College of Engineering, Construction and Living Sciences
Bachelor of Information Technology
ID511001: Programming 2
Level 5, Credits 15

Project 1 (C# Console App): Student Management System

Assessment Overview

In this assessment, you will design and develop a student management system **Console App** using **C#**.

Learning Outcomes

At the successful completion of this course, learners will be able to:

1. Build interactive, event-driven GUI applications using pre-built components.
2. Declare and implement user-defined classes using encapsulation, inheritance and polymorphism.

Assessments

Assessment	Weighting	Due Date	Learning Outcomes
Project 1: Student Management System	35%	20-09-2023 (Wednesday at 4.59 PM)	1 and 2
Project 2: Pong	25%	10-11-2023 (Friday at 04.59 PM)	1 and 2
Theory Examination	30%	15-11-2023 (Wednesday at 12.10 PM)	1 and 2
Classroom Tasks	10%	Multiple Due Dates	1 and 2

Conditions of Assessment

You will complete this assessment during your learner-managed time. However, there will be time during class to discuss the requirements and your progress on this assessment. This assessment will need to be completed by **Wednesday, 20 September 2023** at **4.59 PM**.

Pass Criteria

This assessment is criterion-referenced (CRA) with a cumulative pass mark of **50%** over all assessments in **ID511001: Programming 2**.

Authenticity

All parts of your submitted assessment **must** be completely your work. If you use code snippets from **GitHub**, **StackOverflow** or other online resources, you **must** reference it appropriately using **APA 7th edition**. Provide your references in the **README.md** file in your repository. Failure to do this will result in a mark of **zero** for this assessment.

Policy on Submissions, Extensions, Resubmissions and Resits

The school's process concerning submissions, extensions, resubmissions and resits complies with **Otago Polytechnic — Te Pūkenga** policies. Learners can view policies on the **Otago Polytechnic — Te Pūkenga** website located at https://www.op.ac.nz/about-us/governance-and-management/policies.

Submission

You **must** submit all app files via **GitHub Classroom**. Here is the URL to the repository you will use for your submission – https://classroom.github.com/a/xIHtZr71. Create a **.gitignore** and add the ignored files in this resource - https://raw.githubusercontent.com/github/gitignore/main/VisualStudio.gitignore. The latest app files in the **master** or **main** branch will be used to mark against the **Functionality** criterion. Please test before you submit. Partial marks **will not** be given for incomplete functionality. Late submissions will incur a **10% penalty per day**, rolling over at **5:00 PM**.

Extensions

Familiarise yourself with the assessment due date. Contact the course lecturer before the due date if you need an extension. If you require more than a week's extension, you will need to provide a medical certificate or support letter from your manager.

Resubmissions

Learners may be requested to resubmit an assessment following a rework of part/s of the original assessment. Resubmissions are to be completed within a negotiable short time frame and usually **must** be completed within the timing of the course to which the assessment relates. Resubmissions will be available to learners who have made a genuine attempt at the first assessment opportunity and achieved a **D grade (40-49%)**. The maximum grade awarded for resubmission will be **C-**.

Resits

Resits and reassessments **are not** applicable in **ID511001: Programming 2**.

Instructions

You will need to submit an app and documentation that meet the following requirements:

Functionality - Learning Outcomes 1 and 2 (50%)

