CMIS 315 Project 2 Gradebook

Development Approach Plan

Version 1.0

2/10/2015

Contents

[1. Introduction 1](#_Toc409960318)

[2. Overview 1](#_Toc409960319)

[3. Assumptions/Constraints/Risks 1](#_Toc409960320)

[3.1 Assumptions 1](#_Toc409960321)

[3.2 Constraints 1](#_Toc409960322)

[3.3 Risks 1](#_Toc409960323)

[4. Development Approach 1](#_Toc409960324)

[4.1 Development Methodology 1](#_Toc409960325)

[4.2 Lifecycle Management & Transition Approach 2](#_Toc409960326)

[4.3 Methods & Tools 2](#_Toc409960327)

[4.4 Difficulties Encountered 2](#_Toc409960328)

[4.5 Testing 2](#_Toc409960329)

[5. Weaknesses and Improvements 5](#_Toc409960330)

[5.1 Weaknesses 5](#_Toc409960331)

[5.2 Improvements 6](#_Toc409960332)

List of Tables

Table 1: Development Approach Processes ……………………………………………………2

Table 2: Test Data Test Run 1 ………………………………………………………………….3

Table 3: Test Data Test Run 2 ………………………………………………………………….4

List of Figures

[Figure 1: Screen Shot Test Run 1 3](#_Toc409960060)

[Figure 2: Screen Shot Test Run 2 5](#_Toc409960061)

# Introduction

This Development Approach Plan describes the approach for development of the Grade Book program. This document will outline the development approach, solution design, difficulties encountered, testing strategy, system weaknesses, and possible improvements. This document is written for Professor Omar Zevallos, CMIS 315 instructor.

# Overview

The purpose of this project was to create a program that uses input from the user to calculate and display the GPA for a given student. The program should allow the user to enter the student’s name, total number of grades to be entered, and letters for each grade. The program should prompt the user to input each letter grade, with a counter corresponding to each input iteration. The program should assign a numeric value to the letters A through D and F and have a default error output for any other character entered. Once all letter grades are entered the program should calculate and display the student’s name and GPA, and then ask the user if they would like to input data for another student. The program should repeat until the user chooses to exit and then the program should exit cleanly.

# Assumptions/Constraints/Risks

## Assumptions

None

## Constraints

The program must use the following values for letter grades:

A = 4, B = 3, C = 2, D = 1, F = 0. The program must have a default error message for all other values entered. The program must utilize at least one user-defined function.

## Risks

None

# Development Approach

## Development Methodology

I used the waterfall approach for software development for this project. I concluded this approach at the coding and testing phase, as the software will not be implemented, and operational support will not be needed. I used this methodology because:

1. The project requirements where clearly defined and unambiguous;
2. The requirements for this project are stable and not subject to change;
3. The project had clear objectives and solutions.

I divided the project into 7 parts:

1. GpaCalc.h class header
2. Menu.h class header
3. StudentRec.h class header
4. Menu.cpp class – displays welcome message
5. StudentRec.cpp class – Prompts the user to input student name, number of grades, and each letter grade.
6. GpaCalc.cpp class – calculates the students GPA using the grades total from the StudentRec class as a parameter along with the name to display output
7. Gradebook.cpp class – main class, creates instances of the other three classes and controls program flow.

## Lifecycle Management & Transition Approach

I started with the requirements definition phase, as there was no need for an initial investigation. Some of the requirements were defined in the project description; the rest became apparent during the writing of the initial algorithm. After defining all of the requirements I started with the basic system design by hand writing class diagrams and the basic algorithm. Once I had a clear picture of the system, I began to code the classes. Once each class was defined, I coded the main method. Once I had completed all coding I compiled and ran my program in the debugging mode to begin testing.

## Methods & Tools

Table 1: Development Approach Processes

| Process | Tools & Techniques |
| --- | --- |
| Requirements definition | Paper and Pencil |
| Class diagrams | Paper and Pencil |
| Coding | Microsoft Visual Studio 2013 |
| Testing | Microsoft Visual Studio 2013 |
| Approach Document Creation | Microsoft Word 2010, Microsoft Paint |
| Assignment Submission | Microsoft Word 2010, Notepad |

## Difficulties Encountered

I had a difficult time deciding on my exact approach to this program. At first, I thought about simply using a struct for student record data and defining methods for inputing data into the struct as well as calculating and outputting GPA within the body of the main method. This would have been a simple approach, but in my opinion would have made for clunky code with no reusability. My opinion is that modularity should always be used when possible in an effort to simplify code, and possibly create reusable classes and methods. I chose to make this program as modular as possible. In doing so I encountered some problems along the way. I am used to Java where it is easy to pass objects as parameters and has a difficult time when C++ didn’t want to behave the same way. I spent a long time going back through the reading and the videos about pointers, and then realized that the simplest solution was to write return methods for all necessary parameters and then call those methods in the main program to assign values to local variables and pass those as parameters. This may not be the “best” or most advanced way, but it worked, and I am satisfied with the results.

