UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science
Faculty of Engineering, Built Environment & IT
University of Pretoria

COS132 - Imperative Programming

Practical 10 Specifications

Release Date: 19-05-2025 at 06:00

Due Date: 23-05-2025 at 23:59

Late Deadline: 25-05-2025 at 23:59

Total Marks: 614

Read the entire specification before starting with the practical.

Contents

1 General Instructions

- ***Read the entire assignment thoroughly before you begin coding.***
- This assignment should be completed individually.
- **Every submission will be inspected with the help of dedicated plagiarism detection software.**
- Be ready to upload your assignment well before the deadline. There is a late deadline which is 48 hours after the initial deadline which has a penalty of 20% of your achieved mark. **No extensions will be granted.**
- If your code does not compile, you will be awarded a mark of 0. The output of your program will be primarily considered for marks, although internal structure may also be tested (eg. the presence/absence of certain functions or structure).
- Failure of your program to successfully exit will result in a mark of 0.
- Note that plagiarism is considered a very serious offence. Plagiarism will not be tolerated, and disciplinary action will be taken against offending students. Please refer to the University of Pretoria's plagiarism page at `https://portal.cs.up.ac.za/files/departmental-guide/`.
- Unless otherwise stated, the usage of C++11 or additional libraries outside of those indicated in the assignment, will **not** be allowed. Some of the appropriate files that you have submit will be overwritten during marking to ensure compliance to these requirements. **Please ensure you use C++98**
- All functions should be implemented in the corresponding `cpp` file. No inline implementation in the header file apart from the provided functions.
- The usage of ChatGPT and other AI-Related software to generate submitted code is strictly forbidden and will be considered as plagiarism.

2 Overview

In this practical, you will implement the scenario discussed during Tutorial 9. Assume that for this scenario, in which you will be implementing an assessment analytics system, all of the information for the students who participated in the assessment is stored in a CSV file. The CSV file has the following columns:

```
first name, last name, student number, mark, did the preparation work
```

For the did the preparation work, `true` is indicated by a 't' and `false` is indicated by a 'f'. An example of the file is given below. The names and student numbers have been obfuscated:

```
Beqfkkn,Xtztsrtw,12265309,57.75,f
Tlckvzg,Mpdzmujh,47907523,22.28,t
Bnqfucy,Wxqpurwl,86266278,1.76,f
Sfzizsm,Gammnspj,60379200,27.51,t
```

Note that there are no spaces between the values and the commas.

3 Your Task:

You are required to implement all of the functions laid out in this section. You will be implementing two structs, a `StudentDetails` and `Assessment` struct. Each struct can be found in their own respective file, with a set of functions that utilise the struct.

Task	Mark
studentDetails	35
AsmConInsertDes	128
loadFromCSV	139
calculationFuncs	66
sortBestWorst	154
histogram	59
Testing	59

3.1 StudentDetails

This struct is used to model a student's details for a certain assessment. The struct has the following members and functions.

3.1.1 Members

- `firstName`
  - This stores the first name of the student.
- `lastName`
  - This stores the last name of the student.
- `mark`
  - This stores the marks of the student.
- `studentNumber`
  - This stores the student number of the student.
- `didPrepWork`
  - This stores whether the student did the prep work for the assessment or not.

3.1.2 Functions

- `constructor`
  - This function has a single parameter.
  - This function needs to create, populate and return an instance of the `StudentDetails` struct.
  - The format of the input is the same as is used in the CSV file.
  - The function needs to capitalise both the first name and last name of the `StudentDetails` instance.
- `constructor`
  - This function has multiple parameters.
  - This function needs to create, populate and return an instance of the `StudentDetails` struct.
  - Used the passed-in parameters to populate the struct.
  - The function needs to capitalise both the first name and last name of the `StudentDetails` instance.
- `destructor`
  - This function needs to deallocate the allocated memory for the passed in parameter, and set it to `NULL`.
  - If the passed-in parameter is already `NULL`, the function should simply return.
- `toString`
  - This function needs to return a string representation of the passed-in parameter.
  - The format of the string is the same one that is used by the CSV file.

3.2 Assessment

This struct is used to store all the information for an assessment that students needed to complete.

