ФЕДЕРАЛЬНОЕ АГЕНСТВО ЖЕЛЕЗНОДОРОЖНОГО ТРАНСПОРТА

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ПУТЕЙ СООБЩЕНИЯ Императора Александра I»

Кафедра «Информационные и вычислительные системы»

Дисциплина «Программирование С++»

**ОТЧЁТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2**

ВАРИАНТ 19

|  |  |
| --- | --- |
| Выполнил студент  Факультет: АИТ  Группа: ИВБ-211 | Шефнер А. |
| Проверил: | Проузин О.В. |

**Санкт-Петербург**

**2023**

Оценочный лист результатов ЛР № 2

Ф.И.О. студента \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Шефнер Альберт\_\_\_\_\_

Группа \_\_\_\_\_\_\_\_\_\_\_\_\_ИВБ-211\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№**  **п/п** | **Материалы необходимые для оценки знаний, умений**  **и навыков** | **Показатель**  **оценивания** | **Критерии**  **Оценивания** | **Шкала оценивания** | **Оценка** |
| 1 | Лабораторная работа№ | Соответствие методике выполнения | Соответствует | 7 |  |
| Не соответствует | 0 |
| Срок выполнения | Выполнена в срок | 2 |  |
| Выполнена с опозданием на 2 недели | 0 |
| оформление | Соответствует требованиям | 1  0 |  |
| Не соответствует |  |
|  | **ИТОГО количество баллов** |  |  | 10 |  |

Доцент кафедры

«Информационные и вычислительные

системы» Проурзин О.В. «\_\_» \_\_\_\_\_\_\_\_\_\_2023 г.

**Цели работы:**

* освоить работу с указателями.
* Узнать основные методы обработки строк

**Задание**

Создать тестовую функцию **main ( )**, которая реализует алгоритм, показанный на рисунке:

**Условие ввода строк:**

Пока последние 2 символа не являются цифрами.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Условие обработки строки**:

Всякие разные.

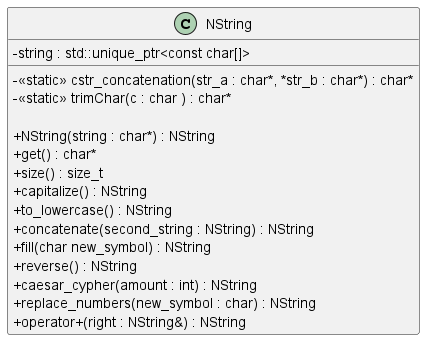
**Используемые средства**

В качестве интегрированной среды разработки использовалась JetBrains CLion.

Для работы в консоли с потоками ввода-вывода использовалась стандартная библиотека <iostream>.

Для работы со строками использовалась библиотека <cstring>.

**UML-диаграмма программы:**

****

**Код программы с комментариями**

**main.cpp:**

#include <iostream>  
#include "NString.h"  
  
bool check\_str(NString \*str);  
void process\_str(NString \*str);  
NString get\_string();  
void clear();  
void pause();  
  
char tmp[80];  
  
int main() {  
 while (true) {  
 clear();  
 // мне не хочется придумывать пустой конструктор  
 // имея умный указатель, как я понял это будет  
 // боль на 100+ строчек, поэтому сделать  
 // глобальную переменную str не выйдет.  
 NString str = get\_string();  
 if (!check\_str(&str)) break;  
 process\_str(&str);  
 pause();  
 }  
 return 0;  
}  
  
// лучше дать имя в коде, чем писать комментарий что делает эта строчка  
void clear() {  
 system("cls");  
}  
  
NString get\_string() {  
 std::cout << "Enter a string (80 char max):\n";  
 std::cin.getline(tmp, 80);  
 return NString(tmp);  
}  
  
// проверка на то, являются ли 2 последних символа числами  
bool check\_str(NString \*str) {  
 size\_t len = str->size();  
 if (len <= 1) {  
 return true;  
 }  
 return !(str->get()[len - 1] >= '0' && str->get()[len - 1] <= '9' && str->get()[len - 2] >= '0' &&  
 str->get()[len - 2] <= '9');  
}  
  
// вывести все методы, которые я смог придумать  
void process\_str(NString \*str) {  
 std::cout << str->get() << std::endl;  
 std::cout << str->to\_lowercase().get() << std::endl;  
 std::cout << str->capitalize().get() << std::endl;  
 std::cout << str->fill('\_').get() << std::endl;  
 std::cout << str->reverse().get() << std::endl;  
 std::cout << str->caesar\_cypher(5).get() << std::endl;  
 std::cout << str->caesar\_cypher(5).caesar\_cypher(-5).get() << std::endl;  
 std::cout << str->replace\_numbers('#').get() << std::endl;  
}  
  
