**Program Specifications – Student Data Analysis**

Soumil Vavikar

Purdue University Global

IN501 – Fundamentals of Computer Programming

Dr. Mortoza Abdullah

January 31, 2024

**Program Specifications – Student Data Analysis**

This document contains the IPO chart and pseudocode for the project that has to be completed as part of the curriculum for the course “IN501 – Fundamentals of Computer Programming”. Per the project specification document, the program to be developed/written will be used to analyze students' data seeking either of the two courses - MSIT and MSCM. The program should be able to take user input for which operation needs to be performed, then do necessary calculations and display results in the expected format.

**Input Processing Output (IPO) Chart**

|  |  |  |
| --- | --- | --- |
| Input | Processing | Output |
| * User input 1: “Display average grade for all students.” * Default Input – STUDENTDATA.TXT | 1. Read the STUDENTDATA.TXT file line by line and treat each line as one student record. 2. Validate each record, convert each valid record into a student object, and put them in a student’s list. 3. Calculate average grades for all students. 4. Print the average grades for all the students to the console. | 1. All students' average grades are printed in the console. 2. Print the operations the user can perform. 3. Ask the user to input a valid option to perform the next operation. |
| * User input 2: “Display average grade for each program.” * Default Input – STUDENTDATA.TXT | 1. Read the STUDENTDATA.TXT file line by line and treat each line as one student record. 2. Validate each record, convert each valid record into a student object, and put them in a student’s list. 3. Separate student's lists into two lists, one for students pursuing MSIT and the other for students pursuing MSCM. 4. Calculate average grades for students in MSIT and MSCM. 5. Print the average grades for degree courses – MSIT and MSCM to the console. | 1. The average grades for students pursuing MSIT and MSCM are printed in the console. 2. Print the operations the user can perform. 3. Ask the user to input a valid option to perform the next operation. |
| * User input 3: “Display highest grade record.” * Default Input – STUDENTDATA.TXT | 1. Read the STUDENTDATA.TXT file line by line and treat each line as one student record. 2. Validate each record, convert each valid record into a student object, and put them in a student’s list. 3. Iterate through the list of students and find the record with the highest grade. 4. Print the record to the console. | 1. The record/student details with the highest grade are printed in the console. 2. Print the operations the user can perform. 3. Ask the user to input a valid option to perform the next operation. |
| * User input 4: “Display lowest grade record.” * Default Input – STUDENTDATA.TXT | 1. Read the STUDENTDATA.TXT file line by line and treat each line as one student record. 2. Validate each record, convert each valid record into a student object, and put them in a student’s list. 3. Iterate through the list of students and find the record with the lowest grade. 4. Print the record to the console. | 1. The record/student details with the lowest grade are printed in the console. 2. Print the operations the user can perform. 3. Ask the user to input a valid option to perform the next operation. |
| * User input 5: “Display students in MSIT.” * Default Input – STUDENTDATA.TXT | 1. Read the STUDENTDATA.TXT file line by line and treat each line as one student record. 2. Validate each record, convert each valid record into a student object, and put them in a student’s list. 3. Iterate through the list of students and collect the students pursuing degree courses MSIT in a list. 4. Print the list to the console. | 1. The list of students pursuing a degree in MSIT is printed in the console. 2. Print the operations the user can perform. 3. Ask the user to input a valid option to perform the next operation. |
| * User input 6: “Display students in MSCM.” * Default Input – STUDENTDATA.TXT | 1. Read the STUDENTDATA.TXT file line by line and treat each line as one student record. 2. Validate each record, convert each valid record into a student object, and put them in a student’s list. 3. Iterate through the list of students and collect the students pursuing degree courses MSCM in a list. 4. Print the list to the console. | 1. The list of students pursuing a degree in MSCM is printed in the console. 2. Print the operations the user can perform. 3. Ask the user to input a valid option to perform the next operation. |
| * User input 7: “Display all students in sorted order by student ID.” * Default Input – STUDENTDATA.TXT | 1. Read the STUDENTDATA.TXT file line by line and treat each line as one student record. 2. Validate each record, convert each valid record into a student object, and put them in a student’s list. 3. Sort the students' list based on the student\_id field in each student object within the list. 4. Print the sorted list on the console. | 1. The list of students is printed in the console in the sorted order by student\_id field. 2. Print the operations the user can perform. 3. Ask the user to input a valid option to perform the next operation. |
| * User input 8: “Display invalid records.” * Default Input – STUDENTDATA.TXT | 1. Read the STUDENTDATA.TXT file line by line and treat each line as one student record. 2. Perform validations and checks on each record. 3. Add the record/line to the invalid\_students list if any validations fail. Find the list of validations in the project specifications document. 4. If the invalid\_students list is not empty:    1. Print the invalid records on the console.    2. Write the invalid records to the “BADRECORDS.TXT.”    3. Print “BADRECORDS.TXT has been created.” to the console. | 1. Invalid student records are printed on the console. 2. “BADRECORDS.TXT” is generated. 3. “BADRECORDS.TXT has been created.” It is printed on the console. |
| User input 9: “Exit Program.” | Print “Exiting the Program.” Furthermore, exit the program. | Console output: “Exiting the Program.” |
| The user inputs any number or string other than a number ranging between 1 and 9 | 1. Validate the user input. 2. If the user input is not an integer between 1 and 9, print the error message: “Invalid menu option selected. Valid values are between 1 and 9. Please try again.” 3. Print the options and ask the user to input one between 1 and 9. | 1. Error message in the console: “Invalid menu option selected. Valid values are between 1 and 9. Please try again.” 2. Request the user to input again. |
| Either the STUDENTDATA.TXT is provided with no valid records, or it is not present. | 1. Check if the STUDENTDATA.TXT is present and has at least one valid record. 2. If the file is not provided or no valid records are present, print an error message on the console: “Exception occurred while reading the input file.” 3. Exit the program. | 1. An error message is printed on the console: “Exception occurred while reading the input file.” 2. The program is exited. |