## Testing

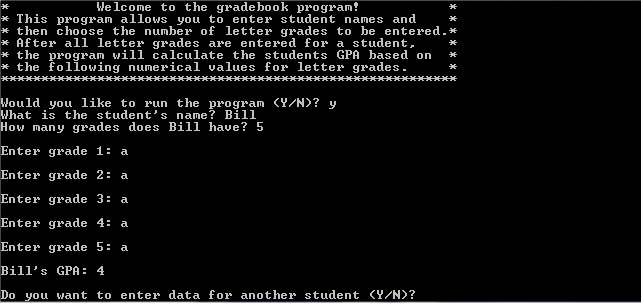
I began testing the program by first compiling the program in Microsoft Visual Studio 2013 and then running in the debugging mode. I chose Y to start the program and then entered a student name. I chose total number of grades to enter at which point the program calculated a GPA at an unusual number and asked if I would like to enter data for another student. The program would exit no matter if I chose Y or N. I proceeded to comment out all of my main method after the first step and then rebuilt and reran the program uncommenting one step at a time to see how the program behaved. I had to correct the placement of some of the code in a few of my class methods and then all worked as it should. After making the necessary corrections I ran the program in debugging mode again. The below table shows the test data input along with the expected and actual output.

Table 2: Test Data Test Run 1

| Input | Output |
| --- | --- |
| Start Debugging | Welcome Message Displayed. Prompted to start program (Y/N) |
| y | What is student’s name? |
| Bill | How many grades does Bill have? |
| 5 | Enter grade 1: |
| a | Enter grade 2: |
| a | Enter grade 3: |
| a | Enter grade 4: |
| a | Enter grade 5: |
| a | Bill’s GPA: 4  Do you want to enter data for another student (Y/N)? |

The below screen shot shows the output of this run.

Figure 1: Screen Shot Test Run 1

**

For the second test run I entered several incorrect data types to test my data validation methods. The below table shows the data entered during the second test run with the outputs.

Table 3: Test Data Test Run 2

| Input | Output |
| --- | --- |
| Start Debugging | Welcome Message Displayed. Prompted to start program (Y/N) |
| y | What is student’s name? |
| Dave | How many grades does Dave have? |
| 7 | Enter grade 1: |
| e | Error: invalid grade entered. Try again.  Enter grade 1: |
| q | Error: invalid grade entered. Try again.  Enter grade 1: |
| b | Enter grade 2: |
| c | Enter grade 3: |
| 5 | Error: invalid grade entered. Try again.  Enter grade 3: |
| $ | Error: invalid grade entered. Try again.  Enter grade 3: |
| d | Enter grade 4: |
| F | Enter grade 5: |
| A | Enter grade 6: |
| B | Enter grade 7: |
| C | Dave’s GPA: 2.14286  Do you want to enter data for another student (Y/N)? |

The below screen shot shows the output of this run.

Figure 2: Screen Shot Test Run 2

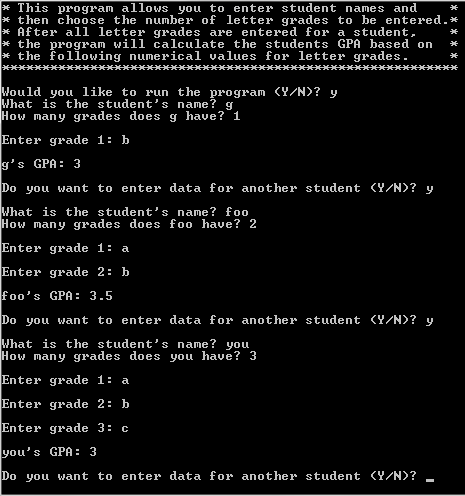
**

For the third test run I checked to see if the program would allow me to enter multiple students in a single run. The below table shows the data entered during the second test run with the outputs.

Table : Test Data Test Run 3

| Input | Output |
| --- | --- |
| Start Debugging | Welcome Message Displayed. Prompted to start program (Y/N) |
| y | What is student’s name? |
| g | How many grades does g have? |
| 1 | Enter grade 1: |
| b | g’s GPA: 3  Do you want to enter data for another student (Y/N)? |
| y | What is student’s name? |
| foo | How many grades does foo have? |
| 2 | Enter grade 1: |
| a | Enter grade 2: |
| b | foo’s GPA: 3.5  Do you want to enter data for another student (Y/N)? |
| y | What is student’s name? |
| you | How many grades does you have? |
| 3 | Enter grade 1: |
| a | Enter grade 2: |
| b | Enter grade 3: |
| c | you’s GPA: 3.5  Do you want to enter data for another student (Y/N)? |

The below screen shot shows the output of this run.



# Weaknesses and Improvements

## Weaknesses

The weaknesses that I found in the system are:

1. There is no data storage in the system. Each time the program is run the previous data is overwritten. There is no way to retrieve student data for later comparison.

## Improvements

To improve the program I would do the following:

1. Use arrays or vectors to store letter grade values and corresponding numeric values.
2. Use array to store student record objects to be able to compute average, median, high, and low GPA values for class as well as be able to output which student has the highest and lowest GPA’s, or which students were above or below the average or median value.
3. Use counters to determine total number of students and total number of grades entered, as well as total A’s, B’s, C’s etc.
4. Write output to a file.
5. Input from file.