CIS-11 Project Documentation

Coding Crusaders

Talia Craft,

Celeste Hernandez Mora,

Melanie Caines

Option B: Test Score Calculator

26 May 2024

Advisor: Kasey Nguyen, PhD

Part I – Application Overview

Objectives

This project's objective is to create a program that accurately calculates and displays the maximum, minimum, and average from a set of five test scores given by the user. Additionally, the program will provide a letter grade for each of the test scores based upon the grading criteria. This program will ensure accurate computations are made to provide a quick and simple alternative to evaluating the statistics of test scores. One key objective of this program is creating a reliable user-friendly interface that provides clear prompts for the user and in return gives accurate values.

The business objectives of this project are to reduce and simplify the work of its users. It specifically contributes to educational motives by providing statistical information on test score data. To achieve this objective the program will find the highest value, lowest value, and average of all values provided. Along with saving time, this program also ensures improved accuracy while evaluating scores.

The timing of this project is important as it addresses the needs for efficient tools for evaluating data. Beginning the project now will ensure that the reliance on less efficient methods for computing test score data is minimized, leading to more accurate and timely analysis of a student's performance. Furthermore, this project may aid in the development of further projects within the scope of data analysis.

The primary people who will benefit from this project include educators and students. This group deeply relies on timely and accurate analysis of grades and test score data. By delaying the project, we withhold a valuable tool for this

group to benefit from. This project aims to help this group of people and by considering alternative projects it may alter out objectives. Therefore, to ensure that our priorities align with aiding in educational pursuits, we believe this is the best project to aid in our goals.

Business Process

Currently, the business process existing in educational institutions for score data analysis involves either manual or automated methods. This may include the time-consuming process of grading and analyzing test scores manually to calculate the minimum, maximum, and average values. There are also automated programs for score analysis such as the method used in Canvas. Canvas can provide the minimum, maximum, and average of scores along with providing each student with a letter grade.

The goal of this project is to emulate this process within an LC3 program. Within this program the user should be able to input five scores for analysis. The program will then accurately calculate the minimum, maximum, and average of the test scores and provide the user with a letter grade for the scores. This LC3 program may also contribute to future projects that must utilize sorting, calculating, or assigning values.

In this section you describe the business process and how your application will be used in this context.

In some cases you will need two sections of this sort – one describing the existing business process using existing systems and the other describing the future business process using the system you are developing. This happens any time the business process changes once your system is introduced. When this is the case, the purpose of describing the existing business process is to have a basis of reference to explain the new business process.

> If the business process won’t change when your application is introduced, you should be able to describe it in a single section. However, in this case be sure you understand and communicate to others what value your application brings to the customers. This question should be answered in the Objectives section.

User Roles and Responsibilities

This project will be executed by:

* Talia Craft, who shall oversee documentation and code.

* Celeste Hernandez Mora, who shall oversee documentation and code.

* Melanie Caines, who shall oversee documentation and code.

The contributors will be collaborating on every aspect of the system's development. In this way, there will be significant supervision for each process that is in development.

The tasks and overall flow of the development process has been mapped out as such:

In this section you describe who the users are and how the system fits into what they do.

> You need to list all users for your system in terms of user roles. Typically each individual performs multiple roles in the course of his work since his job involves meeting multiple business objectives. A user role is related to meeting a specific business

objective. When gathering requirements it is most useful to consider roles since you will want to focus only on those business objectives that are relevant to your application.

For each role you need to list the tasks that involve the use of your system (directly or indirectly). You also need to describe the relationships among the tasks for each individual user role and the hand-offs from one role to another. This is usually represented as a workflow diagram.

Consider time in describing tasks and their relationships – different sets of tasks may be performed at different times (daily, monthly, etc.) and several workflow diagrams may be needed.

Once you have written the Objectives, Business Process, and the User Roles and Responsibilities sections, give them to the business expert to read. If you and he agree on what's written, congratulations! You are well on your way to understanding what needs to be done.

Production Rollout Considerations

To ensure the project is communicated to the world efficiently, the simplicity of its objectives can be emphasized. Its purposes can be summed into four phrases: maximum, minimum, average, and label. The label refers to the letter grade in this project. This program's missions can be summarized in a user-friendly way.

The project will be introduced following the letter grading system of A-F for scores of 0-100. The provided set of values consists of 52, 87, 96, 79, and 61. The expected results should be a maximum of 96 with a grade of A, a minimum of 52 with a grade of F, and an average of 75, which is around a C in terms of letter grades. The expectation is that this program will be employed by all those who wish to quickly access the highest, lowest, and average of values.