3.2.1 Members

- `roster`
  - This is a dynamic 2D array as discussed in Tutorial 9.
  - Each "row" in the 2D array represents a letter, which is the starting letter of a student's last name.
  - In other words, the array contains 26 rows, logically starting with the letter 'A' and ending with the letter 'Z'.
  - Each cell in the 2D array contains a pointer to a possible `StudentDetails` instance.

  - In other words the 2D array may contain "gaps".

- `numberOfStudents`
  - This is a dynamic 1D array containing the sizes of each "sub-array" in the `roster` member.
  - Thus, the 0^{th} index in the `numberOfStudents` array contains the maximum "storage space" for students whose last name starts with 'A'.

- `assessmentName`
  - This is the name of the assessment.

- `fullMarks`
  - This is the total marks that can be awarded for the assessment.

3.2.2 Functions

- `constructor`
  - This function needs to create, populate and return an instance of the `Assessment` struct.
  - Populate the appropriate members with the passed-in parameters.
  - Initialise the `roster` member to be of 26 rows and 0 columns (array of size 0).
  - *Hint: there is a difference between an array of size 0 and a pointer pointing to NULL.*
  - Initialise the `numberOfStudents` member to be of size 26 and filled with the appropriate value.
  - *Hint: think of the purpose of the values contained in the `numberOfStudents` member when determining the correct value to populate the array with.*

- `destructor`
  - This function needs to deallocate any memory allocated to the passed-in `Assessment` instance.
  - After deallocating all the memory (including the memory of the struct itself), set the passed-in member to `NULL`.
  - If the passed-in member is already set to `NULL`, the function should simply return.

- `insert`
  - This function needs to insert the passed-in `newStudent` into the first open space in the `roster` of the passed-in `assessment`.
  - If the sub-array that the student needs to be inserted into is too small, resize the array by 1 and insert the student at the end of the sub-array.
  - *Hint: Remember to increment the appropriate index in `numberOfStudents` if you do resize the sub-array.*

- `sort`
  - The sort function needs to sort each sub-array of students based on the marks they obtained.
  - The final order should be going from lowest mark to highest mark.
  - *Remember all the students should still be grouped based on the first letter of their last name.*
  - *Hint: Think of it as grouping the students based on the first letter of their last names, and then sorting each grouping based on the marks they achieved.*
  - If the passed-in assessment is not valid (is `NULL`, has a `roster` that is `NULL` or has a `numberOfStudents` that is `NULL`) the function should simply return.
- `totalNumberOfStudents`
  - This function needs to determine what the actual number of students is that took part in the passed-in assessment.
  - *Hint: Actual in this sense means the total number of students excluding the "gaps" in the sub-arrays.*
  - If the passed-in assessment is not valid (is `NULL`, has a `roster` that is `NULL` or has a `numberOfStudents` that is `NULL`) the function should simply return.
- `avg`
  - This function needs to calculate the average of the average of each sub-array.
  - In other words, the function needs to calculate what the average is for each sub-array, and then take the average of all the averages.
  - If the passed-in assessment is not valid (is `NULL`, has a `roster` that is `NULL` or has a `numberOfStudents` that is `NULL`), the total marks is 0, or the number of students that participated in the assessment is 0, the function should return 0.
- `numberThatCompletedPrep`
  - This function needs to determine the number of students who completed the preparation work for the assessment.
  - If the passed-in assessment is not valid (is `NULL`, has a `roster` that is `NULL` or has a `numberOfStudents` that is `NULL`), the total marks is 0, or the number of students that participated in the assessment is 0, the function should return 0.
- `passRate`
  - This function needs to determine the pass rate for the assessment, assuming the pass mark is 50%.
  - If the passed-in assessment is not valid (is `NULL`, has a `roster` that is `NULL` or has a `numberOfStudents` that is `NULL`), the total marks is 0, or the number of students that participated in the assessment is 0, the function should return 0.

