Team Members:

Jeptha Peoples

Candace Willson

Allen Camarena

Advisor: Kasey Nguyen, PhD

CIS-11 Project Documentation

Team Tech

Appendix A: Program Pseudocode and Flow Logic

PART I: Application Overview

Objectives:
The goal of this application is to streamline the process of inputting, validating, analyzing, and reporting test scores. The application allows users to enter multiple test scores, validates them to ensure they fall within a specified range, assigns a corresponding letter grade, and then calculates and displays the maximum, minimum, and average scores.

Why are we doing this? This project simplifies the average of student test scores. It eliminates manual calculation errors, improves data through input validation, and grades assignments. Putting this now ensures students and teachers access accurate and immediate feedback.

Business Process: Test scores may be managed manually or through disconnected systems. Our application introduces a unified process: users input scores into the system, which then processes and displays results in real time. This leads to better accuracy.

-Existing Business Process
Currently, the existing process has the instructor manually calculating minimum, maximum and average test scores that are provided with letter grades for students. The instructors can have errors in their calculations, resulting in a long process to correct the mistakes. The results will most likely be recorded in a spreadsheet or a gradebook.

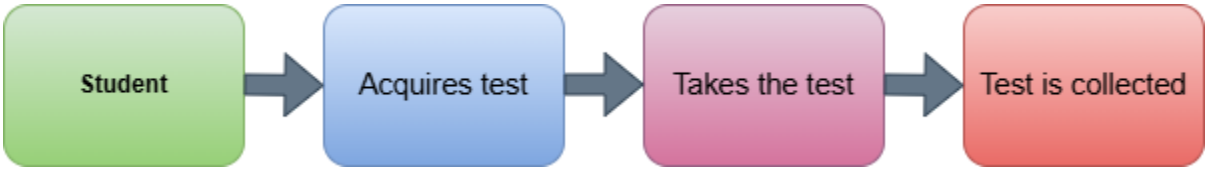
-Future Business Process
The future business process involves the application, calculating the Minimum, Maximum, Average test scores and providing a letter grade. The teacher simply inputs the test scores and the program outputs the grade or the student. This eliminates the process of doing manual calculation while also reducing potential errors. The results are then easily imputed into a spreadsheet or gradebook.

User Roles and Responsibilities:

- **Student:**
    1. Student is provided test by the teacher
    2. The students take the test
    3. When test is finished, the teacher collects them all

- **Time**
- The teacher takes a few minutes to pass the exam. The students might spend many minutes to an hour to complete the test.

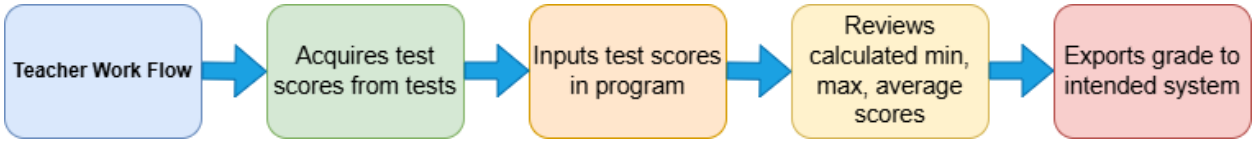
- **Work Flow Chart**



- **Teacher/Administrator:** Reviews the output scores and grades.
    1. Tests students
    2. Acquires test scores from tests
    3. Inputs test scores in program
    4. Reviews calculated minimum, maximum, average scores and letter grades
    5. Exports grade to intended system

- **Time**
  Depending on the educator schedule the test scores may take days or even weeks to input and calculate test scores. It could take months to export and record the grades.

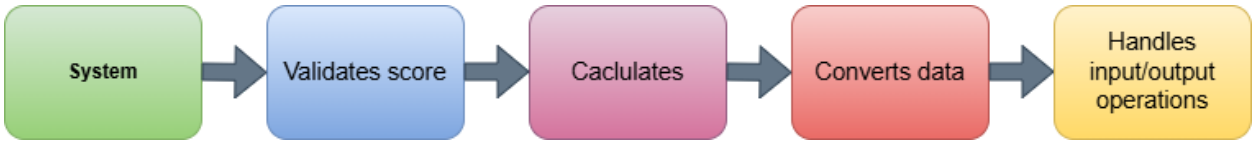
- **Work Flow Chart**



- **System:**
    1. Validates scores
    2. calculates them
    3. converts data
    4. handles input/output operations.