The strategy for production rollout of this project it to utilize documentation heavily. This will ensure that the program is following our own expectations and objectives. Furthermore, the program itself will undergo testing. This allows us to understand the full scope of the program and apply adjustments where needed.

Terminology

FUNCTION - Relevant content is sorted into a body based on functionality so that it may be used in the future.

ASSEMBLY LANGUAGE - A low-level programming language capable of translating languages into machine language.

INPUT - Information submitted by the user into the program upon being prompted to do so.

STACK - A pile of values, piled in a similar manner to a pile of dishes.

POP - The removal of a value from the stack. The top value is removed first.

PUSH - The addition of a value to the stack. The added value is to be placed on top of the stack.

ASCII - Short for 'American Standard Code for Information Interchange'. To convert a character or symbol into ASCII means to get its numerical counterpart, so that it may be read by computers.

Part II – Functional Requirements

This part of the requirements document states in a detailed and precise manner what the application will do.

Statement of Functionality

This application emphasizes its four features of the following:

* *** Finding the maximum of a list of values**
* *** Finding the minimum of a list of values**
* *** Finding the average from a list of values**
* *** Assigning a grade per value, according to the following grading criteria:**
    * *** A = 90-100**
    * *** B = 80-89**
    * *** C = 70-79**
    * *** D = 60-69**
    * *** F = 0-59**

Optional functions may also be involved to further simplify and organize the program, ideally having the ability to do:

* *** Multiplication**
* *** Division**
* *** ASCII conversion**
* *** Combine two digits into a double-digit number**
* *** Pop and push operations for a stack**

The four main functions will do as such:

- MAXIMUM

- This function will find the highest value out of all values within the set. Each element within a set of numbers will be compared with each other to find the highest number. The highest value will proceed onto the next comparisons until a higher value succeeds it. The outcome will be the maximum value of the set of elements once the final comparison has been made, which shall be output.

- MINIMUM

- This function will work similarly to the maximum function, but instead for the lowest value. Each element within a set of numbers will be compared with each other to find the lowest number. The lowest value will

proceed onto the next comparisons until a lower value succeeds it. The outcome will be the minimum value of the set of elements once the final comparison has been made, which shall be output.

- AVERAGE

- This function will calculate the average. The average is calculated through dividing the sum of all elements by the total number of elements. This process involves first finding the sum of the set of values. A function for division may be established beforehand to conduct the division of the sum with the total number of elements in the set, as the process of division must be defined in the assembly language. The average is then found in the form of an output of the sum of all elements divided by the number of elements.

- LETTER LABEL

- This function will provide a letter label that corresponds to the focused element. In this case, the letter label will be based on the grading system of A-F, which is meant to correspond with the given scores of 0-100. The assigning of the letter grade must follow the rubric below:

*** A = 90-100**

*** B = 80-89**

*** C = 70-79**

*** D = 60-69**

*** F = 0-59**

The letter grade shall be output alongside the score.

The optional functions are listed here:

- MULTIPLICATION

- Due to the nature of the assembly language, multiplication is not so easily accessible. As such, a function for multiplication may be made. This function will play a significant role in the process of combining two digits into a double-digit number, which will further be elaborated on the optional function MULTIDIGIT.

- DIVISION

- Due to the nature of the assembly language, division is not so easily accessible. As such, a function for division may be made. This function is not emphasized in the objectives of the project, but it is an essential component that will make it possible to find the average of a set of values. Organizing the process of division into one function will allow for a cleaner program and easier access to division.

- ASCII

- The ASCII function's purpose is to convert the digits to their respective ASCII code. ASCII conversion will play a significant role in the calculations for each of the four central functions for finding the maximum, minimum, average, and respective grade.

- MULTIDIGIT

- Assembly language input takes only one value at a time, and as the goal is to take a set of double-digit values to evaluate its maximum, minimum, average, and respective grade, it will be necessary to combine two singular digits into a double-digit value. The double-digit value must be taken from each set of two values as input by the user.

- POP

- This feature is to pop, or remove, a value from a stack.

- PUSH

- This feature is to push, or add, a value to a stack.

In this section you state precisely what the application will do.

This part, more than anything else in the requirements document, spells out your contract with your customers. The application will include all functions listed here and will not include any of the functions not listed.

In this section you must use as precise language as you can since the developers will use it to code the application. When reviewing this part with other people you should pay extreme attention to removing any possibility for ambiguous interpretation of any of the requirements.

If your application has several distinct categories of users, you can list the requirements by user category. User categories may be defined in terms of their job title (clerk, manager, administrator), the frequency with which they will use the system (heavy or casual), the purpose for which they will use the system (operational decisions, long-term decisions), etc. If each category of users uses the system in its own way, structuring the requirements based on user category will make sense.