- `distinction`
  - This function needs to determine the number of students who received a distinction for the assessment, assuming the mark for a distinction is 75%.
  - If the passed-in assessment is not valid (is `NULL`, has a `roster` that is `NULL` or has a `numberOfStudents` that is `NULL`), the total marks is 0, or the number of students that participated in the assessment is 0, the function should return 0.
- `fullMarks`
  - This function needs to determine the number of students who received full marks for the assessment.
  - If the passed-in assessment is not valid (is `NULL`, has a `roster` that is `NULL` or has a `numberOfStudents` that is `NULL`), the total marks is 0, or the number of students that participated in the assessment is 0, the function should return 0.
- `bestStudent`
  - This function needs to return the student who received the highest mark for the assessment.
  - If there are multiple students, return the student that was first encountered in the `roster`, when at position (0,0).
  - If the passed-in assessment is not valid (is `NULL`, has a `roster` that is `NULL` or has a `numberOfStudents` that is `NULL`), the total marks is 0, or the number of students that participated in the assessment is 0, the function should return `NULL`.
- `worstStudent`
  - This function needs to return the student who received the lowest mark for the assessment.
  - If there are multiple students, return the student that was first encountered in the `roster`, when at position (0,0).
  - If the passed-in assessment is not valid (is `NULL`, has a `roster` that is `NULL` or has a `numberOfStudents` that is `NULL`), the total marks is 0, or the number of students that participated in the assessment is 0, the function should return `NULL`.
- `loadFromCSV`
  - This function needs to populate the passed-in `assessment` with the contents of the passed-in CSV file.
  - It can be assumed that the file exists and that the contents of the file is valid.
  - Ensure that this function works correctly, as it will be the predominant method of populating the assessment during FitchFork evaluation.

- `marksHistogram`
  - This function needs to create a 2D `char` array with 11 rows and 26 columns.
  - The function then needs to plot the average mark of each sub-array on the histogram, by "colouring" in the bars, using 'X'.
  - Empty cells should contain the space (' ') character.
  - The last row (bottom-most row), contains the letters that represents each sub-array.
  - In order to determine the amount of "rows" you need to colour, the following function can be used:

$$row(avg) = \left\lfloor \frac{avg}{totalMarks} * 10 \right\rfloor$$

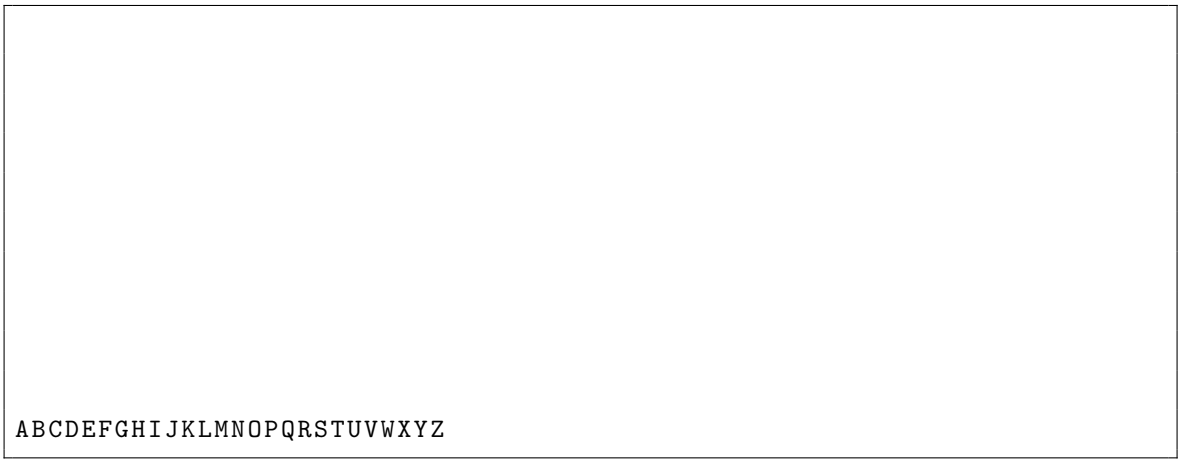
  - Example:
    - Assuming the total for the assessment was 100, the average and result of the *row* function of each letter is given below:

A: 67.5837 -> row(67.5837) = 6
B: 43.72 -> row(43.72) = 4
C: 43.61 -> row(43.61) = 4
D: 33.685 -> row(33.685) = 3
E: 52.64 -> row(52.64) = 5
F: 51.8033 -> row(51.8033) = 5
G: 30.112 -> row(30.112) = 3
H: 81.26 -> row(81.26) = 8
I: 47.9125 -> row(47.9125) = 4
J: 41.1875 -> row(41.1875) = 4
K: 29.5667 -> row(29.5667) = 2
L: 58.8167 -> row(58.8167) = 5
M: 44.4538 -> row(44.4538) = 4
N: 49.7125 -> row(49.7125) = 4
O: 62.135 -> row(62.135) = 6
P: 74.4275 -> row(74.4275) = 7
Q: 49.3333 -> row(49.3333) = 4
R: 79.71 -> row(79.71) = 7
S: 53.646 -> row(53.646) = 5
T: 37.466 -> row(37.466) = 3
U: 100 -> row(100) = 10
V: 57.026 -> row(57.026) = 5
W: 30.705 -> row(30.705) = 3
X: 37.2511 -> row(37.2511) = 3
Y: 37.7333 -> row(37.7333) = 3
Z: 8.33 -> row(8.33) = 0

    - Below is the resulting histogram that should be returned:

```
                    X
                    X
       X            X
       X        X X X
X      X       XX X X
X   XX X   X  XX XX XX
XXX XX XXX XXXXXXXX XX
XXXXXXXXXX XXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXX
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

* If the assessment is invalid, the following histogram should be returned:



4 Memory Management

As this practical has a heavy memory management focus, specialised tools will be used to evaluate you on your memory management skills.

The tool that will be used is `valgrind`. To determine the effectiveness of your memory management skills, the following command can be used:

```
valgrind --leak-check=full --keep-stacktraces=alloc-and-free ./main
```

This command assumes that the code was compiled to an executable `main`, and that the debugging flag was used during compilation.

5 Testing

As testing is a vital skill that all software developers need to know and be able to perform daily, 10% of the assignment marks will be allocated to your testing skills. To do this, you will need to submit a testing main (inside the `main.cpp` file) that will be used to test an Instructor Provided solution. You may add any helper functions to the main.cpp file to aid your testing. In order to determine the coverage of your testing the gcov [1] tool, specifically the following version *gcov (Debian 8.3.0-6) 8.3.0*, will be used. The following set of commands will be used to run gcov:

```
g++ --coverage *.cpp -o main
./main
gcov -f -m -r -j ${files}
```

This will generate output which we will use to determine your testing coverage. The following coverage ratio will be used:

$$\frac{\text{number of lines executed}}{\text{number of source code lines}}$$

We will scale this ration based on class size.

[1] For more information on gcov please see `https://gcc.gnu.org/onlinedocs/gcc/Gcov.html`

The mark you will receive for the testing coverage task is determined using Table 1:

Coverage ratio range	% of testing mark
0%-5%	0%
5%-20%	20%
20%-40%	40%
40%-60%	60%
60%-80%	80%
80%-100%	100%

Table 1: Mark assignment for testing

Note the top boundary for the Coverage ratio range is not inclusive except for 100%. Also, note that only the functions stipulated in this specification will be considered to determine your testing mark. Remember that your main will be testing the Instructor Provided code and as such, it can only be assumed that the functions outlined in this specification are defined and implemented.

As you will be receiving marks for your testing main, we will also be doing plagiarism checks on your testing main.

6 Implementation Details

- You must implement the functions in the header files exactly as stipulated in this specification. Failure to do so will result in compilation errors on FitchFork.
- You may only use **C++98**.
- You may only utilize the specified libraries. Failure to do so will result in compilation errors on FitchFork.
- Do not include using `namespace std` in any of the files.
- You may only use the following libraries:
  - <iostream>
  - <cmath>
  - <string>
  - <sstream>
  - <fstream>

7 Upload Checklist

The following C++ files should be in a zip archive named uXXXXXXXX.zip where XXXXXXXX is your student number:

- StudentDetails.cpp
- Assessment.cpp
- `main.cpp`
- Any textfiles used by your `main.cpp`

The files should be in the root directory of your zip file. In other words, when you open your zip file you should immediately see your files. They should not be inside another folder.

8 Submission

You need to submit your source files on the FitchFork website (`https://ff.cs.up.ac.za/`). All methods need to be implemented (or at least stubbed) before submission. Your code should be able to be compiled with the following command:

```
g++ -std=C++98 -Wall -Werror -g *.cpp -o main
```

and run with the following command:

```
./main
```

Remember your `h` file will be overwritten, so ensure you do not alter the provided `h` files.
You have 10 submissions and your final submission's mark will be your final mark. Upload your archive to the Practical 10 slot on the FitchFork website. Submit your work before the deadline. **No late submissions will be accepted!**