CIS11 Course Project Part 1: Documenting the Project

Fill in the following areas (purple).

Introduction

1.1 Purpose

The objective of this program is to rearrange 8 user inputted numbers and sort them by numerical value.

1.2 Intended Audience and Users

The primary user of the program could be those who simply want to rearrange a given set of numbers by numerical value.

1.3 Product Scope

The intentions of this program is to rearrange a given set of 8 numbers in numerical order by using bubble sort method.

1.4 Reference

Source Documents for the Program Requirements and Specification

Reference Project requirements and LC-3 specifications.

Program:

Create an LC-3 program that implements Bubble Sort for the following values:

Input: User input 8 numbers, ranging from 0 – 100.

Output: Display sorted values in ascending order in console.

The program must fulfill the following criteria:

1. Contain appropriate addresses: origination, fill, array, input and output. (20 points)

2. Display sorted values in console. (20 points)

3. Use appropriate labels and comments. (20 points)

4. Contain appropriate instructions for arithmetic, data movement and conditional operations.

(40 points)

5. Comprise of 2 or more subroutines and implement subroutine calls. (20 points)

6. Use branching for control: conditional and iterative. (30 points)

7. Manage overflow and storage allocation. (20 points)

8. Manage stack: include PUSH-POP operation on stack. (20 points)

9. Include save-restore operations. (30 points)

10. Include pointer (20 points)

11. Implement ASCII conversion operations (30 points)

12. Use appropriate system call directives. (10 point)

13. Testing (20 points): Test the program using the below values (green).

The result of the sort needs to be in the ascending order (small to high).

Input & Output

Input:

11	8	2	17	6	4	3	21

Output:

2	3	4	6	8	11	17	21

2. Overall Description

2.1 Product Perspective

Primary program objectives and goals.
- The primary objective of the program is to sort the number in by numerical value.

Data type
- int

2.2 Product Functions

The overall description of functionality:

Highlight the program functionality: Identify tasks and subtasks of the program in summary.

Description: Bubble sort is a simple sorting algorithm that repeatedly steps through a list of elements, compares adjacent elements, and swaps them if they are in the wrong order.

Task is to sort a list of elements using bubble sort.

Subtasks:

1. Initialize the list of elements to be sorted.
2. Set up a loop to iterate through the list multiple times.
3. Compare adjacent elements and swap them if they are in the wrong order.

4. Repeat the above step for each pair of adjacent elements in the list, moving from the beginning to the end.
5. After completing one pass through the list, the largest element will be in its correct position at the end.
6. Repeat steps 3-5 for the remaining elements (excluding the last one, as it is already in the correct position after each pass).
7. Continue looping through the list until no more swaps are performed during a pass, indicating that the list is sorted.
8. Output the sorted list.

Technical functionality

What are the technical functions of the program? Subroutines and operations.

- Technical functions of the program are sort(), swap(), compare(), and isSorted().

2.3 User Classes and Characteristics

What type of users are involved in this development process? Include business and technical personnel and their tasks.

- **There are a lot of business and technical personnel who may be a user of this process. One example of a business personal is an Analyst and their task may be to gather and document business requirement related to sorting functionalities. One example of a technical personnel may be a software developer and their task may be to write a code in any programming language to perform the sorting operation.**

2.4 Operating Environment

What type of system will the application be operated on? Operating system? System types? Development platform?

- Some operating system this application may be operated on are windows, mac, or Linux. System can be a laptop or PC. Development platform will be LC3Edit.

How should the application be used? Simulator version?

- The application can be used on LC3 simulate.

2.5 Design and Implementation Constraints

Note any constraints or limitations to the application.

- Some constraints or limitations to the application are data types, number of elements and range for the numbers. You can’t sort other data types like string. This application only takes in 8 numbers. This application range for an element is 0-100.

2.6 Assumptions and Dependencies

Note any dependencies.

Is this application dependent on other applications or services? Browser? Web services? Simulator?

- It dependents on any simulator like LC3 simulator. It doesn't need a browser or web services.

3. External Interface Requirements

3.1 **User Interfaces**

The user will communicate with the program via the console. The user is expected to input 8 numbers into the console, each followed by a space.

3.2 **Hardware Interfaces**

Specify hardware interface – computer types? Terminal types?

3.3 **Software Interfaces**

Specify additional software interface – if any. What type of software will the application require to run?

The program requires LC3 edit to view the code and LC3 simulate to execute.

3.4 Communications Interface

Does your application require web, Internet, or network connectivity? If so, which browser? What type of network connection?
No, the program can be run from either an application based LC3 simulator or a web based LC3 simulator, internet is not required.

4. Detailed Description of Functional requirements

4.1 Type of Requirement (summarize from Section 2.2)

What are the functions? Their purposes? Inputs? Outputs? Data? Where is the data stored (internal or external to the application)?

Sort() - begins the process of sorting by taking as its input 8 numbers. Its output will be the 8 numbers sorted in ascending order.