**Pseudocode**

|  |
| --- |
| START  READ: Input file “STUDENTDATA.TXT” INTO records  ITERATE: records  IF record IS VALID  ADD to LIST valid\_student\_records.  ELSE IF record IS INVALID  ADD to LIST invalid\_records.  PRINT: “OPTIONS 1 – 9 from which user has to select the operation to perform.”  READ: action\_to\_perform  WHILE 9 != action\_to\_perform  IF “action\_to\_perform” IS NOT an integer OR IS NOT BETWEEN 1 and 9  PRINT: “Invalid menu option selected. Valid values are between 1 and 9. Please try again.”  READ: action\_to\_perform  ELSE IF 1 == action\_to\_perform  CALCULATE: Average grade for all students  PRINT: Average grade for all students: average\_grade  ELSE IF 2 == action\_to\_perform  CALCULATE: Average grade for degree course MSIT  PRINT: Average grade for degree MSIT: avg\_grade  CALCULATE: Average grade for degree course MSCM  PRINT: Average grade for degree MSCM: avg\_grade  ELSE IF 3 == action\_to\_perform  FIND: Student with the highest grades  PRINT: Student details with highest grades: student\_record  ELSE IF 4 == action\_to\_perform  FIND: Student with lowest grades  PRINT: Student details with lowest grades: student\_record  ELSE IF 5 == action\_to\_perform  FIND: Students in degree course MSIT  PRINT: Students details in degree MSIT: student\_records  ELSE IF 6 == action\_to\_perform  FIND: Students in degree course MSCM  PRINT: Students details in degree MSCM: student\_records  ELSE IF 7 == action\_to\_perform  SORT: valid\_student\_records based on the student\_id field  PRINT: Students list sorted by student\_id: sorted\_students  ELSE IF 8 == action\_to\_perform  PRINT: Invalid records in the input file: invalid\_records  WRITE: Write invalid records to “BADRECORDS.TXT.”  READ: action\_to\_perform    IF 9 == action\_to\_perform  PRINT: “Exiting the program.”  EXIT EXIT |

**Collaboration and Peer Review Details**

I collaborated with **Louis Gorski** for documentation and code skeleton peer review. We used email as our primary mode of communication and leveraged Zoom meetings for doing code and documentation walkthroughs and performing peer reviews.

Louis and I discussed our approach on a high level on Wednesday over email and met on Zoom meetings on Sunday morning, where we both did our code and documentation walk-through and demonstrated our working code. We shared feedback and review comments during the meeting. We then worked on the review comments and shared the project and document files for final review.

**Feedback Received**

Louis did an in-depth review and walkthrough of my code and documentation and gave me positive and constructive feedback.

The first feedback was to make menu option 8 more descriptive; I had “8 - Display invalid records” as the menu option, which printed invalid records, and I also wrote them to the BADDATA.TXT file (as discussed with the professor). He has asked to change the menu option to “8 - Display invalid records and create BADRECORDS.TXT file.” to make it clear that this option would also generate the BADDATA.TXT file.

The second feedback was to update the functions that printed the student records with the maximum grade and minimum grade to handle the scenario where there was more than one student with the maximum or minimum grade. Initially, the program made an inherent assumption that only one student record could have the highest grade or the lowest grade.

The third and final review comment was to read the input file once at the program's start instead of reading it every time the user selected a menu option. Initially, the file reading logic was inside the loop, which I moved out after the review comment.

**Feedback Given**

I did an in-depth review and walkthrough of Louis’s code and documentation and gave him positive and constructive feedback.

The first feedback was to update the IPO chart to be more descriptive and contain records for each possible user input option. Louis had done a good job initially as well, but the IPO chart felt more like an algorithm; hence, I asked him to tweak it.

The second feedback was to make minor changes to the flowchart to make it look more precise and correct some typing errors that were found.

The third feedback was implementing robust error handling in the code to handle use cases where the user input was invalid and the STUDENTDATA.TXT had no valid records.

The fourth feedback was to change the implementation of the functions printing the student records with maximum and minimum grades to have more efficient logic.

The fifth feedback was to implement the grading calculation (A to F) as mentioned in the project specifications for the initial menu options.

The sixth feedback was to write the BADDATA.TXT file at the exact location where the STUDENTDATA.TXT is read from to make it easier to run the program as only one location we need to mention the file location.

The final feedback was to move the main function to the bottom of the Python file, as mentioned in the assignment requirements.