1. Ministry of Science and Higher Education of the Russian Federation
2. Peter the Great St. Petersburg Polytechnic University
3. —
4. **Cybersecurity and Information Security department**

**REPORT**

**FOR Laboratory work № 1**

**«String class»**

subject «OOP»

1. Completed by

student from gr. 5151003/10802 <*signature*> K. K. Galkin

1. Checked by

teacher’s assistant <*signature*> E. V. Zavadksy

Saint – Petersburg

2023

# Content

[1 Content 2](#__RefHeading___1)

[2 Goal 3](#__RefHeading___2)

[3 Task 3](#__RefHeading___3)

[3.1 Restrictions 3](#__RefHeading___4)

[4 Theory 4](#__RefHeading___5)

[5 Results 7](#__RefHeading___6)

[5.1 Creating constructors 7](#__RefHeading___7)

[5.2 Methods 9](#__RefHeading___8)

[5.3 MyString::insert 10](#__RefHeading___9)

[5.4 MyString::erase 11](#__RefHeading___10)

[5.5 MyString::replace 12](#__RefHeading___11)

[6 Additional tasks 12](#__RefHeading___12)

[6.1 Unit-tests 12](#__RefHeading___13)

[6.1.1 Cmake build system 12](#__RefHeading___14)

[6.1.2 Unit-test code syntax 13](#__RefHeading___15)

[6.2 Additional methods 15](#__RefHeading___16)

[6.2.1 New structures 15](#__RefHeading___17)

[6.2.2 Conversion constructor 15](#__RefHeading___18)

[6.2.3 to\_int and to\_float methods 16](#__RefHeading___19)

[6.2.4 Filestream operator overloading 17](#__RefHeading___20)

[7 Conclusion 18](#__RefHeading___21)

# Goal

The goal of the work is to study the basic principles of OOP, as well as to practice the acquired knowledge on the basis of creating your own class

# 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.

## Restrictions

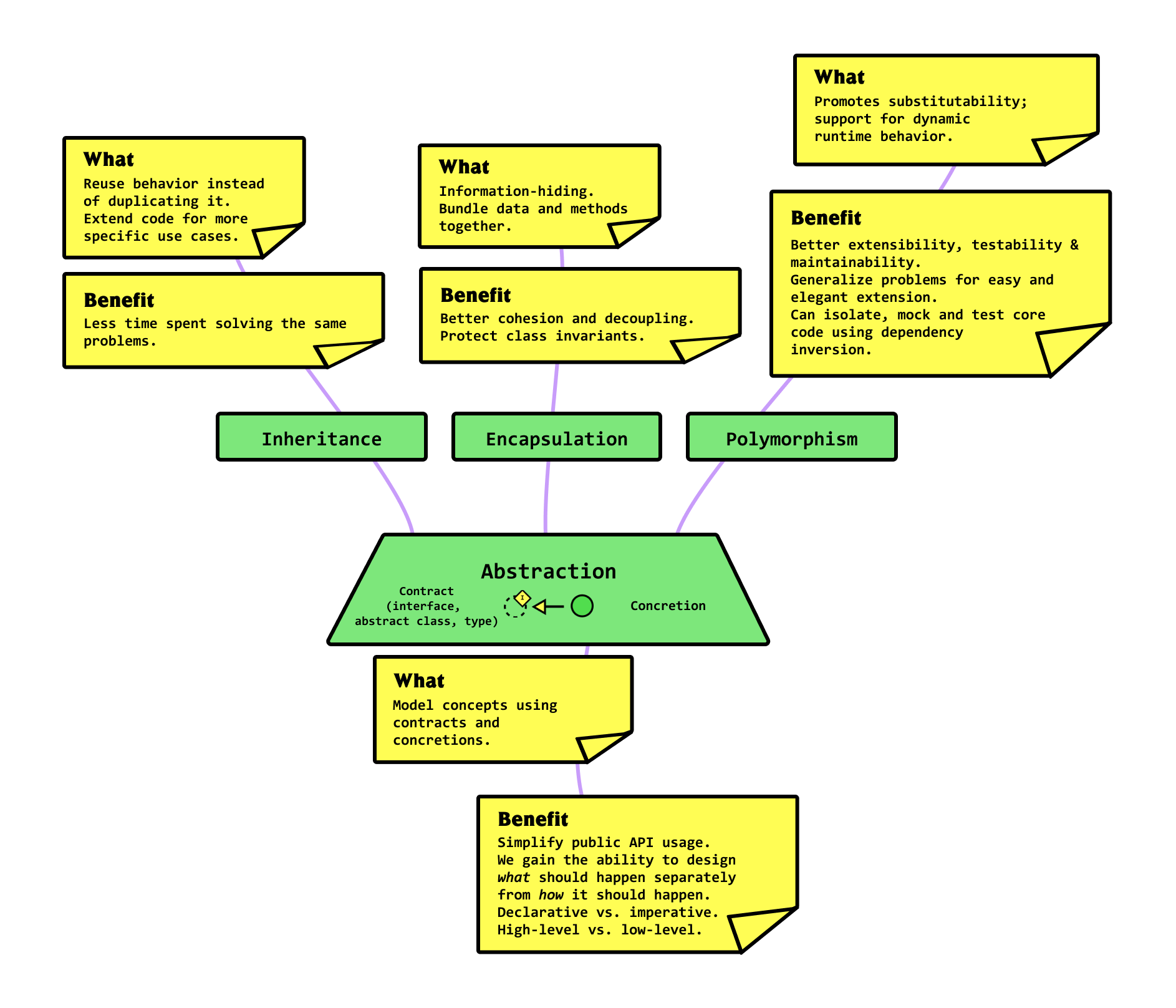
During the execution of the task, you should follow the rules:

