*Leonardo Rodriguez*

*Valentinno Cruz*

*CIS 11 Assembly Programming*

*Grade Calculator Project*

***Documentation***

# IS11 Course Project Part 1: Documenting the Project

**Introduction**

* 1. **Purpose**

The purpose of this program is to display a minimum, maximum and average test scores. Based on the users entered values a letter grade will be populate.

* 1. **Intended Audience and Users**

The primary audience would be any instructor, school faculty or individual that may need to calculate a Minimum, Average, and Maximum values.

* 1. **Product Scope**

What is the intention of this program?

The Test Score Calculator is intended for anyone (professor, student, instructor, etc) who may need to calculate a grade with the Minimum, Maximum, and Average. This will eliminate having to calculate Average, Minimum, and Maximum scores by hand. This alleviates time for the individual whom is calculating said grades.

* 1. **Reference**

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

Reference Project requirements and LC-3 specifications

1. **Course Project instructions Part 1**
   1. Contain appropriate addresses: origination, fill, array, input and output. (20 points)
   2. Display minimum, max, average values/grades 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 (52, 87, 96, 79, 61)
2. **Course Project Objectives and Tasks Outline** 
   1. Plan and design an LC-3 program.
   2. Create flowcharts and pseudocode for program.
   3. Compose documentation to specify the objectives, components and implementation techniques.
   4. Test the program to achieve accuracy, efficiency and performance.
   5. Debug, modify and control version on GitHub.
   6. Build a successful program, with team collaboration, to meet project objectives.
3. **Requirements**
   1. Review program requirements, pseudocode (from part 1) and flowchart (from part 1).
   2. Write an LC-3 program that meets the requirements based on the given project options.
   3. Test and analyze the program outcome to assess program limitations, performance, efficiency and future improvements.
   4. Submit LC-3 program (.asm file)

**Companion Application Requirements Documents (If applicable)**

1. Team Task Assignment.doc
2. Pseudocode.doc
3. Flowchart.doc

**2. Overall Description**

**2.1 Product Perspective**

What other documents should be reviewed with this document?

The program:

* Prompts the user to input five score values.
* Computes Max test score and outputs result
* Computes Minimum test score and outputs result
* Computes Average test score and outputs result
* Sums up values and outputs a letter grade
  1. **Product Functions**

**The overall description of functionality:**

Primary program objectives

1. Initializations
2. Get test scores
3. Find Max Test score
   1. Convert it to string
   2. Print it
4. Find Minimum Test score
   1. Convert it to string
   2. Print it
5. Find Average Test score
   1. Convert it to string
   2. Print it
6. Find Grade
   1. Print Grade

**Technical functionality**

A configurable toolkit of functions including:

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

1. Find Maximum, Minimum, Average and Grade Value using Loop and Array input
2. Convert Integer to Alphanumeric
3. Division/Multiplication in conjunction with loop
4. Convert Alphanumeric to Integer
   1. **User Classes and Characteristics**

Who are involved in this development process? Include business and technical personnel and their tasks.

**Leonardo Rodriguez:** Responsible for planning and effectively executing a successful flowchart. Also, collaboratively developed and tested Assembly Language application.

**Valentinno Cruz:** Responsible for creating a efficient and straightforward pseudocode Also, collaboratively developed and tested Assembly Language application.

**Operating Environment**

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

The “LC-3 editor” application will be operated on Windows platform.

* 1. **Design and Implementation Constraints**

Note any constraints or limitation to the application.

The application will only be able to calculate whole numbers. If the user were to input a partial value, then the application will cease to operate.

* 1. **Assumptions and Dependencies**

Note any dependencies

It is assumed that the user has all the data needed to perform tasks using the application. Also, it is presumed said individual is familiar with operating a numpad or a keyboard with numeric characters. Lastly, it is also expected that the user has the necessary program to run the application.

**3. External Interface Requirements**

* 1. **User Interfaces**

How will the user interface with your program? Menus? Access prompt? Links? Icons?

The user will be greeted with a prompt asking them to input their values. Then, the user will use a numeric pad or keyboard to key in their data. After which, the application will output the desired information.