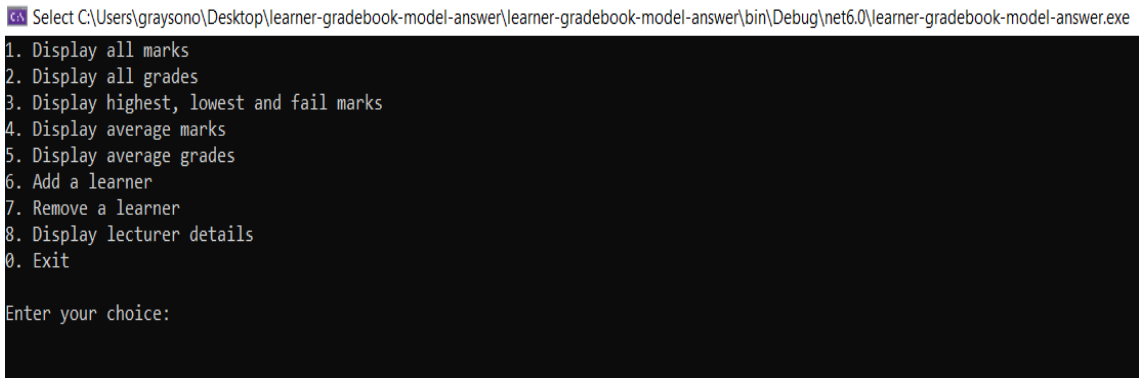
- The app **must** open without code or file structure modification in **Visual Studio**.
- Create the following classes:
  - **AssessmentMarks** which has the following field:
    - **assessmentMarks** of type **List<int>**
  - **Person** which is an **abstract** class and has the following fields and method:
    - **id** of type **int**
    - **firstName** of type **string**
    - **lastName** of type **string**
    - **DisplayDetails()** which is a **public abstract** method, has no arguments and returns a **string**
  - **Learner** which inherits from **Person** and has the following additional fields and method:
    - **prog1AssessmentMarks** of type **AssessmentMarks**
    - **prog2AssessmentMarks** of type **AssessmentMarks**
    - **DisplayDetails()** which is an **override** method, has no arguments and returns a **Learner's id**, **first name** and **last name**
  - **Lecturer** which inherits from **Person** and has the following additional fields and method:
    - **position** of type **string**
    - **salary** of type **int**
    - **DisplayDetails()** which is an **override** method, has no arguments and returns a **Lecturer's id**, **first name**, **last name**, **position** and **salary**
- Read a text file called **learners.txt** which contains information about five learners. This information includes **id**, **first name**, **last name**, three **ID510001: Programming 1 assessment marks** and three **ID511001: Programming 2 assessment marks**. **Note: learners.txt must** be located in the **bin/Debug** folder.
- Create a **List** of **Learner** objects and populate it with the information from the **learners.txt** file.
- Read a text file called **lecturers.txt** which contains information about three lecturers. This information includes **id**, **first name**, **last name**, **position** and **salary**. **Note: lecturers.txt must** be located in the **bin/Debug** folder.
- Create a **List** of **Lecturer** objects and populate it with the information from the **lecturers.txt** file.
- An **AssessmentMarks** object has several behaviours such as getting all marks, all grades, highest mark(s), lowest mark(s), fail mark(s), average marks and average grades. Create the following **public** methods in the **AssessmentMarks** class:
  - **GetAllMarks()** which has no arguments and returns a **List<int>**
  - **GetAllGrades()** which has no arguments and returns a **List<string>**
  - **GetHighestMarks()** which has no arguments and returns a **List<int>**
  - **GetLowestMarks()** which has no arguments and returns a **List<int>**
  - **GetFailMarks()** which has no arguments and returns a **List<int>**
  - **GetAverageMark()** which has no arguments and returns a **double**
  - **GetAverageGrade()** which has no arguments and returns a **string**
- A grade is calculated using the following grade table:

Grade	Mark Range
A+	90-100
A	85-89
A-	80-84
B+	75-79
B	70-74
B-	65-69
C+	60-64
C	55-59
C-	50-54
D	40-49
E	0-39

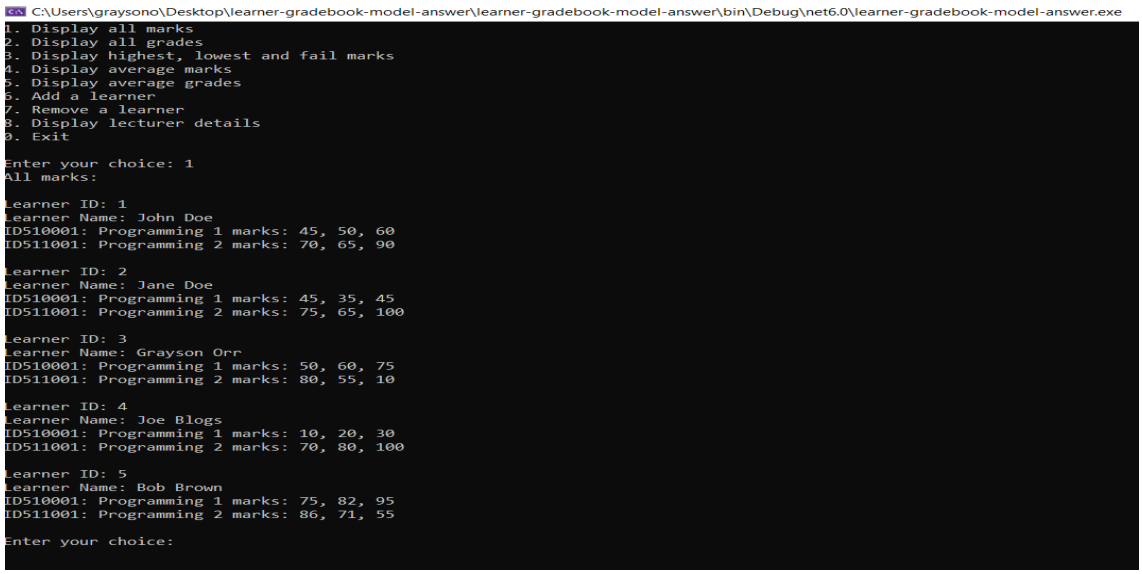
Note: The **lowest passing mark** is 50. If a learner achieves a mark of 50 in each assessment, i.e., 50, 50, 50 for a course, i.e., **ID510001: Programming 1** the **lowest passing mark** is 50 and the **highest passing mark** is 50.