* 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.

# Theory

Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic. An object can be defined as a data field that has unique attributes and behavior.

The four principles of object-oriented programming (abstraction, inheritance, encapsulation, and polymorphism) are features that - if used properly - can help us write more testable, flexible, and maintainable code.



Picture 1 – Main OOP principles.

Since the class characterizes individual new entities, it is necessary to interact with them somehow. This is done by creating methods and overloading existing operators. Overloading is necessary to override the behavior of some operators for correct work with the object.

In C++, a class consists of the following main items:

* **Class Properties.** These are essentially variables that store certain values.
* **Class Methods.** Functions that allow you to interact with the class properties. These include getters, setters, and overloaded operators.
* **Required methods:** constructor(s) and destructor. Constructor can be overloaded just like other methods, depending on the data type used in the parameters.

Also, there are some specifiers in the class responsible for differentiating access to methods or properties:

* **public**. All the class members declared under the public specifier will be available to everyone. The data members and member functions declared as public can be accessed by other classes and functions too. The public members of a class can be accessed from anywhere in the program using the direct member access operator (.) with the object of that class.
* **protected.** The protected access modifier is similar to the private access modifier in the sense that it can’t be accessed outside of its class unless with the help of a friend class. The difference is that the class members declared as Protected can be accessed by any subclass (derived class) of that class as well.
* **private**. The class members declared as private can be accessed only by the member functions inside the class. They are not allowed to be accessed directly by any object or function outside the class. Only the member functions or the friend functions are allowed to access the private data members of the class.

**Char arrays in C are a fundamental part of the language** and are used extensively for storing and manipulating strings. However, they can be challenging to work with and prone to errors if not used correctly. One common issue is buffer overflow, where more data is written to the array than it can hold, leading to memory corruption and program crashes. To avoid this, developers must ensure that the array size is sufficient for the data being stored, and they must also use functions like strncpy and strncat instead of strcpy and strcat.

**Python, on the other hand, is an interpreted language** with dynamic typing, which means that variables do not have to be declared with a specific data type before they are used. This makes it easier to work with complex data structures like lists and dictionaries, as the language automatically handles memory allocation and type conversion. Python's object-oriented nature also makes it easy to create and manipulate objects, which can be useful for building large-scale applications.

**The STL in C++ is a powerful library that provides** a wide range of generic algorithms, containers, and iterators that can be used with any C++ program. It includes commonly used data structures like vectors, sets, and maps, as well as algorithms for sorting, searching, and manipulating data. The STL is an essential tool for C++ developers and can significantly reduce development time by providing pre-built solutions for many common programming tasks.

