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 2 Specifications

Release Date: 24-02-2025 at 06:00

Due Date: 28-02-2025 at 23:59

Late Deadline: 02-02-2025 at 23:59

Total Marks: 60

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

Up till now, you have been focusing on Algorithmic Thinking and the slow process of turning these algorithms into actual C++ programs. In this practical, you will be implementing a statistical program which will be able to provide a series of statistical analyses on a collection of data points.

Luckily, the majority of the "complex" programming has already been completed for you, and due to the power of separation of concerns, you do not need to know how this code works, only its purpose, pre-conditions and post-conditions. All of the "complex" programming follows a general algorithm.

Figure 1 illustrates the general algorithm.

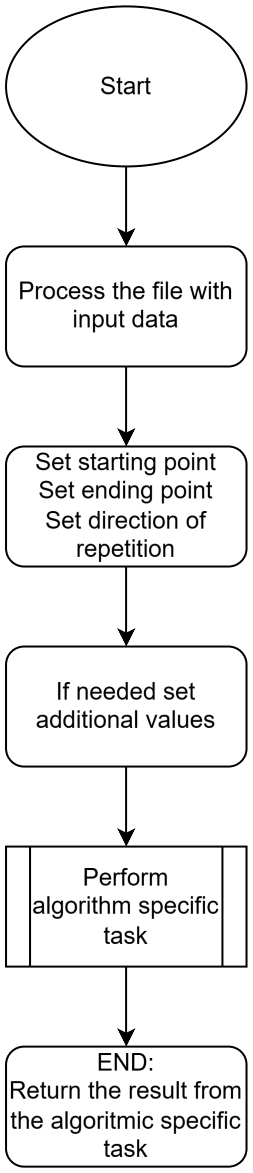
Start
Process the file with input data
Set starting point
Set ending point
Set direction of repetition
If needed set additional values
Perform algorithm specific task
END:
Return the result from the algoritmic specific task

Figure 1: Generic Algorithm

The data you will indirectly be working with is stored linearly in a structure that can be thought of as a table with a single row. For example, if the data looks like the following:

```
1
5
10
15
```

then the data can be thought of as structures represented below:

1	5	10	15

The position of the first data point in the structure is 0 and the position of the last data point is 3.

3 Your Task:

You are required to complete the functions in the `StatsInterface.cpp`, and `AlgorithmicConfiguration.cpp`. The other file, `StatsCalculator.cpp` is provided for you. **Do not try to understand what is happening in this file.** This whole practical can be completed with only the knowledge of what each function in the `StatsCalculator.cpp` does and **not** how exactly it is done.

Task	Marks
averageDataPoints	10
isValueInList	12
listOfPoints	10
standardDeviation	12
sumDataPoints	10
testing	6

Table 1: Marks breakdown

The remaining sections provide a description of each variable and function that is present in each file.

3.1 AlgorithmConfiguration

The following variables have been declared for you:

3.1.1 Variables

- `int START`
  - This variable indicates the position in the structure the algorithm will start at.
- `int END`
  - This variable indicates the position in the structure the algorithm will terminate at. In other words, when the algorithm reaches this position it will terminate.
- `bool MOVETONEXT`
  - This variable indicates if the algorithm will move from forward (i to $i+1$) in the structure, or backward (i to $i-1$) in the structure.
  - If set to `true`, the algorithm will move forward.
  - If set to `false`, the algorithm will move backward.
- `float AVERAGE`
  - This is a special variable that is only used in certain function(s) [1]

[1]See Section 3.2.

- `float VALUE`
  - This is a special variable that is only used in certain function(s) [2]

3.1.2 Functions

Table 2 contains descriptions of each of the functions you need to complete as part of this file. Note with the format that the _ indicates a single space, $ indicates a newline, and the variable between the `{` and `}` is the variable you need to print out. The examples in the table use the data points listed in Section 2.

Function	Description	Format	Example
`void printStart()`	This function should print out the `START` variable.	START:_{START}$	START: 0
`void printEnd()`	This function should print out the `END` variable.	END:_{END}$	END: 3
`void printMoveToNext()`	This function should print out the `MOVETONEXT` variable.	MOVETONEXT:_{MOVETONEXT}$	MOVETONEXT: 1
`void printAverage()`	This function should print out the `AVERAGE` variable.	AVERAGE:_{AVERAGE}$	AVERAGE: 7.75
`void printValue()`	This function should print out the `VALUE` variable.	VALUE:_{VALUE}$	VALUE: 2

Table 2: Function descriptions of the `AlgorithmConfiguration` file.

3.2 StatsCalculator

3.2.1 Variables

You will notice that there are **variables** declared in this file. Do not use/alter them, as it might negatively affect your marks on FitchFork.

3.2.2 Functions

Only the important functions in this file are described in Table 3. For the other functions, please see the comments in `StatsCalculator.h`.

