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

COS110 - Program Design: Introduction

Practical 10 Specifications

Release Date: 03-11-2025 at 06:00

Due Date: 06-11-2025 at 23:59

Late Deadline: 07-11-2025 at 00:59

Total Marks: 84

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 1 hour 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

The aim of this practical is to gain experience with the Stack Container in the C++ Standard Template Library.

3 Your Task:

You are required to implement the following class diagram illustrated in Figure 1. Pay close attention to the function signatures as the `h` files will be overwritten, thus failure to comply with the UML, will result in a mark of 0.

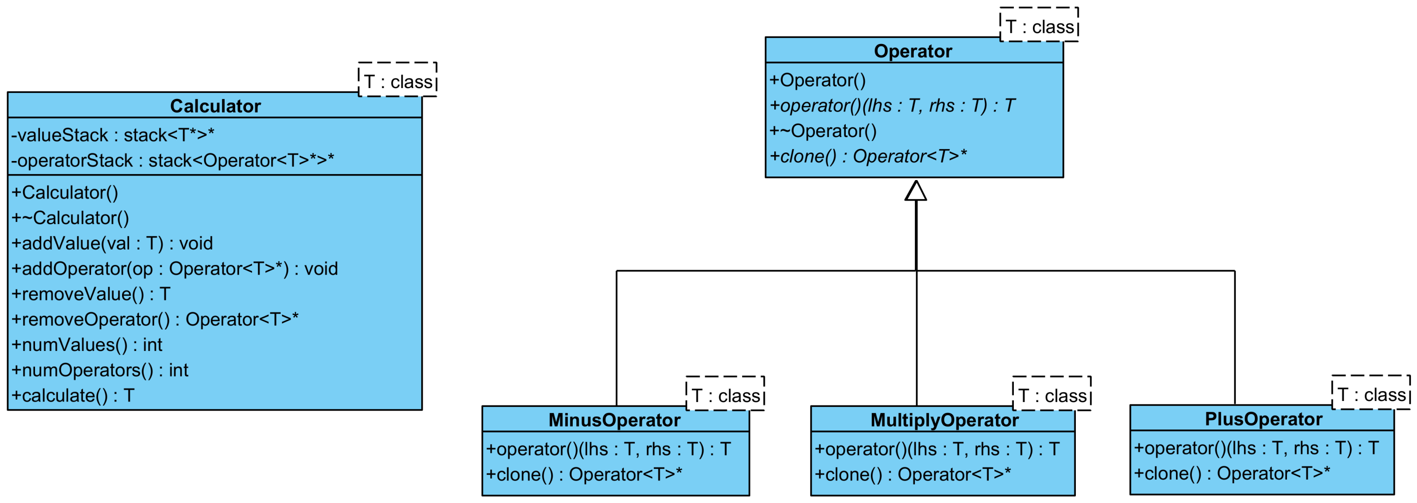

Figure 1: Class diagrams

Note, the importance of the arrows between the classes are not important for COS 110 and will be expanded on in COS 214.

3.1 Operator

This class will act as an abstract class for the all the operators that the calculator can utilize.

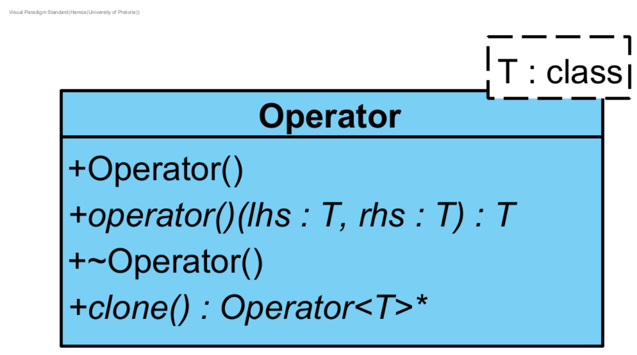

Figure 2: UML for Operator

3.1.1 Functions

- Function 1
  - This is the constructor for the operator class.
  - The body of this function should be left empty in the implementation file.
- Function 2
  - This is a pure virtual function and will be implemented in the derived classes.
- Function 3
  - This is a virtual function.
  - This is the destructor for the operator class.
  - The body of this function should be left empty in the implementation file.
- Function 4
  - This is a pure virtual function and will be implemented in the derived classes.