- **Time**
  The system may take a few seconds to a few minutes to process and organize scores.

- **Work Flow Chart**



Production Rollout Considerations: Deployment involves setting up the application in an educational setting, providing a simple UI for users. Initial data for testing will use predefined score sets.

Terminology:

- **R1–R4:** Register names used in the LC-3 architecture.
- **charArray:** Array storing ASCII character values.
- **testArray:** Array storing integer test scores.
- **ValidateScore:** Subroutine that ensures scores fall within the allowed range (0–100).
- **ReturnGrade:** Subroutine that assigns letter grades based on numeric values.
- **Test Score:** The total score a student acquired on a test
- **Minimum:** The lowest performing test score a student acquired.
- **Maximum:** The highest test score a student acquired
- **Average Score:** The average test score was acquired based on the low and highest score from all students.
- **Letter Grade:** A letter grade (A, B, C, D, F) represents a students performance based on their test scores

---

PART II: Functional Requirements

Statement of Functionality:
The application allows users to input up to 5 test scores. Each score is validated to be within the range 0–100. After input, the application calculates and displays the maximum score, minimum score, and average score, converting each to ASCII for output. Letter grades are assigned per score. The output should be clearly formatted and easy to understand.

Scope:
This version covers basic score input, validation, grading, and statistical summary. Future phases may include data export, graphical output, or with school databases.

Performance:

- Validate and store a score in under 1 second.
- Calculate and print all analytics in under 3 seconds for 5 scores.
- All requirements for running this program are low and not too demanding.
- Any software and hardware can run the program without any problems as long as there is 16 bits of memory and 4 mb of available ram.

Usability:

- Simple text input interface.
- Clear error messaging and prompts for invalid entries.
- The output is formatted clearly with labels like “Max Score:”

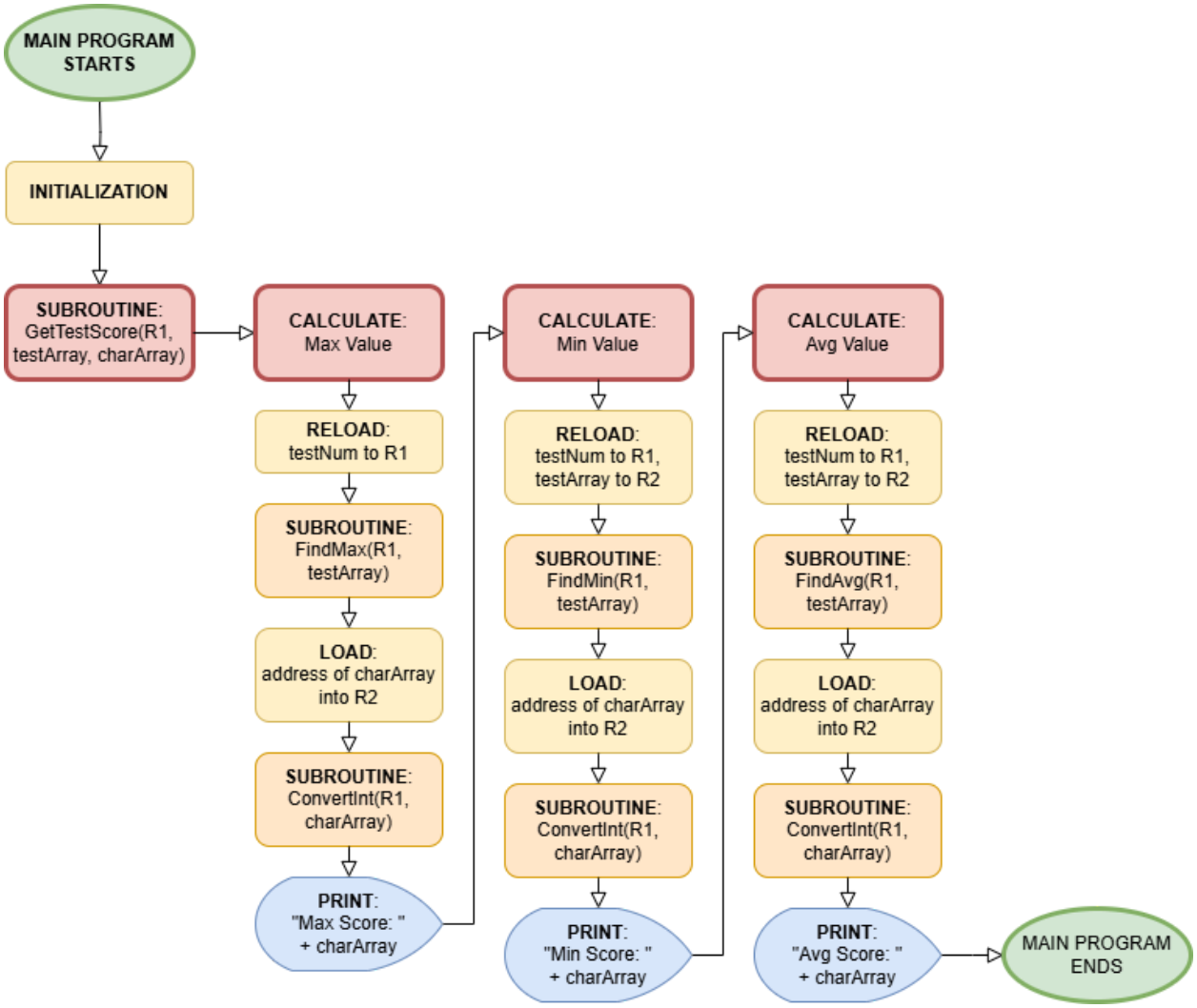
Documenting Requests for Enhancements:

