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 3 Specifications: Operator overloading without dynamic memory

Release date: 21-08-2023 at 06:00

Due date: 25-08-2023 at 23:59

Total Marks: 130

Contents

1 General Instructions:

- *Read the entire assignment thoroughly before you start coding.*
- This assignment should be completed individually, no group effort is allowed.
- **To prevent plagiarism, every submission will be inspected with the help of dedicated software.**
- The only libraries/imports allowed for this assignment are ("<iostream>", "<fstream>", "<sstream>", "<stdlib.h>", "MyString.h"). Note you do not need all of these imports, however, any imports apart from these will result in a mark of 0
- Be ready to upload your assignment well before the deadline, as **no extension 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 classes).
- 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 http://www.ais.up.ac.za/plagiarism/index.htm.
- 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**
- We will be **overriding** your .h files, therefore do not add anything extra to them that is not in the spec

2 Plagiarism

The Department of Computer Science considers plagiarism as a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone else's work without consent, copying a friend's work (even with consent) and copying material (such as text or program code) from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to http://www.library.up.ac.za/plagiarism/index.htm (from the main page of the University of Pretoria site, follow the Library quick link, and then choose the Plagiarism option under the Services menu). **If you have any form of question regarding this, please ask one of the lecturers, to avoid any misunderstanding.** Also note that the OOP principle of code re-use does not mean that you should copy and adapt code to suit your solution.

3 Outcomes

The goal of this practical is to gain experience with working with the basics of operator overloading without dynamic memory.

4 Introduction

Implement the functions as described on the following pages.

5 Class Diagram

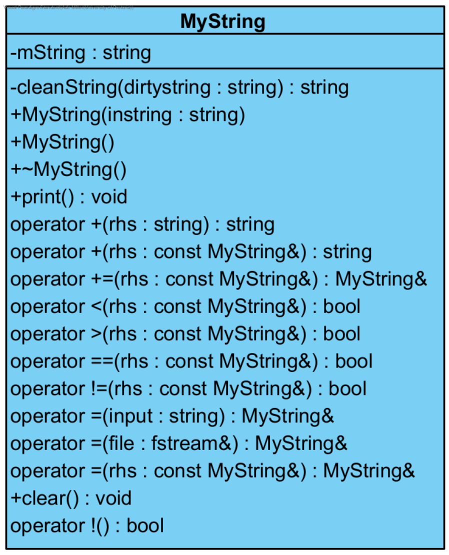
MyString
-mString : string
-cleanString(dirtystring : string) : string
+MyString(instring : string)
+MyString()
+~MyString()
+print() : void
operator +(rhs : string) : string
operator +(rhs : const MyString&) : string
operator +=(rhs : const MyString&) : MyString&
operator <(rhs : const MyString&) : bool
operator >(rhs : const MyString&) : bool
operator ==(rhs : const MyString&) : bool
operator !=(rhs : const MyString&) : bool
operator =(input : string) : MyString&
operator =(file : fstream&) : MyString&
operator =(rhs : const MyString&) : MyString&
+clear() : void
operator !() : bool

Figure 1: Class diagrams

6 Working with strings

The *<string>* library has been disallowed for this practical, this **does not mean you cannot use *std::string***, this means you cannot use string functions such as *getLine* and *strlen*, rather you will need to work through the strings in other ways. A suggested way is by checking ASCII values[1]. A few functions and techniques that can help that are not part of the string library are:

[1]check `https://www.computerhope.com/jargon/a/ascii.htm` if you are unfamiliar with ASCII

- .at(idx) -> returns the char at the index *idx* of the string.
- .size() OR .length() -> returns the length of the string.
- If you have a char variable named *myChar*, then *int(myChar)* will return the ASCII value of myChar.
- If you have an int variable named *myInt*, then *char(myInt)* will return the char equivalent of the ASCII value myInt.
- If you have an uppercase character, you can get its lowercase equivalent by subtracting 32 from its ASCII value.
- tolower() -> allows you to get the lowercase version of a char[2].
- Numbers are part of the ASCII table, however, they do not map to their numeric value, i.e. the ASCII value of "1" is 48.
- See `https://www.asciitable.com/` for the full ASCII table.

7 Comparison

Before implementing any of the comparison operators, ensure you **read** this section first.

- The comparison operator will only be called on strings that are "clean", meaning they consist **only** of letters and numbers.
- **Case does *not* matter**, this means the following comparisons are all true:
  - a<b
  - A<b
  - A<B
  - a==A
- Numbers can be present in the string, however, they are the lowest priority compared to the letters but still can be compared to letters and to each other, hence:
  - 0<1
  - 1<9
  - 9<a
- Length only matters if one string is the first part of another, then the longer string will be larger, i.e. while **aa<aa** is false, **aa<aaa** is true. This can include other letters as well, **aba<abaa** is true. Since case does not matter, **aBa<aba** is false, and **aBa<abaa** is true. However, in cases like **b<abbbb** or **a92bazzz<a93bR**, then length does not matter.
- Below are some examples that should help clarify the matter, all the following evaluate to **TRUE**.

```
a==a, 01a==01A, z>yaj, abcu>abce, K<z30, K30<z, zoey>abigal, metric>imperial
```