**In C++, "const" is used to declare** a variable as read-only, preventing it from being modified after it has been initialized. "Friend" is used to grant access to private members of a class, allowing external functions or classes to access data that would otherwise be inaccessible. "&" is used to pass variables by reference instead of by value, which can improve performance by avoiding unnecessary copying of data.

Inheritance is a key concept in object-oriented programming that allows classes to inherit properties and methods from other classes. This can simplify code by reducing duplication and improving modularity, as well as allowing developers to create complex class hierarchies.

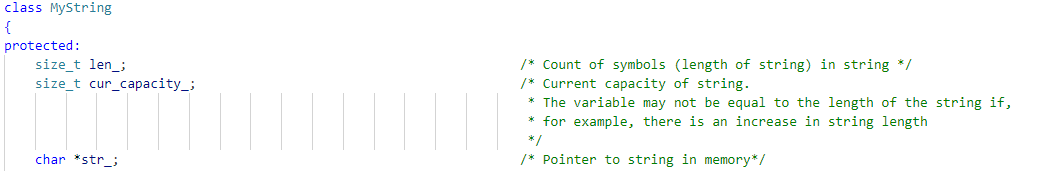
The "const" specifier is also used in class methods to indicate that the current method will not change the fields in the object.

**Wrapping** is a technique used to integrate code written in one language with code written in another language. This can be useful for combining the strengths of different languages or for using existing libraries written in a different language. There are several methods for wrapping code, including *SWIG, Boost.Python, and ctypes*, each with its own strengths and weaknesses. The wrapping process involves generating wrapper code that allows the two languages to communicate with each other, which can be challenging for complex codebases.

# Results

During the laboratory work it was necessary to implement some of the functionality from the *std::string* class without using it:

* Basic constructors
* Operator overloading for different operations
* Basic functions of interaction with string
* Methods that output "characteristics" of the string

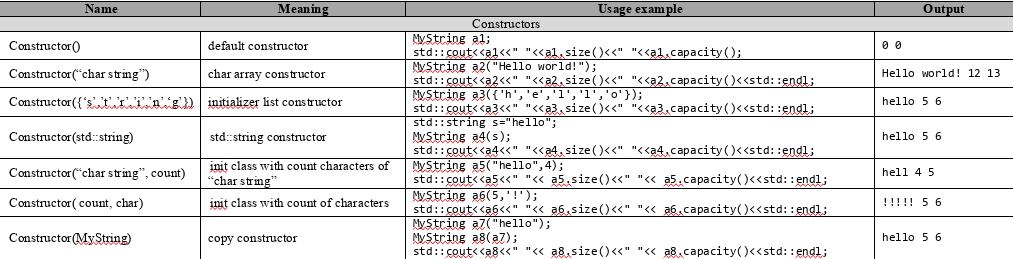


Picture 2 – Properties of class.

The point is that each method accepting parameters must work with different data types. Using template functions, unfortunately, cannot be a solution, because it is very difficult to adapt a template to work with data types and objects at the same time. Therefore, each function is overloaded for a limited set of data types.

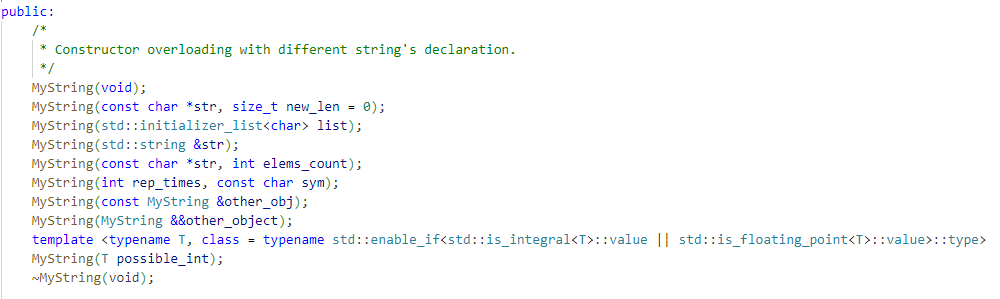
Taking into account the identical functionality of the overloaded functions, it is possible to fully describe only one function, and in the others to do only a small part of the manipulation to translate arguments into the required data type, and then call the main function.