- The app **must** display the following menu options:
  - 1. Display all marks
  - 2. Display all grades
  - 3. Display highest, lowest and fail marks
  - 4. Display average marks
  - 5. Display average grades
  - 6. Add a learner
  - 7. Remove a learner
  - 8. Display lecturer details
  - 0. Exit

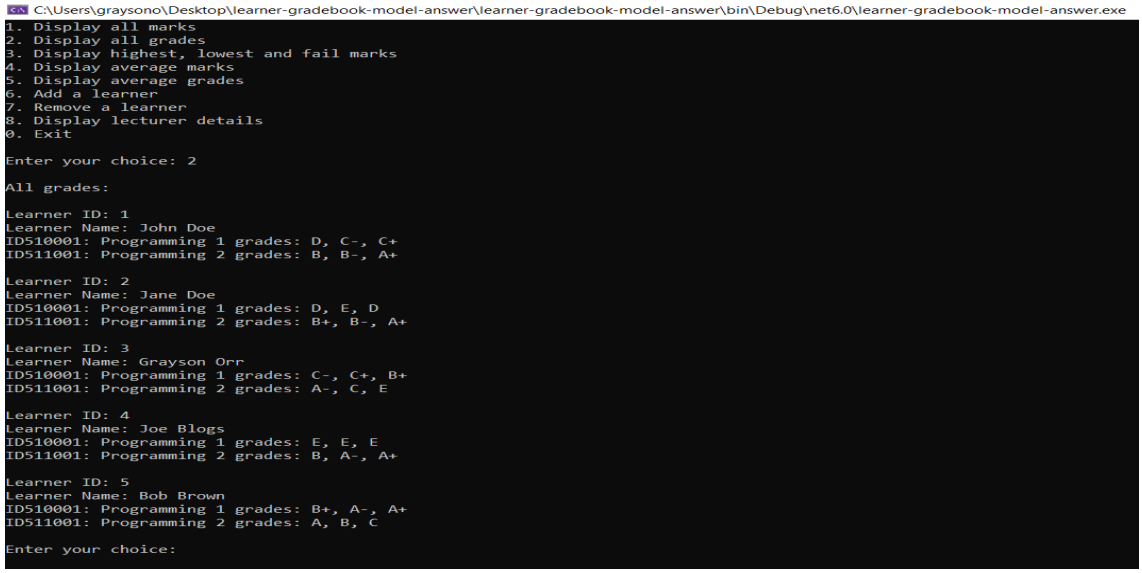
  Note: In the **Program.cs** file, you will need to create methods to achieve this functionality. Also, the app **must** be able to handle invalid user input. If the user enters an invalid option, a message **must** be displayed.