[2] see `https://www.geeksforgeeks.org/tolower-function-in-cpp/`

8 Classess

8.1 MyString

- Members
  - mString: string
    - A basic private string member variable.
- Functions
  - MyString()
    - This is the basic constructor for the class.
    - This method should initialize *mString* to the empty string.
  - cleanString(dirtyString: string): string
    - This function takes in a string and extracts the letters and numbers. It should return a new string that only contains letters and numbers from the passed in string.
    - If dirtyString is empty, or there are no letters or numbers, simply return an empty string.
    - For example, if we pass in the following string into the function (note that whitespace and tabs may be present in the string):

      ```
      aBC%$k$48J f {}d

          kl- D
      ```

      The function will then output the following string:

      ```
      aBCk48JfdklD
      ```
  - MyString(string: instring)
    - This is a constructor for the MyString class.
    - This should be set the *mString* member variable to the value of the *instring* parameter, after it has been sent through the *cleanString* function.
  - ~MyString()
    - This method should be present but left empty in the cpp file.
  - print():void
    - print *mString* to console followed by an endline.
  - operator+(rhs: string): string
    - This operator should return a string which is the concatenation of *mString* and the parameter *rhs*, after it has been sent through the *cleanString* function.
    - this function should not modify *mString*.
  - operator+(rhs: const MyString&): string
    - This operator should return a string which is the concatenation of *mString* and the *mString* member of *rhs*.
    - Neither the *mString* of the the current object or *rhs* should be modified.
  - operator+=(rhs: const MyString&): MyString&
    - This function should set *mString* of the current object to the concatenation of the current *mString* and the *mString* of *rhs*.

    - The operator should return a reference to *this*.

- operator<(rhs: const MyString&): bool
    - This function should return true if the current value of *mString* is "smaller than" the *mString* of rhs.
    - This comparison should be based on the rules listed in the Comparison section (section 7).
    - This function should not modify the current objects *mString* or the *mString* of *rhs*
- operator>(rhs: const MyString&): bool
    - This function should return true if the current value of *mString* is "larger than" the *mString* of rhs.
    - This comparison should be based on the rules listed in the Comparison section (section 7).
    - This function should not modify the current objects *mString* or the *mString* of *rhs.*
- operator==(rhs: const MyString&): bool
    - This function should return true if the current value of *mString* is "equal" the *mString* of rhs.
    - This comparison should be based on the rules listed in the Comparison section (section 7).
    - This function should not modify the current objects *mString* or the *mString* of *rhs.*
- operator!=(rhs: const MyString&): bool
    - This function should return true if the current *mString* is "not equal" the *mString* of rhs.
    - This comparison should be based on the rules listed in the Comparison section (section 7).
    - This function should not modify the current objects *mString* or the *mString* of *rhs.*
- operator=(input: const string): MyString&
    - This operator sets the value of *mString* to the result of *input* after it has been sent through the *cleanString* function.
    - The operator should return a reference to *this*.
- operator=(rhs: const MyString&): MyString&
    - This operator sets the value of *mString* to the *mString* value of *rhs.*
    - The operator should return a reference to *this*.
- operator=(file: fstream&): MyString&
    - This function accepts a filestream and sets *mString* to the contents of the file being read in.
    - It is possible that an invalid file/filestream is sent in[3]. If that is the case, do nothing, simply return the reference to *this.*
    - As with *operator=(instring: const string)*, you must only take the letters and numbers from the file.
    - Since you may not use the *<string>* library, you may not use *getline()*. You must use some other method for file reading such as *.get()*.
    - If the file is empty, or there are no letters or numbers, simply set *mString* to an empty string.
    - The operator should return a reference to *this*.

[3]Hint: The .fail() function can be used for this

- clear(): void
  - This function sets *mString* to the empty string.
- operator!(): bool
  - This operator will return true if *mString* is empty, meaning it has a length of 0.

9 Testing

As testing is a vital skill that all software developers need to know and be able to perform daily, 10% of the practical 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 the 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 [4] 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 MyString
```

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 1:

Coverage ratio range	% of testing mark	Final mark
0%-5%	0%	0
5%-20%	20%	6
20%-40%	40%	12
40%-60%	60%	18
60%-80%	80%	24
80%-100%	100%	30

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 function stipulated in this specification will be considered to determine your 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.

10 Upload checklist

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

- main.cpp
- MyString.cpp

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

Ensure the name of your h file is as follows, ("MyString.h"), **REMEMBER**, we overwrite your h 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.

11 Submission

You need to submit your source files, only the cpp files (**including the main.cpp**), on the FitchFork website (https://ff.cs.up.ac.za/). All methods must be implemented (or at least stubbed) before submission. Place the above-mentioned files in a zip named uXXXXXXXX.zip where XXXXXXXX is your student number. There is no need to include any other files or .h files in your submission. **Ensure your files are in the root directory of the zip file**. Your code should be able to be compiled with the C++98 standard
For this practical you will have 10 upload opportunities. Upload your archive to the Practical 3 slot on the Fitch Fork website.