**САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО**

**ИНСТИТУТ КОМПЬЮТЕРНЫХ НАУК И КИБЕРБЕЗОПАСНОСТИ**

**ВЫСШАЯ ШКОЛА ПРОГРАММНОЙ ИНЖЕНЕРИИ**

**Лабораторная работа №10**

**«ООП. Класс для работы со строками»**

**по предмету «Алгоритмизация и программирование»**

**Выполнил: студент гр. 5130904/30002 Севостьянова А.В.**

**Руководитель: Череповский Д.К.**

**Санкт-Петербург**

**2023 г.**

Оглавление

[1. Общая постановка задачи 3](#_Toc153537275)

[2. Детальные требования и тест план 4](#_Toc153537276)

[Таблица с детальными требованиями и тест планом 4](#_Toc153537277)

[3. Программа 4](#_Toc153537278)

[main.cpp 4](#_Toc153537279)

[String.h 4](#_Toc153537280)

[String.cpp 5](#_Toc153537281)

[Приложение 8](#_Toc153537282)

[Вывод 9](#_Toc153537283)

1. **Общая постановка задачи**

Для работы со строками создать свой класс String.

*Класс String должен содержать:*

***private:***

* size – длина строки, тип std::size\_t (#include <cstddef>)
* capacity – размер памяти, выделенный под строку, тип std::size\_t
* pointer – адрес начала строки, тип char\*

***public:***

***1) String();*** - Конструктор, создающий “пустую” строку.

***2) String(const char\* str)*** - Конструктор, создающий строку на основе строки в стиле С (с завершающим нулем).

***3) String(const String& str);*** - Конструктор копирования.

***4) String(String&& str) noexcept;*** - Конструктор перемещения.

***5) ~String ();*** - Деструктор.

***6) String& operator= (const char\* str);*** - Оператор присваивания строки в стиле С (с завершающим нулем).

***7) String& operator= (String&& str) noexcept;*** - Оператор перемещения.

***8) char& operator[] (std::size\_t pos)*** - Возвращает символ находящийся по индексу в строке начиная с 0 до size() – 1.

***9) String& append (const String& str);*** - Конкатенация двух строк

***10) String& operator+ (const String& str);*** - Конкатенация двух строк

***11) String& insert (std::size\_t pos, const char\* str);*** - Вставка строки в стиле С в строку типа String в заданную позицию pos.

***12) int compare (const String& str);*** - Cравнение двух строк типа String. Возвращает число <0, если вызывающая строка лексикографически меньше str, число > 0, если больше, число = 0, если строки равны.

Написать функцию для тестирования класса String.

1. **Детальные требования и тест план**

В данной работе не производится ввод данных пользователем, так что будем считать, что все переменные заданы корректно.

### *Таблица с детальными требованиями и тест планом*

|  |  |  |  |
| --- | --- | --- | --- |
| Требования | Детальные требования | Данные | Ожидаемый результат |
|  | Переменные корректны | – | Программа работает корректно |

1. **Программа**

## *main.cpp*

#include <iostream>

#include "String.h"

int main()

{

try {

setlocale(LC\_ALL, "ru");

testingString();

}

catch (std::runtime\_error& ex)

{

std::cerr << ex.what();

return EXIT\_FAILURE;

}

catch (...)

{

std::cout << "Что-то пошло не так :(\n";

return EXIT\_FAILURE;

}

return 0;

}

## *String.h*

#pragma once

#include<string>

#include<iostream>

#include <cstring>

#include <vector>

const std::string ERROR\_OF\_INDEX = "Индекс вне границ массива";

class String

{

friend std::ostream& operator << (std::ostream& stream, String& str);

public:

String();

String(const char\* str);

String(const String& str) ;

String(String&& str)noexcept;

~String();

String& operator = (const char\* str);

String& operator = (String && str) noexcept;

char& operator[] (size\_t pos);

String& append(const String& str);

String& operator+ (const String& str);

String& insret(size\_t pos, const char\* str);

int compare(const String& str);

private:

size\_t size, capacity;

std::vector <char>\* pointer;

std::vector <char> c\_str;

