COMP1711 Coursework 1 (22/23)

1. Guidance

Programs must be implemented in C. You can develop your solutions on any suitable platform.

Programs must compile and run on the school linux network. It is your responsibility to test this before submission.

Work submitted is expected to be your own work. Discussion of the work is permitted. Sharing of your code is strictly not permitted. Please read the guidance on Academic Integrity.

Please use the labs for individual help with this work. You can post questions on Class Teams but please do not post code to the group. If you send individual Teams messages or email on the work, answers will be posted to the Class Teams area.

2. Submission of work

Work can be submitted from now until **10am on Wednesday 16th November**. Late penalties are applied after that date.

Work is submitted on Gradescope. You will get immediate feedback on only the compilation of your code. You will see a grade of 1 awarded if this is successful. If your code does not compile or you see a grade of 0 then compilation has failed and no further tests will run. You will receive an email confirming submission if it is successful. You can submit multiple times.

It is your responsibility to test your code so that it meets the specification.

Further tests will not run until after the deadline.

There are 2 parts to the work described below and 2 submission points on Gradescope.

3. Coursework brief

3.1 Outline

This program calculates an average grade for a student given a set of grades for individual courses.

- You will read in the set of student data from a text file.
- You will calculate an average grade for each student.
- You will write out the student data to a new text file.

For part 1 of this work you are told that there are 32 students in the class and 4 grades per student.

For part 2 of the work this data is provided as part of the input and can vary.

3.2 Skills tested

Procedural programming; Variables; Expressions; Loops; Conditionals; Arrays; File I/O

3.3 Part 1

You are provided with a template for the solution, **grader.c**. You should not rename that file or edit the print statements that are already included.

Your program will accept one command line argument – the name of the text file containing student grades.

The text data file provided, **grades.txt**, contains records for 32 students with 4 grades for each student.

Each line is one record: student_id grade_1 grade_2 grade_3 grade_4

- student_id is an integer in the range 2022000 to 2022099.
- An id with any other value is an error. Your program should print the appropriate error message and exit immediately.
- grade_i is an integer in the range 0 to 100.
- A grade with any other value is an error. Your program should print the appropriate error message and exit immediately.

Your code will import the student data and compute an average grade over all 4 modules for each student, with the following constraints:

- Any grade less than 20 should be corrected to 20 before averaging.
- Any grade greater than 90 should be corrected to 90 before averaging.
- The final average recorded should be rounded to the nearest integer.

Rounding of the raw grade is defined by: (grade < g+0.5) = g,   (grade >= g+0.5) = g+1

The program output must be a separate file 'averages.txt' which contains records for all 32 students.

- Each line is one record: student_id grade_average
- There is one blank space between the data values, and no other white space in the file
- student_id is in the same order as the input file
- grade_average is an integer

3.4 Part 2

This part is a modification of the Part 1 solution to a more general case. You should not complete this unless you have completed Part 1.

You are provided with a template for the solution , **grader2.c**. You should not rename that file or edit the print statements that are already included.

Your program will accept three command line argument – 2 integers (the number of students and grades, respectively) and the text file containing student grades.

You should allocate appropriate storage for the program data dynamically.

The text data file provided, **grades.txt**, contains records for students and grades.

Each line is one record: student_id grade_1 grade_2 …

- student_id is an integer in the range 2022000 to 2022099.
- An id with any other value is an error. Your program should print the appropriate error message and exit immediately.
- grade_i is an integer in the range 0 to 100.
- A grade of -1 indicates that the grade is not to be included in any average for the student
- A grade with any other value is an error. Your program should print the appropriate error message and exit immediately.

Your code will import the student data and compute an average grade over all included modules for each student, with the following constraints:

- Any grade less than 20 should be corrected to 20 before averaging.
- Any grade greater than 90 should be corrected to 90 before averaging.
- Grades of -1 are ignored in the averaging.
- The final average recorded should be rounded to the nearest integer.

The program output must be a separate file 'averages.txt' which contains records for all students.

- Each line is one record: student_id grade_average
- There is one blank space between the data values, and no other white space in the file
- student_id is in the same order as the input file
- grade_average is an integer

3.5 Testing

You are provided with a sample file of student data (grades.txt), the averages data file that your code should produce (averages.ref) and the output to the terminal when the program runs (output.ref) . You can compare your own program output to the files provided.

To capture output to the terminal you can redirect the program output to a separate file.

For example:

```
Part 1:   ./grader grades.txt > myoutput.txt

Part 2:   ./grader2 32 4 grades.txt > myoutput.txt
```

will redirect any program output from the terminal to the named file. Writing to/from a separate file is unaffected.

On the linux command line you can use the ‘diff’ utility to compare 2 files.

```
diff –y file1 file2
```

will compare file1 and file2 side by side and show differences.

It is important that you can replicate the results that are provided before submitting.

You should consider how to further test your code.

Further tests will be run to produce the final grade.

3.6 Coding advice

You are provided with a template C file for the solution of each part.

- You should add definitions for the problem data and appropriate code for each part.
- You should use only the output messages (printf) that are provided in the code. You will decide when these messages are triggered.
- Any further print statements you may use should be removed from your final submission as they will cause the grading to fail.

The template code breaks the task down into several stages:

- This is a standard design process for procedural programs.
- The first step is to define the data (variables and arrays) required for the problem.
- For the data processing you may wish to consider, on paper, what is involved in each stage before coding.
- You should write your code for each stage in sequence and verify each stage is complete and correct before moving on.
- Compile your code frequently to help with finding errors.

You must test your code compiles in a terminal window on our school linux machines.

3.7 Mark scheme (40 marks overall)

This work is 40% of the overall module assessment.

Automated tests will ensure your solution meets the specification as described above.

Code quality will consider the presentation of your code, appropriate use of comments and consistent style.

Part 1

Automated tests 16

Code quality and consistency 8

Part 2

Automated tests 12

Code quality and consistency 4