Peter the Great St. Petersburg Polytechnic University

—

Institute of cybersecurity and information security

**GUIDELINES FOR LABORATORY WORK № 1**

**«String class»**

course «Object-Oriented Programming»

Saint-Petersburg

2020

**Content**

[1. Task 3](#_Toc59355645)

[2. Theory 4](#_Toc59355646)

[3. Basic task 5](#_Toc59355647)

[4. Additional tasks 10](#_Toc59355648)

[4.1. Unit tests 10](#_Toc59355649)

[4.2. Additional functionality for basic class 10](#_Toc59355650)

[4.3. Iterator pattern 10](#_Toc59355651)

[4.4. Python wrapping 11](#_Toc59355652)

[5. Report’s content 12](#_Toc59355653)

[6. Hints 13](#_Toc59355654)

1. Task

The main tasks of this laboratory work are:

* implement a class for representing a character string in C++, not using the STL library containers and algorithms;
* extend python interpreter functionality with the implemented class.

If you do additional tasks, the task list increases.

2. Theory

For the successful execution of the laboratory work it is necessary to study the following topics:

* the OOP paradigm;
* cons and pros of char arrays in C;
* the data model in python and its difference from C / C++;
* contents of the STL library;
* "const", "friend", " & " what is it and where and why to use it;
* operator overloading, inheritance;
* how the wrapping method works (which one you chose);
* the wrapping process, and the results (wrapping code).

You HAVE TO also read, understand, and remember:

* CODE STYLE.docx;
* REPORT STYLE.docx.

3. Basic task

During the execution of the task, you should follow the rules:

* do not use C functions when there is a replacement in C++ (for example, printf, malloc, realloc, etc.);
* do not duplicate the same code in different functions, you should see how one function depends on another or is based on another;
* if the input values of the function are incorrect, the behaviour of the function must be defined;
* class name is “MyString”;
* capacity changes downwards only in the shrink\_to\_fit() function;
* if you created an object and allocated memory for it and you no longer need it, delete it to avoid memory leaks.

For basic tasks, see the Table 1.