If your application deals with several kinds of real-world objects, you can list the requirements by object. For example, for a reservation system a booking is an important object, and you may want to list all requirements pertaining to bookings in one sub-section.

One of the most common approaches is to list the requirements by feature. For example, features of a word processing application are file management, formatting, editing, etc.

Scope

There are four main functions for the project and six side functions that each or collaboratively can help to realize one of the four main functions. In order, it is best to see the following through:

PHASE 1: Necessary Functions

MAXIMUM

MINIMUM

AVERAGE

LETTER LABEL

PHASE 2: Recommended Cleanup Functions

MULTIPLICATION

DIVISION

POP

PUSH

ASCII

MULTIDIGIT

The first phase of development should outline the four main functionalities of the program. The second phase of development should involve the construction of helpful cleanup functions that will simplify the program's code and contribute to its readability.

In this section you state what functionality will be delivered and in which phase.

You should include this section if your development consists of multiple phases. As an alternative to this section, you can note the planned project phase for each feature in the functionality statement section. Usually, it is better to include a separate scope section for easy reference and communication.

Performance

Each functionality listed in the statement of functionality should be capable of executing within a second or less. At

the heaviest, a portion of each function will reiterate at most ten times for each single-digit input. These iterations should run instantaneously and in a linear manner, however, and do not delay the program beyond a second.

In this section you describe any specific performance requirements.

You should be very specific and use numeric measures of performance. Stating that the application should open files quickly is not a performance requirement since it is ambiguous and cannot be verified. Stating that opening a file should take less than 3 seconds for 90% of the files and less than 10 seconds for every file is a requirement.

Instead of providing a special section on performance requirements, you may include the relevant information for each feature in the statement of functionality.

Usability

In this section you describe any specific usability requirements.

You need to include this section only if there are any “overarching” usability goals and considerations. For example, the speed of navigation of the UI may be such a goal. As in the previous section, use numeric measures of usability whenever possible.

Documenting Requests for Enhancements

There does come a time when the requirements for the initial release of your application are frozen. Usually, it happens after the system acceptance test which is the last chance for the users to lobby for some changes to be introduced in the upcoming release.

Currently, you need to begin maintaining the list of requested enhancements. Below is a template for tracking requests for enhancements.

Date	**Enhancement**	**Requested by**	**Notes**	**Priority**	**Release No/ Status**
05/26/24	Overflow management and storage allocation	Celeste Hernandez Mora	This has yet to be integrated into the idea of the program. It would be beneficial to the longevity and stability of the program upon integration.	Medium-priority	In progress (Planned for Phase 2)

Part III – Appendices

A significant part of the program involves the usage of an array. The array takes in inputs from the user and stores them by element. Ideally, the array should take in five double-digit values, which involves a total input of ten single-digit values. The array will be very helpful in managing these ten single-digit values and finding their double-digit forms upon combining through multiplication and addition.

Grade letters for each double-digit score should also be collected as one through the usage of pointers. Though arrays may be employed, a stack could also be put in use as grade letters will only be used for one occasion-output. Beyond that, there will be no modifications to grade letters.

Appendices are used to capture any information that does not fit naturally anywhere else in the requirements document yet is important. Here are some examples of appendices.

Supporting and background information may be appropriate to include as an appendix – things like results of user surveys, examples of problems to be solved by the applications, etc. Some of the supporting information may be graphical – remember all those charts you drew trying to explain your document to others?

Appendices can be used to address a specialized audience. For example, some information in the requirements document may be more important to the developers than to the users. Sometimes this information can be put into an appendix.

Flow chart or pseudo-code.