3.2 PlusOperator

This class will act as one of the operators that the calculator can utilize. This class publicly inherits from the Operator class.

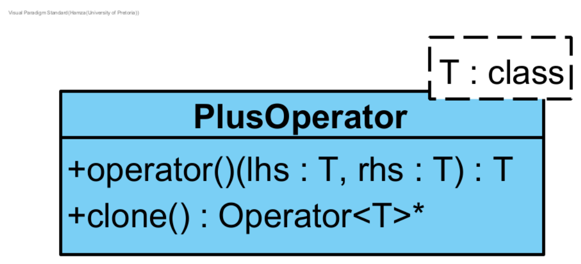
Visual Paradigm Standard(Hamza(University of Pretoria))
T : class
PlusOperator
+operator()(lhs : T, rhs : T) : T
+clone() : Operator<T>*

Figure 3: UML for PlusOperator

3.2.1 Functions

- Function 1
  - This function should return the sum of the two passed in parameters.
- Function 2
  - This function should return a pointer to a new PlusOperator object.

3.3 MinusOperator

This class will act as one of the operators that the calculator can utilize. This class publicly inherits from the Operator class.

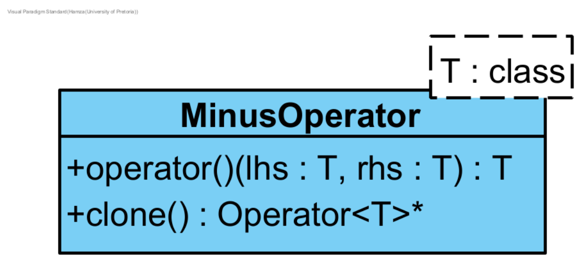
Visual Paradigm Standard(Hamza(University of Pretoria))
T : class
MinusOperator
+operator()(lhs : T, rhs : T) : T
+clone() : Operator<T>*

Figure 4: UML for MinusOperator

3.3.1 Functions

- Function 1
  - This function should return the difference of the two passed in parameters.
- Function 2
  - This function should return a pointer to a new MinusOperator object.

3.4 MultiplyOperator

This class will act as one of the operators that the calculator can utilize. This class publicly inherits from the Operator class.

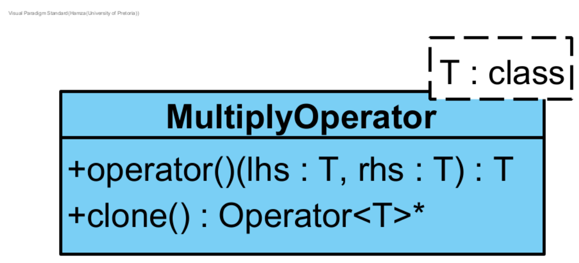
Visual Paradigm Standard(Hamza(University of Pretoria))
T : class
MultiplyOperator
+operator()(lhs : T, rhs : T) : T
+clone() : Operator<T>*

Figure 5: UML for MultiplyOperator

3.4.1 Functions

- Function 1
  - This function should return the product of the two passed in parameters.
- Function 2
  - This function should return a pointer to a new MultiplyOperator object.

3.5 Calculator

This class will use two stacks which will be used to implement a calculator. The stack class will be imported from the Standard Template library (STL).

Note: The interface for the STL Stack is very similar to the one you have seen in the slides. An important difference is that `pop()` does not return the removed element. Thus, you may need to access the top element before calling `pop()`.

Here is a link to the documentation for the stack class:
`https://cplusplus.com/reference/stack/stack/`

