# CIS11 Course Project Part 1: Documenting the Project

**Introduction**

* 1. **Purpose**

The goal of this project is to create an LC-3 program that puts the Bubble Sort algorithm into practice by sorting an array of numbers ranging from 0 to 100 in ascending order. These numbers will be inputted by the user. The bubble sort will repeatedly go through the list, compare two elements, and swap them if they are in the wrong order. The program ends once the list is sorted from least to greatest. This program will also showcase my understanding of LC-3 instructions, subroutines, branching, stack management, and other fundamental concepts in programming.

The objectives are:

- Initialize variables

-Create outer loop

-Create inner loop

-Check array to sort

-Terminate outer loop if array is already sorted

-Output sorted array

* 1. **Intended Audience and Users**

The primary audience for this program includes students enrolled in the CIS11 course, as well as instructors teaching the course. It targets individuals who are learning LC-3 programming and computer architecture.

* 1. **Product Scope**

This program aims to bridge the gap between theoretical knowledge and its practical application. Users will be able to input a set of numbers, witness the Bubble Sort algorithm in action, and see the sorted output. Additionally, the program will demonstrate the effective use of LC-3 instructions and directives for memory management, arithmetic operations, and program flow control; the core understandings of bubble sort will transfer over to more advanced algorithms.

* 1. **Reference**

**Source Documents for the Program Requirements and Specification**

The program's requirements and specifications are based on the course materials on LC-3 programming and the project requirements specified in the course documentation.

**Input**: User input of 8 numbers ranging from 0 to 100.

**Output**: Sorted numbers displayed in ascending order on the console.

**2. Overall Description**

**2.1 Product Perspective**

The main focus of this program is to provide an educational tool for students learning LC-3 assembly language programming. By implementing the Bubble Sort algorithm and incorporating various LC-3 instructions, the program will offer hands-on experience in applying these concepts.

**Input Handling:** Accept and store 8 user inputs.

**Sorting Function:** Implement Bubble Sort to arrange numbers in ascending order.

**Output Display:** Print the sorted numbers to the console.

* 1. **Product Functions**

**The overall description of functionality:**

* **User Input and Interaction:** Accept user input for eight numbers ranging from 0 to 100.
* **Sorting Algorithm:** Implement the Bubble Sort algorithm to sort the input numbers in ascending order.
* **Output Display:** Display the sorted numbers to the console, providing a clear and organized view of the results.
* **Technical Demonstrations:** Showcase the versatility of LC-3 instructions by performing various operations:
  + Arithmetic operations: Basic arithmetic calculations.
  + Data movement: Efficient transfer of data between memory locations and registers.
  + Conditional operations: Decision-making and branching based on specific conditions.
* **LC-3 Concepts and Functionality:** Implement subroutines to demonstrate how reusable code blocks can be defined and called.
  + Utilize branching for control flow, including both conditional and iterative execution paths.
  + Manage stack memory using PUSH and POP operations for efficient memory management.
  + Handle overflow and storage allocation to prevent data loss or corruption.
* **Technical Functionality:** Implement the Bubble Sort algorithm with appropriate labels and comments for improved clarity and maintainability:
  + Display sorted values in ascending order, providing a user-friendly output.
  + Include instructions for arithmetic, data movement, and conditional operations to showcase the range of LC-3 capabilities.
* **Advanced LC-3 Features:** Utilize subroutines and subroutine calls for a modular code structure:
  + Manage overflow and storage allocation to ensure data integrity.
  + Incorporate save and restore operations for maintaining program state across different stages.
* **Memory and Pointer Management:** Implement pointers for accessing memory locations and performing dynamic memory allocation:
  + Perform ASCII conversion operations for character data manipulation and display.
* **System Interactions:** Use appropriate system call directives to interact with the underlying operating system:
  + Facilitate input/output operations and memory management.
* **Subroutines and Operations Refinement:** Define specific subroutines for frequently used code blocks to improve reusability and maintainability:
  + Operations include arithmetic calculations, data movement, conditional statements, iteration, and effective handling of user input and output.
  1. **User Classes and Characteristics**

**Business Personnel:**

* **Students**: Students enrolled in the CIS11 course who will be using the program to learn LC-3 programming and computer architecture.

**Technical Personnel:**

* **Instructor**: The instructor teaching the CIS11 course who will be using the program to evaluate student understanding and provide feedback.
  1. **Operating Environment**

**System**: The application will be developed and run on a system capable of simulating the LC-3 architecture.

**Operating System**: Any system that supports the LC-3 simulator.

**Development Platform**: LC-3 simulator

* 1. **Design and Implementation Constraints**

The program must adhere to the constraints of the LC-3 instruction set architecture, memory limitations, and simulator capabilities.