[2]See Section 3.2.

Function	**Description**	**Pre-condition**	**Post-condition**
`void setFileName(std::string)`	This function sets the data files name	The data file exists	The correct variable is set
`void processFile()`	This function reads the contents of the data file and correctly sets the variables	The data file exists and the name has been set	The structure is populated with the contents of the data file.
`int getPositionOfLastDataPoint()`	This function returns the position of the last data point in the structure	The structure is populated.	Position of last data point is returned.
`float sumDataPoints()`	This function will return the sum of all the data points by starting at the `START` position and moving till it is at the `END` position, travelling in the direction specified by `MOVETONEXT`.	The data points have been loaded, and the correct `START`, `END`, and `MOVETONEXT` are specified.	The sum of all the values in the structure is returned.
`float average()`	This function will return the average of all the data points by starting at the `START` position and moving till it is at the `END` position, travelling in the direction specified by `MOVETONEXT`.	The data points have been loaded, and the correct `START`, `END`, and `MOVETONEXT` are specified.	The average of all the values in the structure is returned.
`float standardDiv()`	This function will return the standard deviation of all the data points by starting at the `START` position and moving till it is at the `END` position, travelling in the direction specified by `MOVETONEXT`. This function also requires that the `AVERAGE` variable is set with the average of all of the data points.	The data points have been loaded, and the correct `START`, `END`, `AVERAGE` and `MOVETONEXT` are specified.	The standard deviation of all the values in the structure is returned.
`string getPointsInOrder()`	This function will return a string representation of the data points starting at the `START` position and move till it is at the `END` position, travelling in the direction specified by `MOVETONEXT`.	The data points have been loaded, and the correct `START`, `END`, and `MOVETONEXT` are specified.	A string representation of all the values in the structure is returned.
`bool isValueInList()`	This function determines if the value, specified by the `VALUE` variable is contained in the list. The algorithm starts at the `START` position and moves till it is at the `END` position, travelling in the direction specified by `MOVETONEXT`. This function also requires that the `VALUE` variable is set with the passed-in value parameter.	The data points have been loaded, and the correct `START`, `END`, `VALUE` and `MOVETONEXT` are specified.	Whether the value is in the structure or not is returned.

Table 3: Function descriptions of the `StatsCalculator` file.

3.3 StatusInterface

In each function in this file, you will need to specify the appropriate variables (which is discussed in Section 3.1) according to the function details discussed in Table 3, such that the function's goal can be achieved. You will notice all of the functions have been set up for you with some of the functionality already implemented. You just need to set the correct variables.

- `float sumDataPoints()`
  - This function should return the sum of all the data points.
- `float sumDataPointsRev()`
  - This function should return the sum of all the data points, by traversing the structure in reverse.
- `float getAverageOfDataPoints()`
  - This function should return the average of all the data points.
- `float getAverageOfDataPointsRev()`
  - This function should return the average of all the data points, by traversing the structure in reverse.
- `float getStandardDeviation()`
  - This function should return the standard deviation of all the data points.
  - *Hint: Use the `STATSCALCULATOR::average` function to get the average, and **NOT** the `STATSINTERFACE` function.*
- `float getStandardDeviationRev()`
  - This function should return the standard deviation of all the data points, by traversing the structure in reverse.
  - *Hint: Use the `STATSCALCULATOR::average` function to get the average, and **NOT** the `STATSINTERFACE` function.*
- `string getListOfPointsInOrder()`
  - This function should return the string representation of all the data points.
- `string getPointsInReverseOrder()`
  - This function should return the string representation of all the data points, by traversing the structure in reverse.
  - In other words this should give the reverse result of the `getListOfPointsInOrder` function.

- `bool isTheValueInTheList(float value)`
  - This function should return if the passed in `value` is in the structure.
- `bool isTheValueInTheListRev(float value)`
  - This function should return if the passed in `value` is in the structure, by traversing the structure in reverse.

The last remaining functions in the file do not need to use any variables, instead, they need to print out the passed-in parameters using the format:

```
{message}{value}$
```

Note that in the format $ indicates a newline, and the variable between the `{` and `}` is the variable you need to print out.

4 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 [3] 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 4:

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

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

80%-100%	100%

Table 4: 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.

5 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 utilise 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>`
  - The `StatsCalculator` has its own list of imports which you do not need for the files you are required to implement.
- You are supplied with a **trivial** main demonstrating the basic functionality of this practical.

6 Upload Checklist

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

- `AlgorithmConfiguration.h`
- `AlgorithmConfiguration.cpp`
- `StatsCalculator.h`
- `StatsCalculator.cpp`

- `StatsInterface.h`
- `StatsInterface.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.

7 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 2 slot on the FitchFork website. Submit your work before the deadline. **No late submissions will be accepted!**