Compare() - called on by the Sort() function to compare two numbers stored next to one another in memory. Its output will be the lesser of the two values.

Swap() - should the Compare() function find that the lesser of the two values is placed higher in memory, the Swap() function will temporarily copy the greater value in a different, temporary address that it points to, then copy the lesser value into the greater value's original address, and finally copy the greater value into the lesser value's original address.

IsSorted() - this function will take in as its input the 8 numbers in the order that they were last adjusted to at the end of every Swap() call, and determine if the list of numbers is in ascending order. If so, it will output the sorted list and terminate the program. If not, the Sort() function will continue.

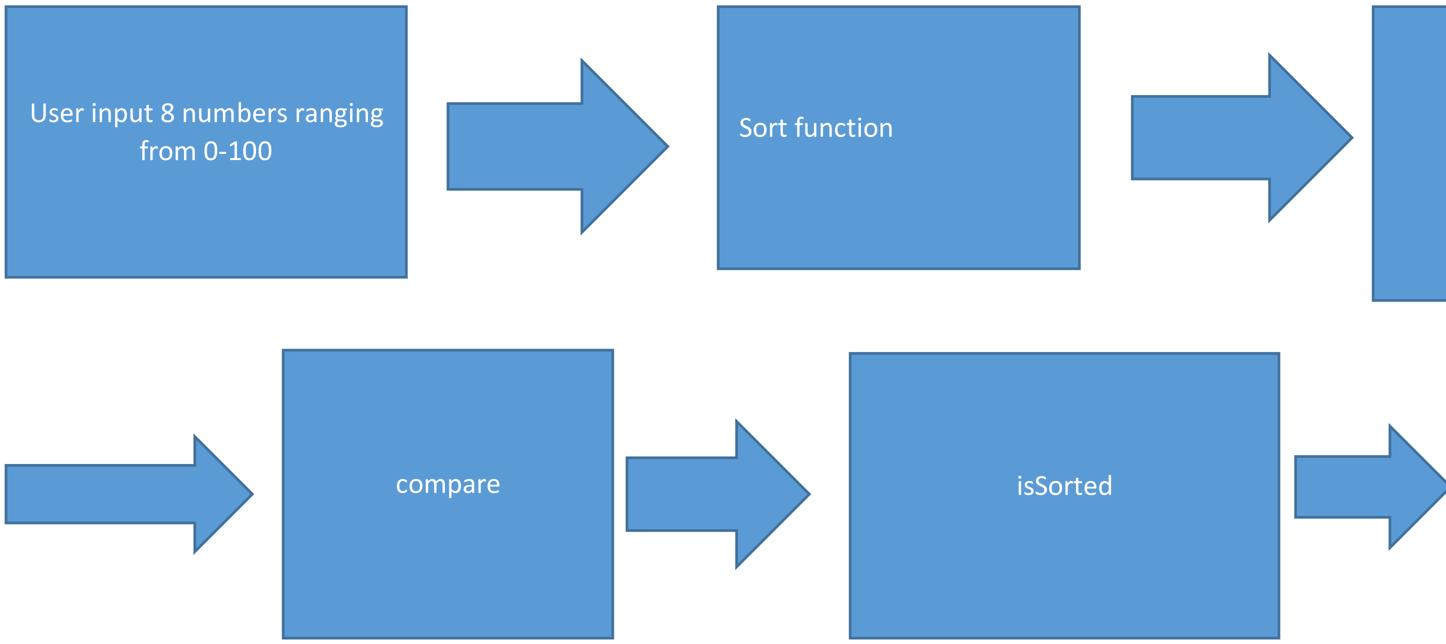
4.2 Performance requirements

What is the expected performance level of the program?

The program is designed to interpret up to 8 numerical values up to 2 digits, this program is not designed to sort words, symbols, letters etc.

In addition, this program is designed to sort numbers in ascending order not descending.

4.3 Flow Chart OR Pseudocode.



Prompt user to input 8 numbers, pressing Enter after each one.
Store each number into a different register in order, R0-R7.
Set sorting pointer to first register

```
Begin Sort() function with all 8 registers as its input.
          Begin Compare() function with the sorting pointer as its input.
              Set compare pointer to sorting pointer + 1 to compare it to the next value.
              Subtract compare pointer value by sorting pointer value and store value in SUM
              variable.
              If positive, break out of Compare() function and continue.
              If zero, break out of Compare() function and continue.
              If negative, call on Swap() function.
                        Begin Swap() function with the sorting and compare pointer as input.
                        Set temporary pointer to TEMP variable.
                        Copy value in sorting pointer to TEMP.
                                  Clear value in sorting pointer.
                                  Copy value in compare pointer to sorting pointer.
                                  Clear compare pointer.
                                  Copy value in TEMP to compare pointer.
                                  End Swap().
              End Compare().
    Begin isSorted() function with the 8 registers as its input.
```