Date	Enhancement	Requested by	Notes	Priority	Release No/Status
05/23/2025	Add support for more than 5 scores	Instructor	Allow dynamic array allocation	Medium	Future
05/23/2025	Implement graphical output	Student	Show a bar graph of scores	Low	Backlog
05/23/2025	Export results to text file	Admin	Save all outputs to a downloadable .txt	High	Planned

---

PART III: Appendices

Flow Chart



Pseudocode

1. **MAIN PROGRAM**:
    a. INITIALIZE:
        i. LOAD: testNum - number of tests, R1
        ii. LOAD: address of testArray - 5 elements for the test scores, R2
        iii. LOAD: address of charArray - contains characters of test scores, R3

iv. LOAD: address of scoreInput - contains the score the user inputs, R4

b. CALL: GetTestScores(R1, testArray, charArray) ➜ validation & grade output also occurs here
c. CALCULATE: max score
    i. RELOAD: testNum to R1
    ii. CALL: FindMax(R1, testArray) ➜ store result in R1
    iii. LOAD: address of charArray into R2
    iv. CALL: ConvertInt(R1, charArray)
    v. PRINT: “Max Score: “
    vi. PRINT: charArray
d. CALCULATE: min score
    i. RELOAD: testNum to R1
    ii. RELOAD: testArray to R2
    iii. CALL: FindMin(R1, testArray) ➜ store result in R1
    iv. LOAD: address of charArray into R2
    v. CALL: ConvertInt(R1, charArray)
    vi. PRINT: “Min Score: “
    vii. PRINT: charArray
e. CALCULATE: avg score
    i. RELOAD: testNum to R1
    ii. RELOAD: testArray to R2
    iii. CALL: FindAvg(R1, testArray) ➜ store result in R1

iv. LOAD: address of charArray into R2

v. CALL: ConvertInt(R1, charArray)