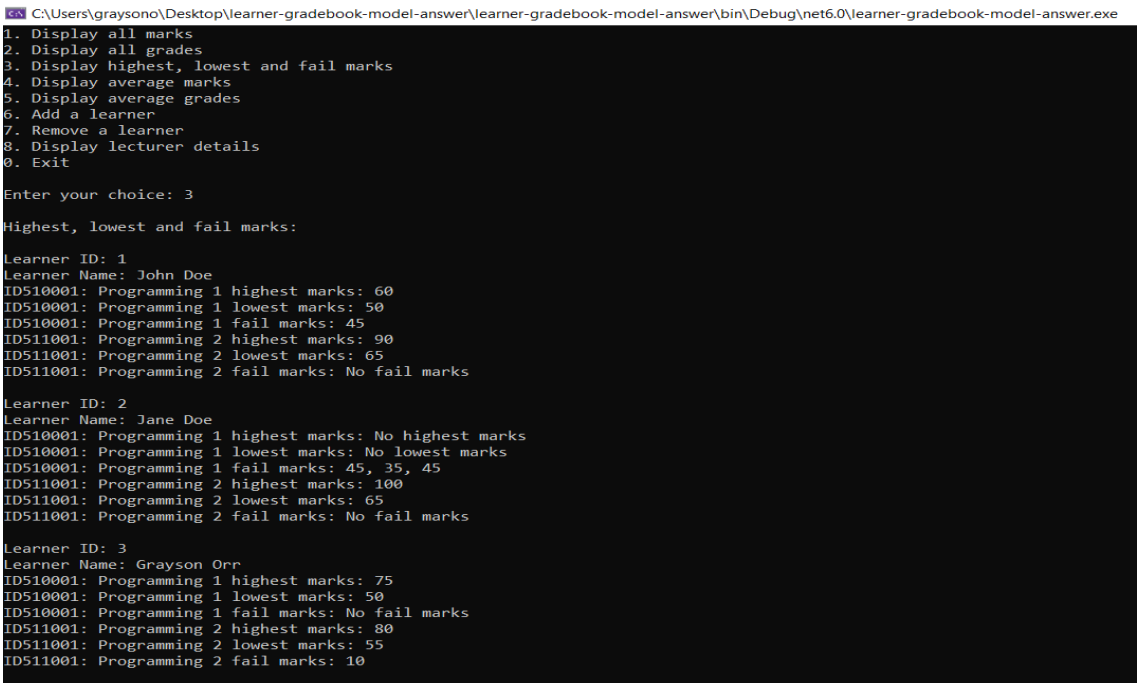
- When the user selects **1. Display all marks**, the app **must** display all marks for all learners. The marks **must** be displayed as follows:



- When the user selects **2. Display all grades**, the app **must** display all grades for all learners. The grades **must** be displayed as follows:

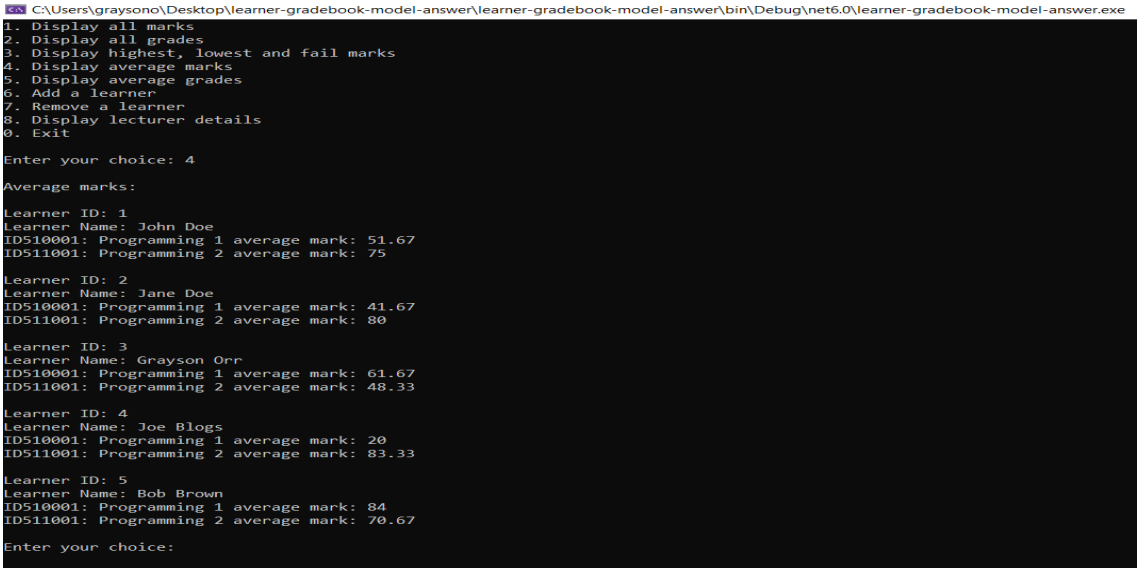


- When the user selects **3. Display highest, lowest and fail marks**, the app **must** display the highest, lowest and fail marks for all learners. The marks **must** be displayed as follows:

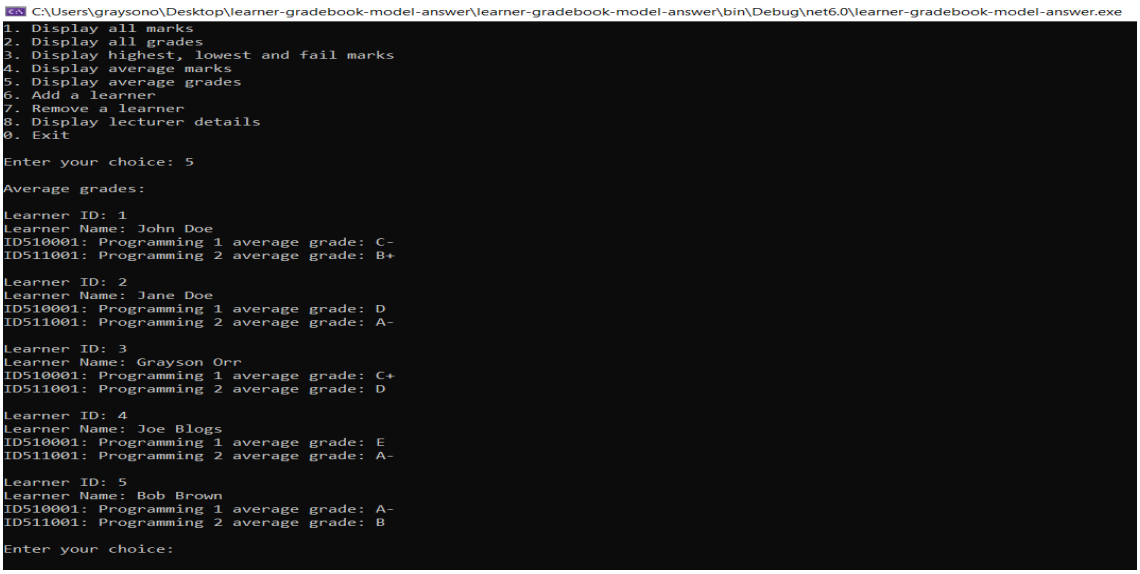


Note: If there is no fail mark(s), the **Fail marks: must** be displayed as **No fail marks**. If there is no lowest mark(s), the **Lowest marks: must** be displayed as **No lowest marks**. If there is no highest mark(s), the **Highest marks: must** be displayed as **No highest marks**.

- When the user selects **4. Display average marks**, the app **must** display the average marks for all learners. The marks **must** be displayed as follows:

