in the array at the index that is equal to the size of the queue. Then the size is iterated/incremented by 1.

Many if else statements and loops are used for longer calculating of the queue.

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.

```
DECLARE #include <iostream>, using namespace std

FUNCTION integer
   enqueue (int queue[], int &size, int val);
END FUNCTION

FUNCTION integer
   dequeue (int queue[], int &size, int &val);
END FUNCTION
```

```
FUNCTION void
    void printQueue(int queue[], int size);
END FUNCTION

DECLARE constant integer MAX = 3

FUNCTION int main()
DECLARE queue[MAX]
DECLARE (integer) size = 0
DECLARE (char) option = ‘ ‘
DECLARE (integer) val = 0

DISPLAY welcome message

DO LOOP START
   DISPLAY “Enter option: “
    INPUT option
    IF (#1) option = +
        DISPLAY “Integer: “
        INPUT val
        IF (#2) FUNCTION enqueue () = 0
                 CALL printQueue()
        ELSE
             DISPLAY Error: Queue overflow message
             CALL printQueue()
        END IF (#2)
    ELSE IF option = -
        DISPLAY “Integer: “
        INPUT val
        IF (#3) dequeue() = 0
               CALL printQueue(
        ELSE IF dequeue() = 1
              DISPLAY “Queue Empty.”
        ELSE
              DISPLAY val + “ is not in the queue.”
              CALL printQueue()
       END IF (#3)
    ELSE IF option = p
        CALL printQueue()
    ELSE IF option = q
        DISPLAY “Goodbye!”
    ELSE
        DISPLAY “Invalid option.”
    END IF (#1)
WHILE option is NOT q
return 0
END DO WHILE LOOP
END MAIN FUNCTION

CALL enqueue(int queue[], int &size, int val)
     IF size < MAX
            queue[size] = val
            size++
            return 0
        ELSE
            return 1 (overflow)
        END IF
END enqueue FUNCTION

CALL dequeue(int queue[], int &size, int &val)
    DECLARE int index = -1
    DECLARE int  i = 0

    IF (#1) size > 0

        FOR (#1) (i = 0; i < size; i++)

               IF (#2)  queue[i] = val
                     SET index = i
                     BREAK
               END IF (#2)

        IF (#3)  index is NOT -1

              FOR (#2) i = 0; i < index, i++
                     SET queue[i] = 0
              END FOR (#2)

              FOR (#3)  i = index; i < size - 1; i++
                 SET queue[i] = queue[i+1]
              END FOR (#3)

              SET queue[size - 1] = 0
              SET size - index - 1
               RETURN 0 → success 🙂
        ELSE (for IF #3)
```

```
            RETURN 2 → not found 🙁
        END IF (#3)

    ELSE (#1 IF)

        RETURN 1 → underflow 🧐
END IF (#1)
END dequeue FUNCTION

CALL void printQueue(int queue[], int size)
    int i = 0

    DISPLAY "["
    FOR i = 0; i < size; i++
        DISPLAY queue[i]
        IF i is NOT size -1
            DISPLAY ", "
       END IF
     END FOR
     DISPLAY "]"
END printqueue 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	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`