Include branching, iteration, subroutines/functions in flow chart or pseudocode.

```
Start program
Load PROMPT string
Output PROMPT string
Clear registers
Add 5 to R3 counter
Load SCORES array

For each iteration in LOOP:
    Get first digit
    Output first digit
    Convert digit to its ASCII counterpart
    Store digit into SCORES array
    ADD 1 to SCORES for next element
    Get second digit
    Output second digit
    Convert digit to its ASCII counterpart
    Store digit into SCORES array
    ADD 1 to SCORES for next element
    Load NEWL string
    Decrement counter in R3 by 1
    If counter is zero
        Go to NEXT
    If counter is positive
        Go to LOOP

From NEXT:
    Load RESX string
        Output
        Call MAX subroutine
        Output
    Load RESN string
        Output
        Call MIN subroutine
        Output
    Load RESG string
        Output
        Call AVG subroutine
        Output
    Load RESR string
        Output
        Call LTR subroutine
        Output

HALT the program
```

```
MAX subroutine:
    Clear registers
    ADD 10 to R2 counter
    Load SCORES to R1
LOOP2 Increment SCORES by value of R2
    Load SCORES[R2] to R0
    ADD R0 by R0
    Decrement counter in R3 by 1
    If zero:
        Go to CONT
    If positive:
        Go to LOOP2
CONT  ADD R0 to R3
    Load SCORES to R1
    Decrement R2 by 1
    ADD R2 to R1
    Load SCORES[R2] to R0
    ADD R3 by R0
    Subtract R3 by R4
        If zero:
            Go to PROCEED
        If positive:
            R4 = R3
        If negative:
```

```
                    Go to PROCEED
PROCEED     Clear registers R0, R1, R3
        Decrement R2 counter by 1
            If zero:
                    Go to ENDX
            If positive:
                    Go to LOOP2
ENDX  Clear R0
        ADD R4 to R0
        End of MAX subroutine


MIN subroutine:
        Clear registers
        ADD 10 to R2 counter
        Load SCORES to R1
LOOP3 Increment SCORES by value of R2
        Load SCORES[R2] to R0
        ADD R0 by R0
        Decrement counter in R3 by 1
        If zero:
            Go to CONT
        If positive:
            Go to LOOP2
CONT  ADD R0 to R3
        Load SCORES to R1
```

```
          Decrement R2 by 1
          ADD R2 to R1
          Load SCORES[R2] to R0
          ADD R3 by R0
          Subtract R3 by R4
                    If zero:
                              Go to PROCEED
                    If positive:
                              Go to PROCEED
                    If negative:
                              R4 = R3
PROCEED             Clear registers R0, R1, R3
          Decrement R2 counter by 1
                    If zero:
                              Go to ENDN
                    If positive:
                              Go to LOOP3
ENDN  Clear R0
          ADD R4 to R0
          End of MIN subroutine


AVG       Clear registers
          ADD 10 to R2 counter
          Load SCORES to R1
LOOP4 Increment SCORES by value of R2
```

```
          Load SCORES[R2] to R0
          ADD R0 by R0
          Decrement counter in R3 by 1
          If zero:
                    Go to CONT
          If positive:
                    Go to LOOP2
CONT  ADD R0 to R3
          Load SCORES to R1
          Decrement R2 by 1
          ADD R2 to R1
          Load SCORES[R2] to R0
          ADD R3 by R0
          ADD R4 by R3
          Decrement R2 counter by 1
                    If zero:
                              Go to STEP2
                    If positive:
                              Clear registers R0, R1, R3
                              Go to LOOP4
STEP2  Clear registers except R4
          ADD #-6 to R0
DIVLOOP       ADD #1 to R1
          ADD R0 to R4
          If zero:
                Go to FOUND
        If positive:
                Go to DIVLOOP
        If negative:
                ADD #6 to R0
                ADD #-1 to R1
                Go to FOUND
FOUND           Clear R0
        Add R1 to R0
        End of AVG subroutine


LTR subroutine:
        Clear registers
        ADD 10 to R2 counter
ASSIGN          Load SCORES to R1
        Increment SCORES by value of R2
        Load SCORES[R2] to R0
        ADD #-10 to R3
LOOP5 ADD R3 to R0
                If zero:
                        Go to FCHECK
                If positive:
                        Increment R4 by 1
                        Go to LOOP5
                If negative:
                    Go to FCHECK
            Decrement R2 by 2
FCHECK      ADD #-5 to R4
                If zero:
                    Assign 'F'
                    Go to REPEAT
                If negative:
                    Assign 'F'
                    Go to REPEAT
                If positive:
                    Go to DCHECK
DCHECK      ADD #-1
                If zero:
                    Assign 'D'
                    Go to REPEAT
                If positive:
                    Go to CCHECK
CCHECK      ADD #-1
                If zero:
                    Assign 'C'
                    Go to REPEAT
                If positive:
                    Go to BCHECK
BCHECK      ADD #-1
                If zero:
                    Assign 'B'
                    Go to REPEAT
          If positive:
                    Assign 'A'
                    Go to REPEAT
REPEAT    ADD #-2 to R2
          If zero:
                    Go to ENDL
          If positive:
                    Go to LOOP5
ENDL  End of LTR subroutine


Fill PROMPT with    "ENTER VALUES: \n"
Fill NEWL with           "\n              "
Fill RESX with      "\nMAX:           "
Fill RESN           "\nMIN:           "
Fill RESG           "\nAVG:           "
Fill RESR           "\nGRADE:    "


End Program
```