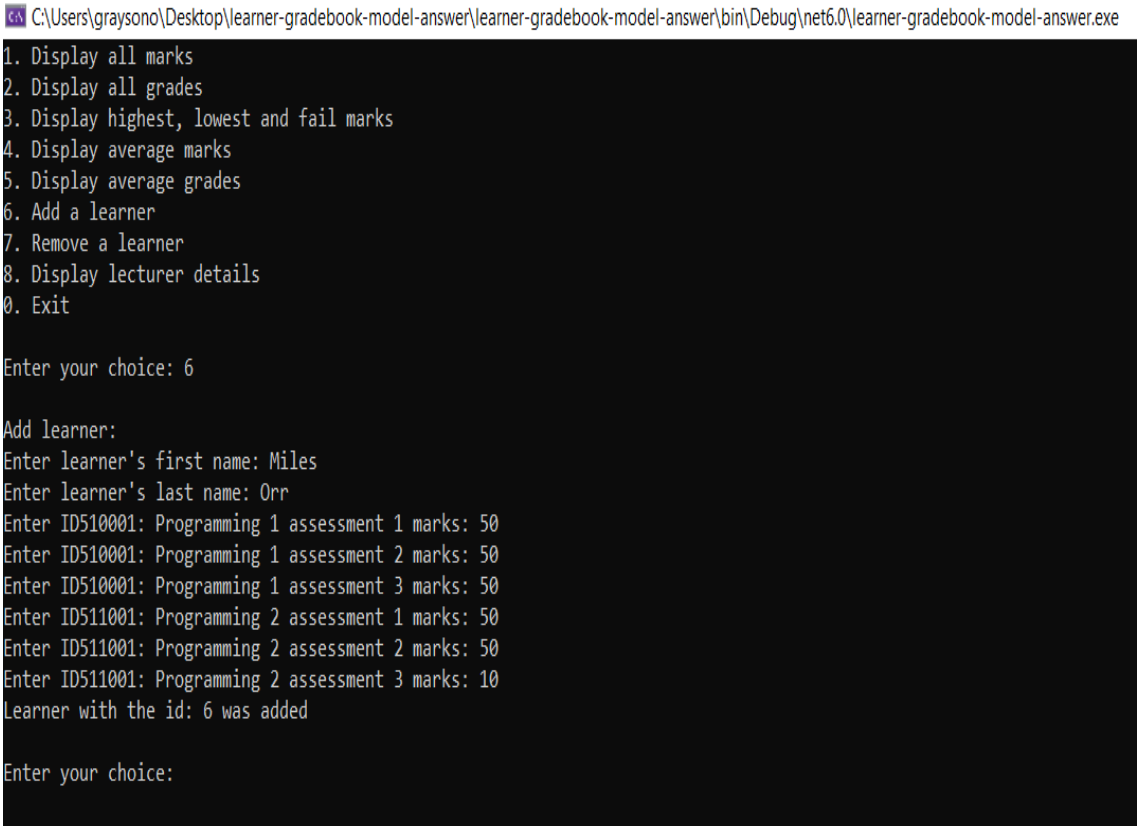


- When the user selects **5. Display average grades**, the app **must** display the average grades for all learners. The grades **must** be displayed as follows:

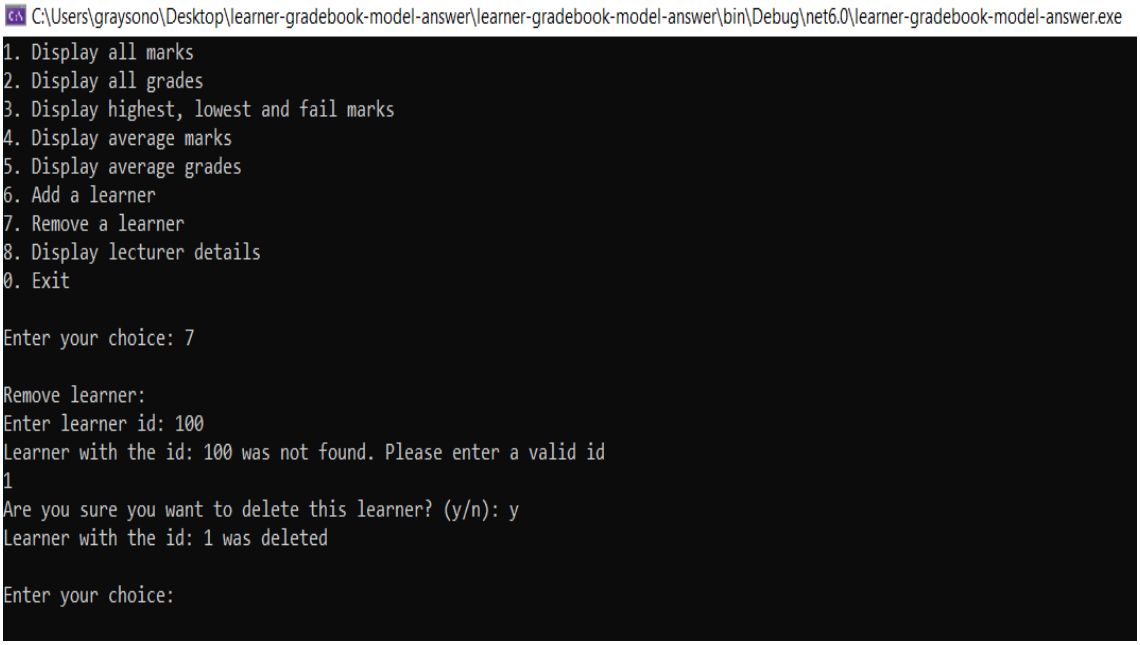


- When the user selects **6. Add a learner**, the app **must** prompt the user to enter the following information:
  - **First name**
  - **Last name**
  - **ID510001: Programming 1 assessment mark 1**
  - **ID510001: Programming 1 assessment mark 2**
  - **ID510001: Programming 1 assessment mark 3**
  - **ID511001: Programming 2 assessment mark 1**
  - **ID511001: Programming 2 assessment mark 2**
  - **ID511001: Programming 2 assessment mark 3**