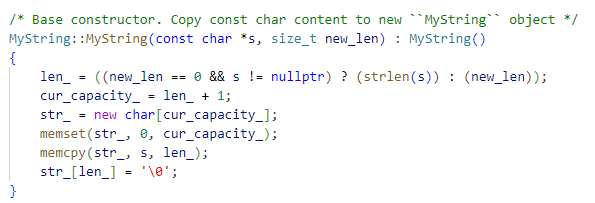
## Creating constructors



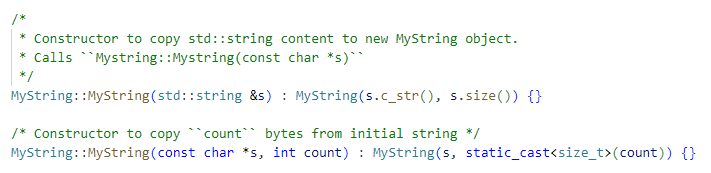
Picture 3 – All possible constructors

The basic idea: describe a constructor with const char \* and count parameters, and then use such a constructor in others.



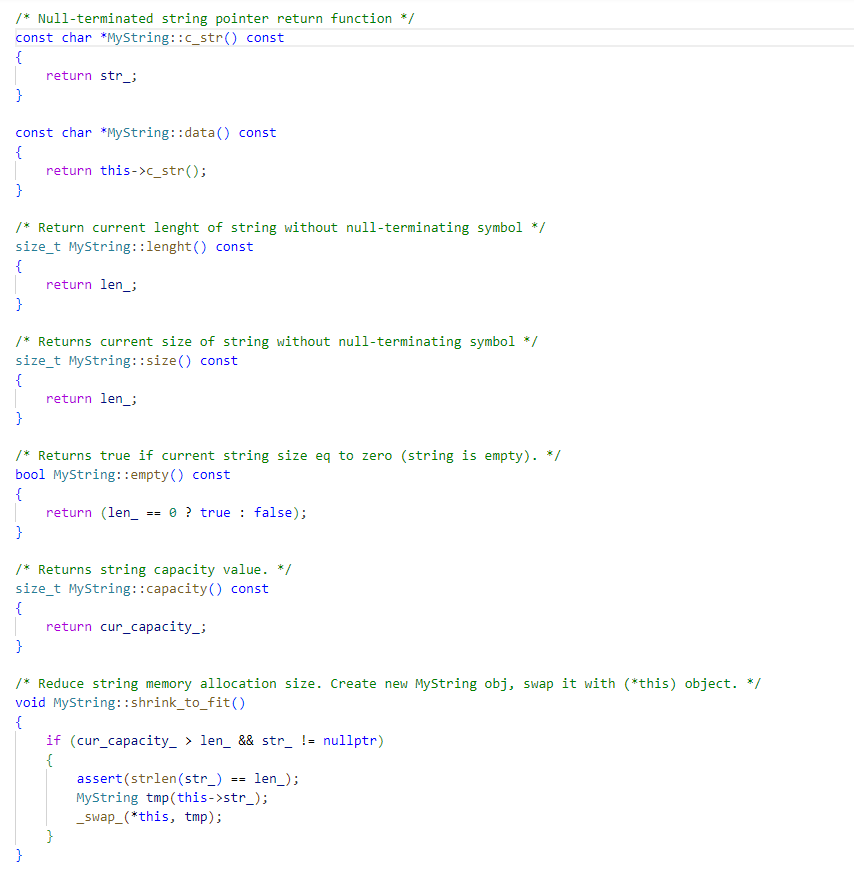
Picture 4 – The definition of all constructors.

Picture 5 – "Main" constructor.



Picture 6 – Example of using the implemented constructor.