Table 1 — Basic class requirements

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Meaning** | **Usage example** | **Output** |
| Constructors | | | |
| Constructor() | default constructor | MyString a1;  std::cout<<a1<<" "<<a1.size()<<" "<<a1.capacity(); | 0 0 |
| Constructor(“char string”) | char array constructor | MyString a2("Hello world!");  std::cout<<a2<<" "<<a2.size()<<" "<<a2.capacity()<<std::endl; | Hello world! 12 13 |
| Constructor({‘s’,’t’,’r’,’i’,’n’,‘g’}) | initializer list constructor | MyString a3({'h','e','l','l','o'});  std::cout<<a3<<" "<<a3.size()<<" "<<a3.capacity()<<std::endl; | hello 5 6 |
| Constructor(std::string) | std::string constructor | std::string s="hello";  MyString a4(s);  std::cout<<a4<<" "<<a4.size()<<" "<<a4.capacity()<<std::endl; | hello 5 6 |
| Constructor(“char string”, count) | init class with count characters of “char string” | MyString a5("hello",4);  std::cout<<a5<<" "<< a5.size()<<" "<< a5.capacity()<<std::endl; | hell 4 5 |
| Constructor( count, char) | init class with count of characters | MyString a6(5,'!');  std::cout<<a6<<" "<< a6.size()<<" "<< a6.capacity()<<std::endl; | !!!!! 5 6 |
| Constructor(MyString) | copy constructor | MyString a7("hello");  MyString a8(a7);  std::cout<<a8<<" "<< a8.size()<<" "<< a8.capacity()<<std::endl; | hello 5 6 |
| Destructor() |  |  |  |
| Operators | | | |
| Operator +(Mystring) | concatenate with Mystring | MyString a1("hel"), a2("lo"), a3;  a3=a1+a2;  std::cout<<a1<<" "<<a2<<" "<<a3 <<std::endl; | hel lo hello |
| Operator +(“char string”) | concatenate with char array | MyString a4("hel"), a5;  a5=a4+"lo";  std::cout<<a4<<" "<<a5 <<std::endl; | hel hello |
| Operator +(std::string) | concatenate with std::string | MyString a6("hel"), a7;  std::string s1 = "lo";  a7=a6+s1;  std::cout<<a6<<" "<<a7<<std::endl; | hel hello |
| Operator +=(“char string”) | assignment concatenate with char array | MyString a1("hel"), a2("lo");  a1+=a2;  std::cout<<a1<<" "<<a2<<std::endl; | hello lo |
| Operator +=(std::string) | assignment concatenate with std::string | MyString a3("hel");  std::string s1="lo";  a3+=s1;  std::cout<<a3<<std::endl; | hello |
| Operator=(“char string”) | char string assignment | MyString a1;  a1="hello";  std::cout<<a1<<std::endl; | hello |
| Operator=(std::string) | std::string assignment | MyString a2;  std::string s1="hello";  a2=s1;  std::cout<<a2<<std::endl; | hello |
| Operator=(‘char’) | char assignment | MyString a3;  a3='!';  std::cout<<a3<<std::endl; | ! |
| Opearator[](int) | index operator | MyString a4;  a4="hello";  std::cout<<a4[2]<<std::endl; | l |
| Operator>()  Operator<()  Operator>=()  Operator<=()  Operator!=()  Operator==() | lexicographically comparing | MyString a, b;  a="abcd";  b="abce";  std::cout<<(a==b)<<(a!=b)<<(a>b)<<(a>=b)<<(a<b)<<(a<=b)<< std::endl; | See how the comparison works in std::string. |
| c\_str() | return a pointer to null-terminated character array |  |  |
| data() | return a pointer to array data that not required to be null-terminated | MyString a1("Hello world!");  std::string s1;  s1 = a1.data();  std::cout<<s1<<std::endl; | Hello world! |
| length() | same as size |  |  |
| size() | return the number of char elements in string | MyString a1("Hello world!");  std::cout<<s1.size()<<std::endl; | 12 |
| empty() | true if string is empty | MyString a1("Hello world!");  std::cout<<s1.empty()<<std::endl; | 0 |
| capacity() | return the current amount of allocated memory for array | MyString a1("Hello world!");  std::cout<<s1.capacity()<<std::endl; | 13 |
| shrink\_to\_fit() | reduce the capacity to size | MyString a1("Hello world!");  a1.erase(5, 6);  std::cout<<a1<<" "<<a1.size()<<" "<<a1.capacity()<<std::endl;  a1.shrink\_to\_fit();  std::cout<<a1<<" "<<a1.size()<<" "<<a1.capacity()<<std::endl; | Hello! 6 13  Hello! 6 7 |
| clear() | remove all char element in string | MyString a1("Hello world!");  std::cout<<a1<<std::endl;  a1.clear();  std::cout<<a1<<std::endl; | Hello world! |
| Operator<<(std::basic\_ostream), Operator>>(std::basic\_istream) |  | MyString a1("Hello world!");  std::cout<<a1<<std::endl; | Hello world! 12 13  0 13 |
| Insert | | | |
| insert(index, count, char) | insert count of char in index position | MyString a1 = "aaaaa";  a1.insert(0,1,'!');  std::cout<<a1<<std::endl;  a1.insert(3, 2, '@');  std::cout << a1 << std::endl; | !aaaaa  !aa@@aaa |
| insert(index, “string”) | insert null-terminated char string at index position | MyString a2 = "aaaaa";  a2.insert(1,"@@@@@");  std::cout<<a2<<std::endl; | a@@@@@aaaa |
| insert(index, “string”, count) | insert count of null-terminated char string at index position | MyString a3 = "aaaaa";  a3.insert(1,"@@@@@",2);  std::cout<<a3<<std::endl; | a@@aaaa |
| insert(index, std::string) | insert std::string at index position | MyString a4 = "aaaaa";  std::string s1 = "@@@@@";  a4.insert(1,s1);  std::cout<<a4<<std::endl; | a@@@@@aaaa |
| insert(index, std::string, count) | insert count of std::string at index position | MyString a5 = "aaaaa";  std::string s2 = "@@@@@";  a5.insert(1,s2,2);  std::cout<<a5<<std::endl; | a@@aaaa |
| Erase | | | |
| erase(index, count) | erase count of char at index position | MyString a1("Hello world!");  a1.erase(5, 6);  std::cout<<a1<<std::endl; | Hello! |
| Append | | | |
| append(count, char) | append count of char | MyString a1;  a1.clear();  a1.append(3,'!');  std::cout<<a1<<std::endl;  a1.append(3,'@');  std::cout<<a1<<std::endl; | !!!  !!!@@@ |
| append(“string”) | append null-terminated char string | MyString a2;  a2.clear();  a2.append("Hello ");  std::cout<<a2<<std::endl;  a2.append("world");  std::cout<<a2<<std::endl; | Hello  Hello world |
| append(“string”, index, count) | append a count of null-terminated char string by index position | MyString a3;  a3.clear();  a3.append("Hello world",0,6);  std::cout<<a3<<std::endl;  a3.append("Hello world",6,5);  std::cout<<a3<<std::endl; | Hello  Hello world |
| append(std::string) | append std:: string | MyString a4;  std::string s1="Hello ", s2="world";  a4.clear();  a4.append(s1);  std::cout<<a4<<std::endl;  a4.append(s2);  std::cout<<a4<<std::endl; | Hello  Hello world |
| append(std::string, index, count) | append a count of std:: string by index position | MyString a5;  std::string s3="Hello world";  a5.clear();  a5.append(s3,0,6);  std::cout<<a5<<std::endl;  a5.append(s3,6,5);  std::cout<<a5<<std::endl; | Hello  Hello world |
| Replace | | | |
| replace(index, count, “string”) | replace a count of char at index by “string” | MyString a1="hello amazing world";  a1.replace(6,7,"wonderful");  std::cout<<a1<<std::endl; | hello wonderful world |
| replace(index, count, std::string) | replace a count of char at index by std::string | MyString a2="hello amazing world";  std::string s1="wonderful";  a2.replace(6,7,s1);  std::cout<<a2<<std::endl; | hello wonderful world |
| Substr | | | |
| substr(index) | return a substring starts with index position | MyString a1 = "hello amazing world", a2;  a2=a1.substr(6);  std::cout<<a2<<std::endl; | amazing world |
| substr(index,count) | return a count of substring’s char starts with index position | MyString a3 = "hello amazing world", a4;  a4=a3.substr(6,7);  std::cout<<a4<<std::endl; | amazing |
| Find | | | |
| find(“string”) | if founded return the index of substring | MyString a="hello amazing world amazing";  int i;  i=a.find("amazing");  std::cout<<i<<std::endl; | 6 |
| find(“string”, index) | same as find(“string”) but search starts from index position | MyString a="hello amazing world amazing";  int i;  i=a.find("amazing", 7);  std::cout<<i<<std::endl; | 20 |
| find(std::string) | if founded return the index of substring | MyString a="hello amazing world amazing";  int i;  std::string s="amazing";  i=a.find(s);  std::cout<<i<<std::endl; | 6 |
| find(std::string, index) | same as find(“string”) but search starts from index position | MyString a="hello amazing world amazing";  int i;  std::string s="amazing";  i=a.find(s,7);  std::cout<<i<<std::endl; | 20 |

