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 5 Specifications

Release Date: 17-03-2025 at 06:00

Due Date: 20-03-2025 at 23:59

There are no late submissions for this practical.

Total Marks: 70

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.**
- 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 create a simplistic testing framework to aid you with the testing of your code as well as debugging and correcting an extremely broken piece of code.

3 Background

A testing framework is a set of tools, libraries, and best practices that provide a structured way to automate and execute software tests. It helps developers and testers write, organize, and run tests efficiently while ensuring software quality and reliability. Testing frameworks typically support different testing types, such as unit testing, integration testing, and end-to-end testing.

4 Your Task:

You are required to implement all of the functions laid out in this section. You will notice that you will be implementing overloaded functions. As such, for the overloaded functions only the `int` function will be discussed but you will need to implement the `float` and `double` functions in the exact same method.

Task	Marks
1_printerTesters	6
2_isSameDifference	24
3_compareToTest	9
4_isEqualTest	10
5_relationTest	13
Testing	8

Table 1: Mark allocation

4.1 TesterHelpers

This file contains the testing framework you need to implement. Use these files in your testing code in the `main.cpp` file. You have only been supplied with the `Header` file and will need to create your own `Implementation` file.

4.1.1 Functions

- `void passedPrinter(int expected, int received)`
  - This function should print out the following string:

    ```
    Value <<receive>> was same as expected value of <<expected>>$
    ```

    where:

    * The name between the « and » is the parameter name.
    * $ refers to a new line.

- `void failedPrinter(int expected, int received);`
  - This function should print out the following string:

    ```
    Value <<receive>> was not the same as expected value of <<expected>>$
    ```

    where:

    * The name between the « and » is the parameter name.
    * $ refers to a new line.

- `bool isSame(int expected, int received)`
  - This function is used to determine if the two inputs are the same or not.
  - If the two inputs are the same, call the `passedPrinter` function and return `true`.
  - If the two inputs are not the same, call the `failedPrinter` function and return `false`.
- `bool isDifferent(int expected, int received)`
  - This function is used to determine if the two inputs are different or not.
  - If the two inputs are the same, call the `passedPrinter` function and return `false`.
  - If the two inputs are not the same, call the `failedPrinter` function and return `true`.
- `int compareTo(int expected, int received)`
  - This function is used to describe how the values differ in a bit more detail.
  - The following expression summarises the logic behind the function:

$$\mathrm{f}(x, y) = \begin{cases} 0, & \text{if } x = y \\ 1, & \text{if } x > y \\ -1, & \text{if } x < y \end{cases}$$

  - *Hint: The following page provides a good description of how to interpret the above expression https://www.mathsisfun.com/sets/functions-piecewise.html*

4.2 ValueTester

This file contains extremely erroneous code, which you will need to debug using the functions laid out in the `TesterHelpers` file. You will need to fix all of the errors in this file.

4.2.1 Functions

- `char isEqual(int valueA, int valueB)`
  - This function is used to describe if the two values are equal or not using symbols.
  - If the two values are equal, return '=' else return '!'.
- `char relation(int valueA, int valueB)`
  - This function is used to describe the relation between the two values using symbols.
  - If the two values are equal, return '='.
  - If `valueA` is greater than `valueB`, return '>'.
  - If `valueB` is greater than `valueA`, return '<'.

- `void printEqual(char symbol)`
  - This function should print out a message based on the provided symbol.

Symbol	Message
'='	The two values are equal
'!'	The two values are not equal
otherwise	An unknown symbol was passed

  - The string should end with an end line.
- `void printRelation(char symbol)`
  - This function should print out a message based on the provided symbol.

Symbol	Message
'='	The two values are equal
'>'	valueA is bigger than valueB
'<'	valueA is smaller than valueB
otherwise	An unknown symbol was passed

  - The string should end with an end line.

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.

The mark you will receive for the testing coverage task is determined using Table 2:

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

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 2: 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
  - string

7 Upload Checklist

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

- `ValueTester.cpp`
- `TesterHelpers.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 -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 20 submissions and your final submission's mark will be your final mark. Upload your archive to the Practical 5 slot on the FitchFork website. Submit your work before the deadline. **No late submissions will be accepted!**