* 1. **Hardware Interfaces**

Specify hardware interface – computer types? Terminal types?

The application can be executed on any Windows, Linux, or MacOS given they have an intelligent terminal.

* 1. **Software Interfaces**

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

The application will run on any program with the capabilities of Compiling and executing assembly code such as but not limited to: Visual Studio Code, LC-3, etc etc.

* 1. **Communications Interface**

Does your application require web, Internet or network connectivity? If so, which browser? What type of network connection?

 Not network connectivity is 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)?

**Example**: Grade Calculator

**Purpose**: To provide Maximum, Minimum, and Average Test scores

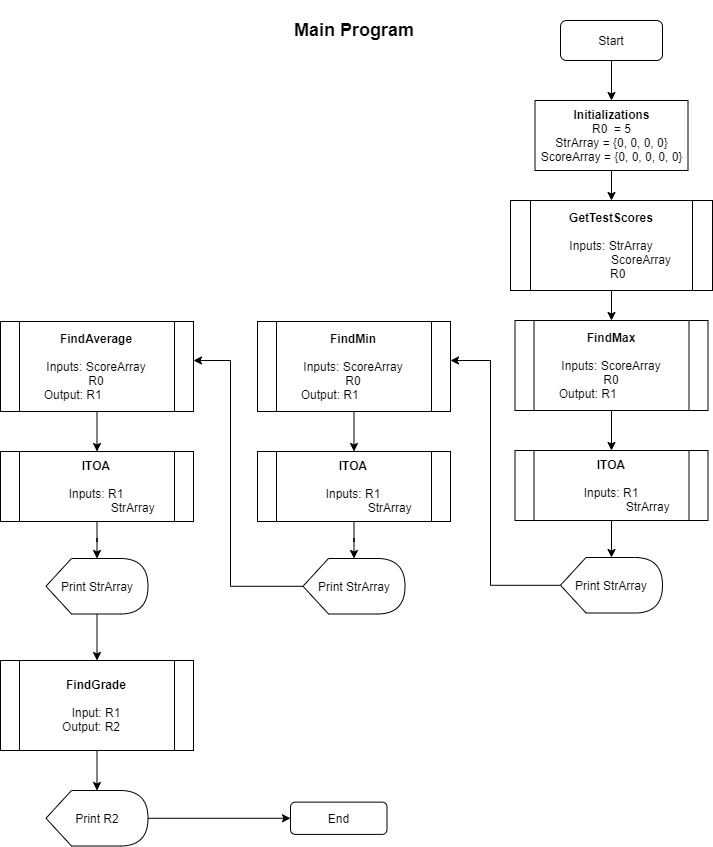
**Inputs**: Via Keyboard and/or numeric keypad

**Processing**: The Data will be filtered for desired outcome

**Outputs**: Maximum, Minimum, and Average Test scores with a letter grade equivalent.

**Data**: User database

**4.2 Performance requirements  
 What is the expected performance level of the program?**

1. It should be accessible to any user regardless of location, or network.
2. The application will be outputting simple data for the user, thus the response time for said calculations should not exceed five seconds.
3. It should be capable of dealing with large value numbers and calculating them effectively.
4. The application should be able to handle all runtime errors
   1. **Flow Chart and Pseudocode**

|  |
| --- |
|  |
|  |  |

**Pseudo Code**

Pseudo Code of the program that displays the minimum, maximum and average grade of 5 test scores and display the letter grade associated with the test scores

**Given:**

*StrArray*, an array having four elements

*ScoreArray*, an array having 5 elements

**Steps:**

1. R0 = 5 ; Number of test scores
2. GetTestScore(*StrArray*, *ScoreArray*, R0) ; Get 5 test scores
3. R1 = FindMax(*ScoreArray*, R0) ; Find max score
4. ITOA(R1, *StrArray*) ; Covert it to string
5. Puts(*StrArray*) ; Print it
6. R1 = FindMin(*ScoreArray*, R0) ; Find min score
7. ITOA (R1, *StrArray*) ; Covert it to string
8. Puts(*StrArray*) ; Print it
9. R1 = FindAverage(*ScoreArray*, R0) ; Find average score
10. ITOA(R1, *StrArray*) ; Covert it to string
11. Puts(*StrArray*) ; Print it
12. R1 = FindGrade(R1) ; Find grade letter
13. Putc(R1) ; Print grade

**GetTestScores(*StrArray*, *ScoreArray, N*)**

**Input:**

*StrArray*, an array having four elements

*ScoreArray*, an array having *N* elements

*N* contains the number of test score to take input

**Output:**

None

**Assumption:**

Test scores are in between 0 and 100.