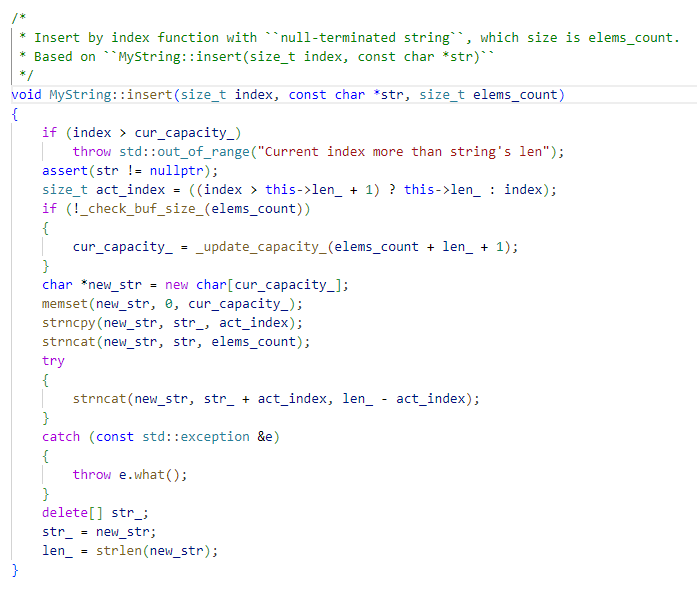
## Methods



Picture 7 – Class getters.

Getters are aimed at obtaining the values of class properties. It is important to note that some getters perform the same functionality. This is because in some C++ standards these methods have become the same (like data and c\_str).

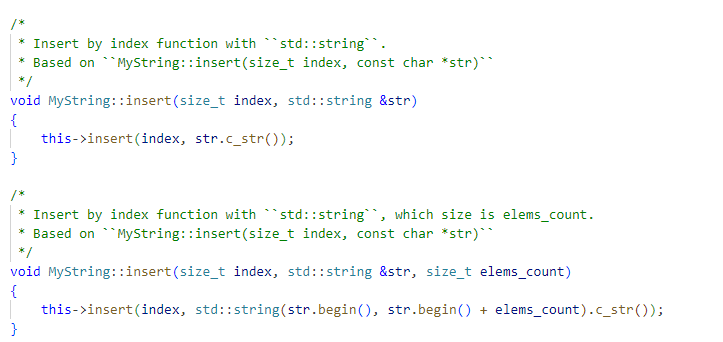
## MyString::insert



Picture 8 – insert method.

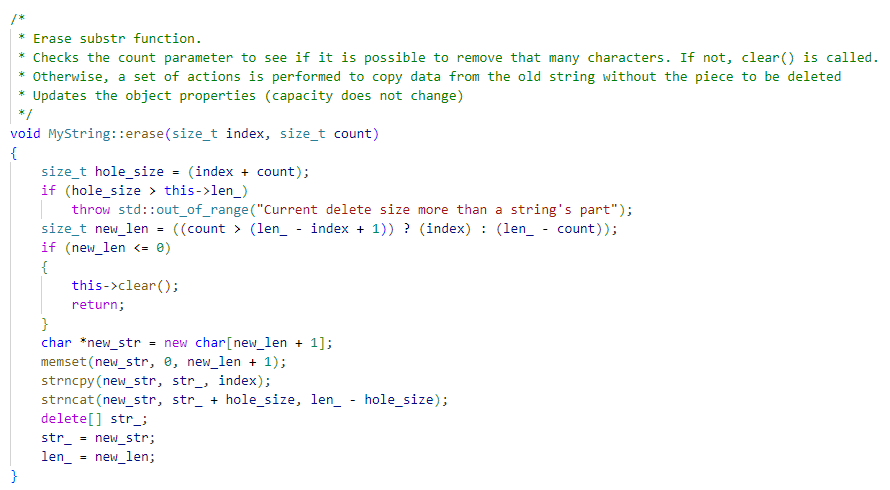
The basic idea of insert: to compute the actual index where the new substring will be added. For example, if there are 5 elements in the string, and the user passed the index 1000 as an argument, the substring will be placed at the end of the string.

Next, the size of the current string storage is checked: if the size allows adding a new string, the strncat function is called, if not - a new string is created, which is assembled piece by piece.



Picture 9 - Reusing the insert method.

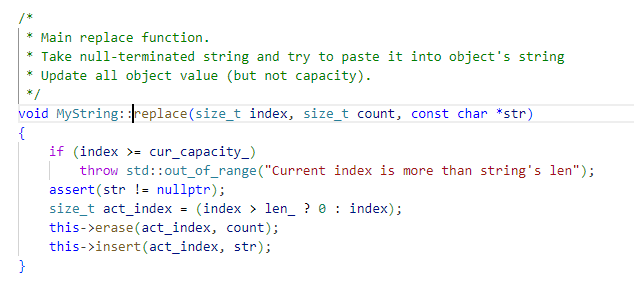
## MyString::erase



Picture 10 – erase method implementation.