4. Additional tasks

Additional tasks should be done in order, otherwise points for them will not be counted.

4.1. Unit tests

You should learn about unit tests[[1]](#footnote-1) and implement unit tests for you class. Each implemented method should be covered by 2-3 test cases.

4.2. Additional functionality for basic class

You should make support basic std::exceptions and implement your own exception type for invalid conversion. substr(), replace(), erase(), insert() functions should throw an out\_of\_range exception. You should implement functions, which are described in the Table 2.

Table 2 — Additional functionality

|  |  |
| --- | --- |
| **Name** | **Meaning** |
| Constructor(&&) | move constructor |
| Constructor(0x1234568) | conversion constructor for integer types |
| Constructor(0.05) | conversion constructor for float types |
| Operator=&& | move assignment |
| Operator<<(std::basic\_ofstream)  Operator>>(std::basic\_ifstream) | file IO read and write |
| find() | Implement Aho-Corasick Algoritm |
| at(index) | return the index element of string if exist, otherwise throw an exception |
| to\_int() | perform conversion from string to integer types: char, short,int an other |
| to\_float() | perform conversion from string to float types: float, double and other |

4.3. Iterator pattern

You should learn about iterator and implement functions, which are described in the Table 3. You should support range-based loop for(auto &el: container). Also all methods, that takes index argument must have an iterator version (e. g. insert(index, “string”) -> insert(iterator, “string”)).

You have to implement 4 different types of iterator (see Table 3):

* iterator – simple forward iterator, goes from string beginning to it ending, can be used for string modification;
* const\_iterator – the same like iterator, but it can’t be used for string modification;
* reverse\_iterator – simple backward iterator, goes from string end to its beginning. Can be used for string modification;
* const\_reverse\_iterator – like reverse\_iterator, but it can’t be used for string modification.

Table 3 — Additional functionality — iterator

|  |  |
| --- | --- |
| **Name** | **Meaning** |
| iterator begin() | Returns an iterator pointing to the first character of the string |
| iterator end() | Returns an iterator pointing to the theoretical symbol AFTER the last character of the string |
| const\_iterator cbegin() | Returns const\_iterator to the first character of the string |
| const\_iterator cend() | Returns const\_iterator to the character AFTER the last character of the string |
| reverse\_iterator rbegin() | Returns reverse\_iterator to the first character of the string |
| reverse\_iterator rend() | Returns reverse\_iterator to the character AFTER the last character of the string |
| const\_reverse\_iterator rcbegin() | Returns const\_reverse\_iterator to the first character of the string |
| const\_reverse\_iterator rcend() | Returns const\_reverse\_iterator to the character AFTER the last character of the string |

4.4. Python wrapping

You should learn 2 or more ways python wrapping methods and extend python interpreter functionality with implemented class.

5. Report’s content

Your report must contain the following items, and MUST also be readable.

1. Goal.
2. Task.
3. Theory — you should briefly outline the theory you have learned to do your lab work.
4. Results — you should describe the process of work:
   1. the main idea of implementing your class;
   2. why your implementation is optimal;
   3. a description of the most interesting algorithms you have come up with;
   4. the process of wrapping python (what files are made, what is added to them and why);
   5. the results of testing in C++ and python (prepare a file with tests so that the teacher can quickly check the functionality of your class).
5. Additional task (optional).
6. Conclusion.
7. References — a properly designed bibliographic list of references that you used.
8. Appendix A — header file.
9. Appendix B — python wrapping code.
10. Appendix C…Z (optional) — whatever you want.

6. Hints

If you are not sure of your English language skills, try the "Grammarly" extension for the browser. The easiest way to draw a diagram is draw.io.

1. “Unit test your code” <https://docs.microsoft.com/en-us/visualstudio/test/unit-test-your-code?view=vs-2019> [↑](#footnote-ref-1)