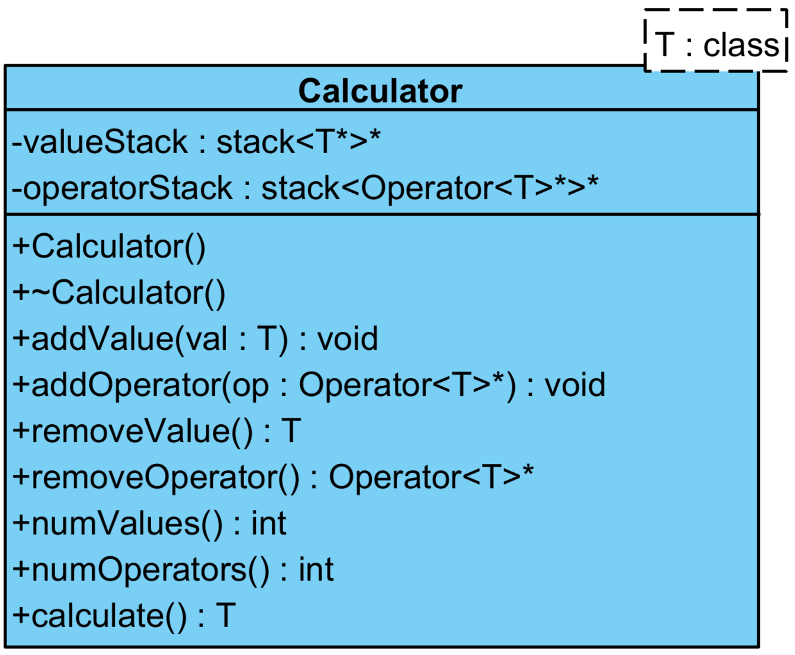

Figure 6: UML for Calculator

3.5.1 Members

- Member 1
  - This stack contains a pointer for each value that has been entered into the calculator.
- Member 2
  - This stack contains all the operators that have been entered into the calculator.

3.5.2 Functions

- Function 1
  - This is the constructor for the `Calculator` class.
  - This function should initialize both the `valueStack` and `operatorStack` by allocating dynamic memory for the member variables.
- Function 2
  - This is the destructor for the `Calculator` class.
  - This function should deallocate both the `valueStack` and `operatorStack`.
- Function 3
  - This function should dynamically allocate memory to store the passed-in parameter, then add it to the appropriate member variable.
- Function 4

  - This function should add the passed in parameter to the appropriate member variable.
  - A Deep copy of the passed in parameter should be made.

- Function 5
  - If the `valueStack` is empty, the function should throw the following string:
    "`Value stack is empty.`"
  - Otherwise, this function should remove the top element from the `valueStack` and return its stored value.

- Function 6
  - This function should remove the top operator from the `operatorStack` and return that operator.
  - If the `operatorStack` is empty, the function should return `NULL`.

- Function 7
  - This function should return the number of values in the `valueStack`.

- Function 8
  - This function should return the number of operators in the `operatorStack`.

- Function 9
  - This function will calculate the result from the values and operators in each respective stack.
  - If the `valueStack` is empty, this function should throw the following string:
    "`Value stack is empty.`"
  - The algorithm to achieve the final result is described below:

```
while (there are still operators remaining on the relevant
    stack)
  begin
      Pop the top values from the appropriate stack.
      Pop the top operator from the appropriate stack.
      Use the appropriate operator to calculate an
          intermediate result by using the two values and
          the operator.
      Push this intermediate result back onto the stack.
  end

  return the top value in the value stack.
```

  - If there are not enough values, this function should throw the following string:
    "`Not enough values to perform operation.`"

- **Example:** The following states of the `valueStack` and `operatorStack` should produce the following equation and results:

valueStack	operatorStack
1	
2	+
3	−

Table 1: Example of a valueStack and operatorStack state

$$(1 + 2) - 3 = 0 \tag{1}$$

4 Memory Management

As memory management is a core part of COS110 and C++, each task on FitchFork will allocate approximately 10% of the marks to memory management. The following command is used:

```
valgrind --leak-check=full ./main
```

Please ensure, at all times, that your code *correctly* de-allocates *all* the memory that was allocated.

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.
- You may include using `namespace std` in any of the files.
- You may only use the following libraries:
  - `stack`
  - `cstddef`
  - `iostream`
  - `string`
  - `sstream`

7 Upload Checklist

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

- `Operator.cpp`

- `PlusOperator.cpp`
- `MinusOperator.cpp`
- `MultiplyOperator.cpp`
- `Calculator.cpp`
- `main.cpp`
- Any textfiles used by your `main.cpp`
- `testingFramework.h` and `testingFramework.cpp` if you used these files.

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++ -Werror -Wall main.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 best mark will be your final mark. Upload your archive to the Practical 10 slot on the FitchFork website. If you submit after the deadline but before the late deadline, a 20% mark deduction will be applied. **No submissions after the late deadline will be accepted!**