Erase substr function. Checks the count parameter to see if this number of characters can be removed. If not, the clear() function is called. Otherwise, a set of actions is performed to copy the data from the old string without the fragment to be deleted. Updating object properties (capacity is not changed).

## MyString::replace



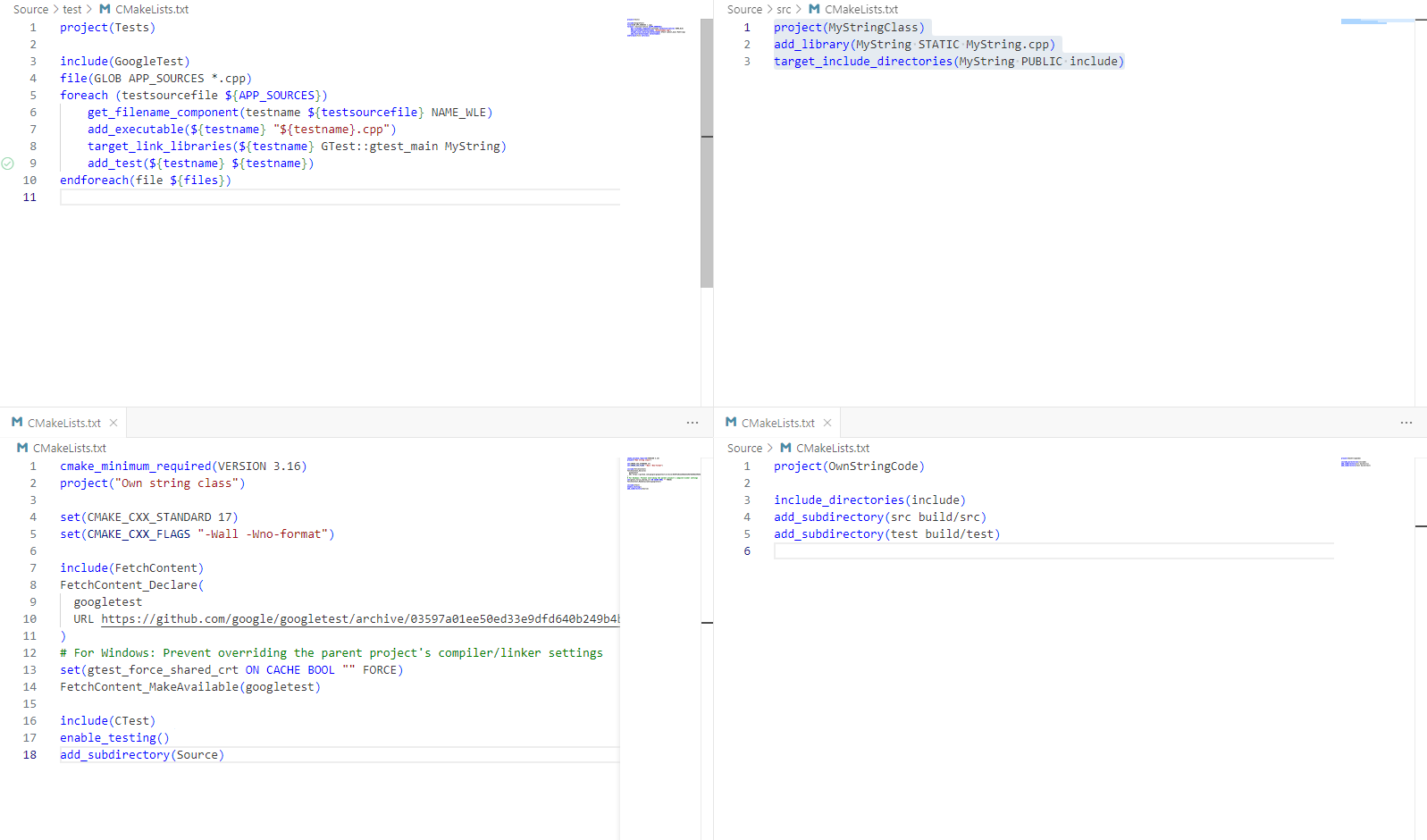
Picture 11 – replace function

In properly written code, the replace method turns out to be quite simple: it reuses the already defined erase and insert methods.