};

void testingString();

## *String.cpp*

#include "String.h"

String::String()

{

size = 0;

capacity = 0;

pointer = nullptr;

};

String::String(const char\* str)

{

size = strlen(str);

capacity = size + 1;

for (int i = 0; i < size; i++)

c\_str.push\_back(str[i]);

c\_str.push\_back('\0');

pointer = &c\_str;

}

String::String(const String& str)

{

size = str.size;

capacity = str.capacity;

pointer = str.pointer;

}

String::String(String&& str)noexcept

{

size = str.size;

capacity = str.capacity;

pointer = str.pointer;

str.size = 0;

str.capacity = 0;

str.pointer = nullptr;

};

String::~String() {}

String& String::operator=(const char\* str)

{

size = strlen(str);

capacity = size + 1;

for (int i = 0; i < size; i++)

c\_str.push\_back(str[i]);

c\_str.push\_back('\0');

pointer = &c\_str;

return \*this;

}

String& String::operator=(String&& str) noexcept

{

size = str.size;

capacity = str.capacity;

pointer = str.pointer;

str.size = 0;

str.capacity = 0;

str.pointer = nullptr;

return \*this;

}

char& String::operator[](size\_t pos)

{

if (pos < 0 || pos >= size) throw std::runtime\_error(ERROR\_OF\_INDEX);

else return (\*pointer).at(pos);

}

String& String::append(const String& str)

{

size += str.size;

capacity = size +1;

(\*pointer).pop\_back();

for (char element : \*str.pointer)

(\*pointer).push\_back(element);

return \*this;

}

String& String::operator+(const String& str)

{

size += str.size;

capacity = size + 1;

(\*pointer).pop\_back();

for (char element : \*str.pointer)

(\*pointer).push\_back(element);

return \*this;

}

String& String::insret(size\_t pos,const char\* str)

{

if (0 > pos || pos >= size) throw std::runtime\_error(ERROR\_OF\_INDEX);

else

{

size += strlen(str);

capacity = size + 1;

auto it = pointer->begin() + pos;

for (int i = 0; i < strlen(str); i++)

{

pointer->insert(it, str[i]);

it = pointer->begin()+pos+1;

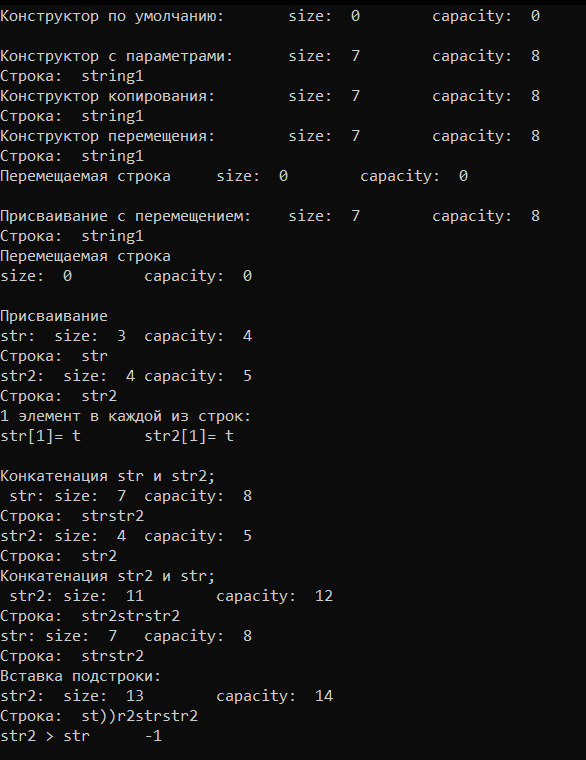
}

return \*this;

}

}

# Приложение



*Рис. 1 Корректная работа программы*

# Вывод

В ходе работы:

1. Были изучены и реализованы конструктор перемещения и операция присваивания с перемещением.
2. Были перегружены операторы +, <<, [], =
3. Были реализованы дружественные функции.