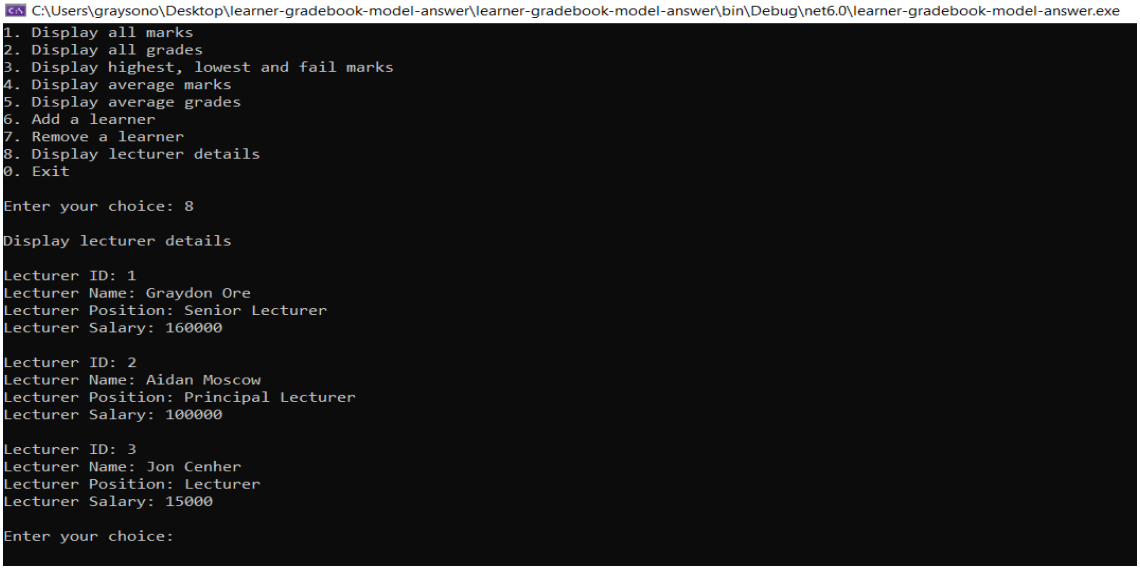
  Note: A **first name** and **last name must** not contain numbers or special characters. An **assessment mark must** be between **0** and **100**. If an **assessment mark** is invalid, an error message **must** be displayed. The learner's **id must** be generated automatically. However, the **id must** be unique. Append the learner's information to the **learners.txt** file.



- When the user selects **7. Remove a learner**, the app **must** prompt the user to enter the **id** of the learner to be removed. If the learner is found, the learner **must** be removed from the **List** of **Learner** objects. If the learner is not found, an error message **must** be displayed. **Note:** Remove the learner's information to the **learners.txt** file.



- When the user selects **8. Display lecturer details**, the app **must** display the lecturer's details. The details **must** be displayed as follows:



- When the user selects **0. Exit**, the app **must** display a thank you message and then exit after three seconds.
- 20 **unit tests** using **MSTest** which verify the functionality.

Code Elegance - Learning Outcomes 1 and 2 (40%)

- Adhere to the principles of **OO**.
- Appropriate naming of classes, fields and methods.
- Use of intermediate variables, constants and try-catch blocks.
- Idiomatic use of control flow, data structures and in-built functions.
- Efficient algorithmic approach.
- Sufficient modularity.
- Each class **must** have a header comment located immediately before its declaration.
- In-line comments where required.
- App files, i.e., **.cs** files are formatted.
- No dead or unused code.

Documentation and Git Usage - Learning Outcomes 1 and 2 (10%)

- Provide the following in your repository **README.md** file:
  - The app's class diagram created in **Visual Studio**. You must show all classes, fields, methods, properties and relationships.
  - How to run the **unit tests**.
  - Known bugs if applicable.
- Commit at least **20** times per week.
- Commit messages **must** be formatted using the convention discussed in **01-github-workflow-and-c#-basics** and reflect the context of each functional requirement change.