# Additional tasks

## Unit-tests

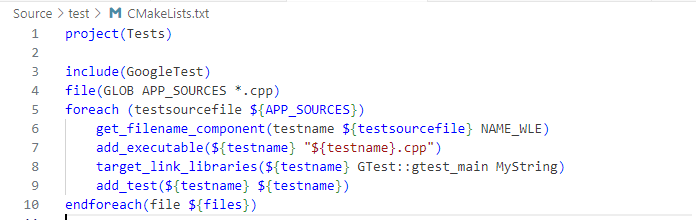
### Cmake build system



Picture 12 – Cmake structure

The build framework uses cmake to build the project and validate the written tests. In order to run tests, a separate library from Google is added to the build process to write unit tests. Since the project is small and there are few additional libraries, GoogleTest is added via Fetch.

Further, since the project supports some kind of file structure, you have to explicitly specify where to collect the source files from. A separate build file for unit tests:

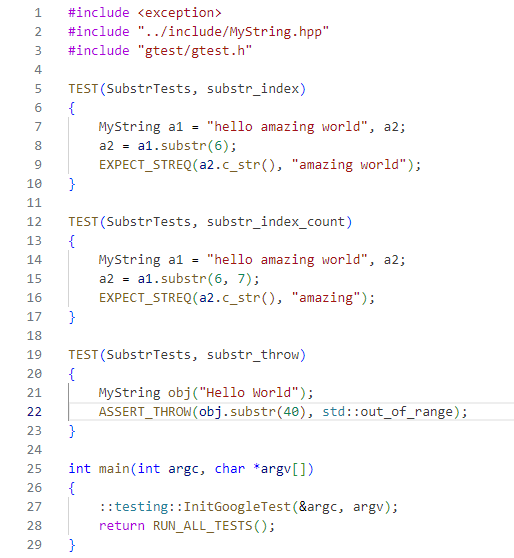


Picture 13 – CMakeLists.txt for tests.

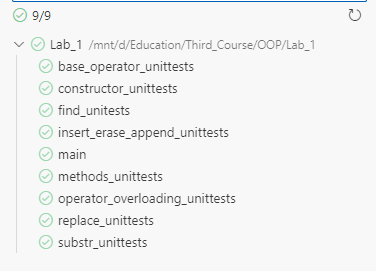
1. Include GoogleTest unit-test framework.
2. List all files in current directory.
3. A separate binary is built for each file, to which the libraries with MyString and the framework for testing are attached.
4. Each file is added to the project as a test to then use the ctest command for general test loading.

### Unit-test code syntax

Each of the .cpp files is a set of tests that check the created methods for correctness by comparing them with reference data. If one of the tests fails, the whole test block is considered a failure, which will be reflected in the terminal or in the development tool. If all tests are successful, the test block is considered to be completed.



Picture 14 – substr method unit-tests.

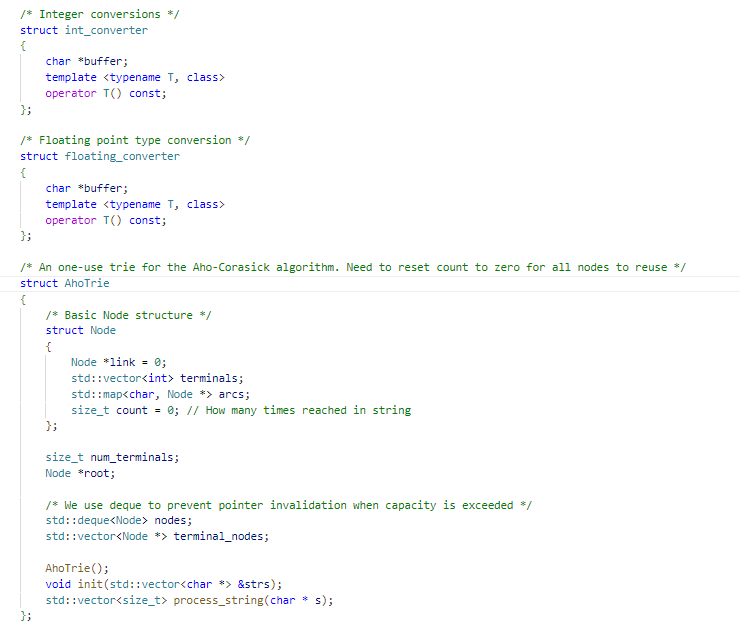


Picture 15 – Result of unit-testing.