**Algorithm:**

1. R0 = *N* ; Loop counter
2. R1 = 0 ; *ScoreArray* index
3. R2 = 0 ; *StrArray* index
4. R3 = getc ; Get a character i.e. an ascii digit
5. Go to step-9 if R3 = 10 ; End getting ascii digits as ENTER encountered
6. *StrArray*[R2]= R3 ; Else, save the ascii digit
7. R2 = R2+1 ; Move on to the next location in *StrArray*
8. Go to step 4
9. *StrArray*[R2] = 0 ; NULL terminate it
10. R4 = ATOI(*StrArray*) ; Convert the ascii formatted number to decimal
11. *ScoreArray*[R1] = R4 ; Save the decimal test score in *ScoreArray*
12. R1 = R1+1 ; Move on to the next location in *ScoreArray*
13. R0 = R0 -1 ; Decrement loop counter
14. Go to step-3 if R0 is not zero
15. End

**FindMax(*Array*, *ArraySize*)**

**Input:**

*Array* containing test scores

*ArraySize* containing the number of test scores *Array* contains

**Output:**

Max test score

**Assumption:**

Array contains at least one test score

**Algorithm:**

1. R3=0 ; Array index
2. R2 = *Array*[R3] ; R2 has the first test score
3. R1 = *ArraySize* ; R1 will serve as loop counter
4. R1 = R1-1 ; Decrement loop counter
5. Go to step-11, if R1 is zero ; End loop if all the test scores have been compared
6. R3 = R3+1 ; Move on to the next test score
7. R4 = *Array*[R3] ; R4 contain the next test score
8. Go to step-4, If R4<R2 ; Check if need to update our current max, R2
9. R2=R4 ; If yes, the update it
10. Go to step-4 ; Repeat the loop
11. Output R2 ; R2 has the max test score

**FindMin(*Array*, *ArraySize*)**

**Input:**

*Array* containing test scores

*ArraySize* containing the number of test scores *Array* contains

**Output:**

Min test score

**Assumption:**

Array contains at least one test score

**Algorithm:**

1. R3=0 ; Array index
2. R2 = *Array*[R3] ; R2 has the first test score
3. R1 = *ArraySize* ; R1 will serve as loop counter
4. R1 = R1-1 ; Decrement loop counter
5. Go to step-11, if R1 is zero ; End loop if all the test scores have been compared
6. R3 = R3+1 ; Move on to the next test score
7. R4 = *Array*[R3] ; R4 contain the next test score
8. Go to step-4, If R4>R2 ; Check if need to update our current min, R2
9. R2=R4 ; If yes, the update it
10. Go to step-4 ; Repeat the loop
11. Output R2 ; R2 has the min test score

**FindAverage(*Array*, *ArraySize*)**

**Input:**

*Array* containing test scores

*ArraySize* containing the number of test scores *Array* contains

**Output:**

Average test score

**Assumption:**

Array contains at least one test score

**Algorithm:**

1. R3 = 0 ; Array index
2. R2 = *0* ; R2 will store the sum of all the test scores
3. R1 = *ArraySize* ; R1 will serve as loop counter
4. R4 = *Array*[R3] ; Get the test score
5. R2 = R2 + R4 ; Accumulate it
6. R3 = R3+1 ; Move on to the next test score
7. R1 = R1-1 ; Decrement loop counter
8. Go to step-4, if R1 is not zero ; Keep on accumulating until all have been processed
9. R2 = Divide(*R2*, *ArraySize)* ; Divide sum by total test scores to get the average
10. Output R2 ; R2 has the average test score