* 1. **Assumptions and Dependencies**

This application hinges on the LC-3 simulator to execute the assembly code and facilitate user input and output. It doesn't necessitate a browser or web services. Instead, it relies on the simulator to faithfully emulate the LC-3 system. Therefore, the program's seamless operation is predicated on the accessibility and correct setup of the LC-3 simulator.

**3. External Interface Requirements**

* 1. **User Interfaces**

The user will interface with the program through the LC-3 simulator console, where they will input the numbers to be sorted and view the sorted output. The program may also utilize menus or other interactive elements provided by the simulator.

* 1. **Hardware Interfaces**

The hardware interface requirements are relatively flexible but include a computer capable of running the LC-3 simulator efficiently. This can include a range of computer types, from desktop PCs to laptops, ensuring compatibility with the simulator's system requirements. Standard input/output devices such as a keyboard and display are necessary for user interaction and program output visibility.

* 1. **Software Interfaces**

The program will require the LC-3 simulator software to execute the assembly code and provide the necessary environment for user interaction and output display.

* 1. **Communications Interface**

The application does not require web, Internet, or network connectivity.

**4. Detailed Description of Functional requirements**

**4.1 Type of Requirement (summarize from Section 2.2)**

**Functions:**

1. **User Input:** Accept eight numbers (0-100) from the user.
2. **Sorting:** Implement Bubble Sort to arrange numbers.
3. **Output Display**: Show sorted numbers on the console.
4. **Technical Demonstrations:** Illustrate LC-3 instructions:

* Arithmetic, data movement, and conditionals.

1. **LC-3 Concepts:** Showcase subroutines, branching, stack management, and data handling.
2. **Advanced Features:** Include overflow management, save-restore operations, and pointer usage.
3. **Memory Management:** Utilize memory efficiently.
4. **System Interactions:** Interface with the OS for I/O operations.

**Purpose:** Enable user input and sorting of numbers using Bubble Sort.

**Inputs:** Eight numbers (0-100) from the user.

**Outputs:** Sorted numbers displayed on the console.

**Data:** Input, sorted output, and intermediate data stored internally for processing.

**4.2 Performance requirements**

The program is expected to perform the Bubble Sort algorithm efficiently, sorting the array of 8 numbers in a reasonable amount of time. The overall execution speed should be sufficient for educational purposes and user interaction.

**4.3 Flow Chart OR Pseudocode.**

Initialize all registers to 0

Use R6 as the frame pointer

Load the file pointer into R3

Set R2 as a counter for the number of items (initialized to 0)

; ASCII conversion subroutine

ASCII Convert:

Save R0 and R1 on the stack

Save the number to R1

; Handle tens place (for double digit inputs)

If less than 10, skips tens place

Subtract 10 to get the tens digit

Convert tens digit to ASCII

Print the tens digit

Print Ones:

Convert ones to ASCII

Print the ones digit

Restore R0 and R1 from the stack

; Push subroutine (save state)

PUSH:

; Pop subroutine (restore state)

POP:

; Count the number of items

Count Items (LOOP):

Load the next file item into R0 using the file pointer in R3

If R0 is 0, go to END\_COUNT

Increment the file pointer

Increment the counter

Repeat LOOP

END\_COUNT:

Store the total number of items in R4 (outer loop counter)

If R4 is 0, go to SORTED (empty file)

; Do the bubble sort

OUTERLOOP:

Decrement R4 (loop n - 1 times)

If R4 is non-zero, go to SORTED (looping complete, exit)

Copy the value of R4 to R5 (initialize inner loop counter to outer)

Load the file pointer to the beginning of the file

INNERLOOP:

Load the item at the current file pointer into R0

Load the next item into R1

Calculate the difference between the next item and the current item

If the difference is negative or zero (in order), go to SWAPPED (don't swap)

;Swap the items

Store R1 (next item) in the current position  
 Store R0 (current item) in the next position

SWAPPED:

Increment the file pointer

Decrement the inner loop counter

If the inner loop counter is positive, repeat INNERLOOP

Decrement the outer loop counter

If the outer loop counter is non-zero, repeat OUTERLOOP

SORTED:

Reset the file pointer

Reset counter for display

; Display sorted values in ascending order

Display Sorted:

Load the next item

If R0 is 0, end display

Convert to ASCII

Output character

Increment file pointer

Decrement counter

Repeat

End Display:

Halt the program

; Data section

; File data

FILE:

Define the memory location of the file

END