## Additional methods

### New structures

This assignment required the addition of several structures:



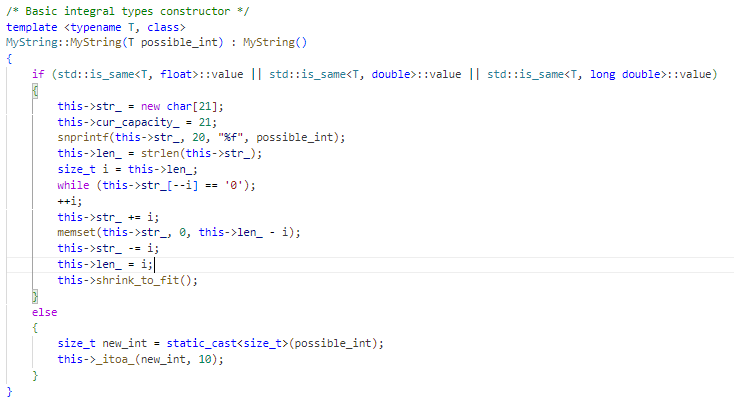
Picture 16 – New structures

* int\_converter / float\_converter - Structures for converting a string to a number depending on the data type (integral or floating point) in template functions by operator overloading.
* AhoeTrie - Tree structure for algorithmic finding. Need to fix

### Conversion constructor

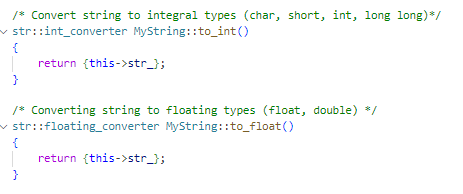


Picture 17 – new MyString constructor based on template.

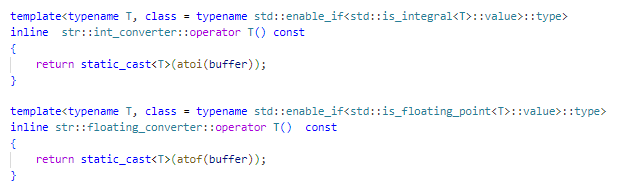


Picture 18 – New constructor implementation.

### to\_int and to\_float methods



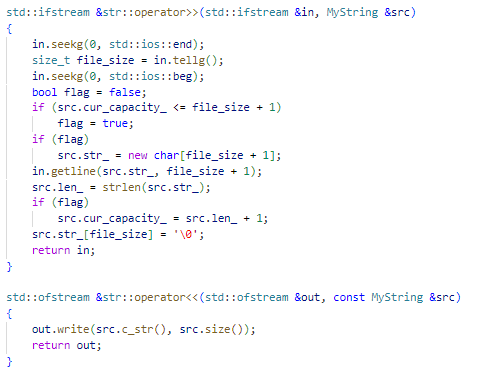
Picture 19 – method's implementations



Picture 20 – Template operator overloading to conversion methods

The basic idea behind the implementation: create new structures that translate the buffer field into a number based on template functions. Using a structure as a return value allows you to define the returned data type for a variable without using unnecessary function parameters. There is one disadvantage: this implementation is considered correct on older versions of compilers. GCC 13 version and Clang 10, 17 versions generate an error on such a construction.

### Filestream operator overloading



Picture 20 – Filestream operator overloading implementation.

When implementing the output operator to file, I thought to use the previously overloaded operator output to stdout, but a possible recursion error occurred during compilation. Therefore, it was decided to use the write function.

From the point of view of input from a file, the correct remark is that there is no need to check cur\_capacity\_, because in fact there is a primary input or a complete overwrite, and therefore cur\_capacity\_ will always depend only on the new data.

# Conclusion

Based on the results of the lab work, we created our own MyString class without using STL. The created solution was tested using GoogleTest framework. Considering the constant changes in the project, we can say that using Git or any other version control system turns out to be convenient for fixing bugs or adding new features.

Also, some additional functionality was added to the class, such as number-based constructor, at() method, reading from file and others.