**FindGrade(*AverageScore*)**

**Input:**

*AverageScore* containing average test score

**Output:**

Corresponding letter grade

**Assumption:**

*AverageScore* is between 0 and 100 both inclusive

**Algorithm:**

1. R0 = 90
2. R1 = *AverageScore*
3. R2 = ‘A’ ; Initially assume that the grade might be A
4. R3 = 60
5. Go to step-10, if R1>=R0 ; Output the grade if R2 is within the scoring limit
6. R0 = R0-10 ; Move on to the previous limit
7. R2 = R2+1 ; Move on to the next grade
8. Go to step-5, if R0>=R3 ; Keep on doing that until we are in F grade limit
9. R2 = R2+1 ; R2 = ‘F’
10. Output R2 ; R2 has the grade letter

**ATOI(*Array*)**

**Input:**

*Array* containing an unsigned number as a sequence of ascii digits e.g. ‘1’, ‘2’, ‘3’

**Output:**

Corresponding unsigned number in decimal form

**Assumption:**

*Array* is NULL-terminated i.e. that last element of array is 0

**Algorithm:**

1. R0 = 0 ; Array index
2. R1 = 10 ; Multiplier
3. R2 = 0 ; R2 will store the corresponding number in decimal
4. R3 = -48 ; Inverse ASCII offset
5. R4 = *Array*[R0] ; Get an ascii digit
6. Go to step-12, if R4 is zero ; End of conversion, if null character is encountered
7. R2 = Multiply(*R2*, *R1*) ; R2 = R2\*R1
8. R4 = R4 +R3 ; Convert it to corresponding decimal digit
9. R2 = R2+R4 ; Keep on accumulating the number in R2
10. R0 = R0+1 ; Move on to the next character in *Array*
11. Go to step-5
12. Output R2 ; R2 has the corresponding decimal number

**ITOA(*Value*, *Array*)**

**Input:**

*Value* contains the unsigned decimal number that needs to represented as an ascii formatted string

*Array* of enough size to be able to store the corresponding ascii formatted string

**Output:**

None

**Assumption:**

None

**Algorithm:**

1. R0 = 0 ; Digits counter
2. R1 = 10 ; Divisor
3. R2 = *Value*
4. R0 = R0+1 ; Increment digit counter
5. [R2, R3] = Divide(R2, R1) ; Keep on dividing *Value* by 10
6. Go to step-4 if R2 is not zero ; until the quotient becomes zero
7. R2 = *Value*
8. *Array*[R0] = 0 ; NULL terminate the *Array*
9. R3 = R0-1 ; *Array* index (of the end of the *Array)*
10. R4 = 48
11. [R2, R5] = Divide(R2, R1) ; Dividing *Value* by 10
12. R5 = R5+R4 ; Convert the remainder to its ascii representation
13. *Array*[R3] = R5 ; Save it in the *Array*
14. R3=R3-1 ; Move on to the previous location in the *Array*
15. Go to step-11 if R2 is not zero ; Keep on doing until the quotient becomes zero
16. End

**Multiply(*num1, num2*)**

**Input:**

*num1* and *num2*, the two unsigned numbers that need to be multiplied

**Output:**

Product of *num1* and *num2*

**Algorithm:**

1. R0 = *num1*
2. R1 = *num2* ; Loop counter
3. R2 = 0 ; R2 will contain the product
4. Go to step-8, if R1 is zero
5. R2 = R2+R0 ; Keep on adding R0, R1 number of times
6. R1 = R1-1 ; Decrement loop counter
7. Go to step-4
8. Output R2 ; R2 has the product

**Divide(*num, den*)**

**Input:**

*num* and *den*, the unsigned numerator and the denominator

**Output:**

Quotient and Remainder of the division of *num* by *den*

**Assumption:**

*den* is not equal to zero

**Algorithm:**

1. R0 = *num*
2. R1 = *den*
3. R2 = 0 ; R2 will contain the quotient
4. Go to step-8, if R0 <R1 ; End subtracting *den* from *num* if *num*<*den*
5. R0 = R0-R1 ; Else, keep on subtracting *den* from *num*
6. R2 = R2+1 ; Increment quotient
7. Go to step-4

Output R2, R0 ; R2 has the quotient, R0 has the remainder