vi. PRINT: “Avg Score: “

vii. PRINT: charArray

f. END MAIN PROGRAM

2. **SUBROUTINE**: Push - to save data onto a stack
    a. Decrement stack pointer
    b. Store value at new stack location
3. **SUBROUTINE**: Pop - to use saved data from the stack
    a. Load value from stack location
    b. Increment stack pointer
4. **SUBROUTINE**: ConvertASCII - to convert ASCII character into integer (important to treat input as ints to validate input & assign grade), NEED: charArray
    a. Initialize result = 0
    b. For loop: looping through characters in charArray until null terminator
        i. Convert ASCII to its numeric value
        ii. Multiply that value by 10
        iii. Add the digit to the result
    c. Return result
5. **SUBROUTINE**: ConvertInt - to convert integer into ASCII (in order to display as output for min/max), NEED: intVal, charArray
    a. Initialize digitNum = 0
    b. Copy intVal to the temp storage

c. Do-While loop: counting the digits of intVal until temp storage = 0
    i. Divide temp by 10
    ii. Increment digitNum
d. Null-terminate the array
e. While loop: iterating through intVal to convert into ASCII value and store in array until intVal = 0
    i. Divide inVal by 10 to get the remainder
    ii. Convert the remainder to ASCII
    iii. Store the ASCII value in an array
f. Return charArray

6. **SUBROUTINE**: GetInput - to get the unvalidated input from the user and convert it into an integer (used in ValidateScore), NEED:
    a. Display input message (“Enter Test Score: “)
    b. Initialize string pointer
    c. Loop: looping through each character of scores entered
        i. If the character entered is “enter” then
            1. Break
        ii. Store character in charArray
    d. Null-terminate charArray
    e. Call ConvertASCII for charArray to convert and store result in R1
7. **SUBROUTINE**: ValidateScore - to make sure the score entered is within 0-100, NEED:
    a. Loop: looping through GetInput subroutine while input is invalid

i. Call GetInput

ii. If R1 < 0 OR R1 < 100 then

1. Print “INVALID: Score must be between 0-100.”
2. Goto Loop (repeats until score is valid)

iii. else

1. Break

8. **SUBROUTINE**: ReturnGrade - to assign the grade for each test, NEED: value from R0
    a. Copy R0 to R1
    b. Set grade = ‘A’
    c. If R1 < 90 then
        i. grade = ‘B’
    d. If R1 < 80 then
        i. grade = ‘C’
    e. If R1 < 70 then
        i. grade = ‘D’
    f. If R1 < 60 then
        i. grade = ‘F’
    g. Print grade
9. **SUBROUTINE**: GetTestScores - to call in other subroutines and end with a valid stored score array and printed letter grades for each of those scores
    a. For loop: looping through number of scores (testNum)
        i. Call ValidateScore ➜ GetInput called within here

            1. If invalid, loops to have user re-enter score
        ii. Call ReturnGrade by passing (now valid) value in R1 to R0
        iii. Store R1 in testArray
10. **SUBROUTINE**: FindMax - to find the maximum test score value, NEED: testNum, testArray
    a. Set the max to the first score entered
    b. For loop: looping through each of the remaining scores entered
        i. If current > max, then
            1. Set max = current
    c. Return max
11. **SUBROUTINE**: FindMin - to find the minimum test score value, NEED: testNum, testArray
    a. Set the min to the first score entered
    b. For loop: looping through each of the remaining scores entered
        i. If current < min, then
            1. Set min = current
    c. Return min
12. **SUBROUTINE**: FindAvg - to find the average of the test scores, NEED: testNum, testArray
    a. Initialize sum = 0
    b. For loop: looping through each of the scores
        i. Add current to sum
    c. Divide sum by testNum

d. Return avg

13. **SUBROUTINE**: Mult - basic multiplication subroutine (used in ConvertASCII)
    a. Save return address and registers using push
    b. Set product = 0
    c. While loop: looping through adding the multiplier while multiplicand > 0
        i. product = product + multiplier
        ii. Decrease multiplicand by 1
    d. Store product in R1
    e. Restore used registers and return address using pop
14. **SUBROUTINE**: Div - basic division subroutine (used in ConvertInt & FindAvg)
    a. Save return address and registers using push
    b. Set quotient = 0
    c. Negate the divisor and save value
    d. While loop:
        i. dividend = dividend + (-divisor)
        ii. Increase quotient by 1
    e. Store dividend (remainder) in R1 and quotient in R2
    f. Restore used registers and return address using pop