void pause() {  
 system("pause");  
}  
  
void free() {  
  
}

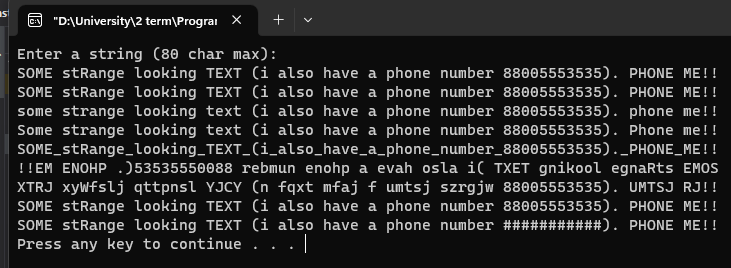
**NString.h:**

#ifndef NSTRING\_NSTRING\_H  
#define NSTRING\_NSTRING\_H  
  
#include <memory>  
  
  
class NString {  
public:  
 explicit NString(const char \*string);  
  
 //~NString() = default;  
  
 const char \*get() const {  
 return string.get();  
 }  
  
 size\_t size() const;  
  
 NString capitalize() const;  
  
 NString to\_lowercase() const;  
  
 NString concatenate(NString second\_string) const;  
  
 NString fill(char new\_symbol) const;  
  
 NString reverse() const;  
  
 NString caesar\_cypher(int amount) const;  
  
 NString replace\_numbers(char new\_symbol) const;  
  
 NString operator+(const NString &right) const;  
  
private:  
 std::unique\_ptr<const char[]> string;  
  
 static char \*cstr\_concatenation(const char \*str\_a, const char \*str\_b);  
  
 static char trimChar(char c);  
};  
  
  
#endif //NSTRING\_NSTRING\_H

**NString.cpp:**

#include "NString.h"  
#include <cstring>  
  
NString::NString(const char \*string) {  
 this->string = std::unique\_ptr<const char[]>(string);  
}  
  
size\_t NString::size() const {  
 return strlen(this->get());  
}  
  
NString NString::to\_lowercase() const {  
 size\_t size = this->size() + 1;  
 char \*new\_string = new char[size];  
 for (int i = 0; i < size - 1; i++) {  
 char tmp = this->string[i];  
 if (tmp >= 'A' && tmp <= 'Z') {  
 tmp += 32;  
 }  
 new\_string[i] = tmp;  
 }  
 new\_string[size - 1] = 0;  
 return NString(new\_string);  
}  
  
NString NString::concatenate(NString second\_string) const {  
 const char \*result\_str = cstr\_concatenation(this->get(), second\_string.get());  
 return NString(result\_str);  
}  
  
NString NString::capitalize() const {  
 size\_t size = this->size() + 1;  
 char \*new\_string = new char[size];  
 bool need\_capitalize = true;  
  
 for (int i = 0; i < size - 1; i++) {  
 char tmp = this->string[i];  
 if (this->string[i] == '.') {  
 need\_capitalize = true;  
 } else if (need\_capitalize) {  
 if (tmp >= 'a' && tmp <= 'z') {  
 tmp -= 32;  
 } else if (tmp != ' ') {  
 need\_capitalize = false;  
 }  
 } else {  
 if (tmp >= 'A' && tmp <= 'Z') {  
 tmp += 32;  
 }  
 }  
 new\_string[i] = tmp;  
 }  
  
 new\_string[size - 1] = 0;  
 return NString(new\_string);  
}  
  
char \*NString::cstr\_concatenation(const char \*str\_a, const char \*str\_b) {  
 size\_t size\_a = strlen(str\_a);  
 size\_t size\_b = strlen(str\_b);  
 size\_t size = size\_a + size\_b + 1;  
 char \*new\_string = new char[size];  
  
 int index = 0;  
 for (int i = 0; i < size\_a; i++, index++) {  
 new\_string[index] = str\_a[i];  
 }  
 for (int i = 0; i < size\_b; i++, index++) {  
 new\_string[index] = str\_b[i];  
 }  
  
 new\_string[size - 1] = 0;  
 return new\_string;  
}  
  
NString NString::fill(const char new\_symbol) const {  
 size\_t size = this->size() + 1;  
 char \*new\_string = new char[size];  
 for (int i = 0; i < size; i++) {  
 if (this->get()[i] == ' ') {  
 new\_string[i] = new\_symbol;  
 } else {  
 new\_string[i] = this->get()[i];  
 }  
 }  
 return NString(new\_string);  
}  
  
NString NString::reverse() const {  
 size\_t count = this->size();  
 char \*new\_string = new char[count + 1];  
 for (int i = 0; i < count; i++) {  
 new\_string[i] = this->get()[count - i - 1];  
 }  
 new\_string[count] = '\0';  
 return NString(new\_string);  
}  
  
NString NString::caesar\_cypher(const int amount) const {  
 size\_t size = this->size() + 1;  
 char \*new\_string = new char[size];  
 for (int i = 0; i < size; i++) {  
 char tmp = this->get()[i];  
 if (tmp >= 'a' && tmp <= 'z') {  
 tmp -= 'a';  
 tmp += amount;  
 tmp = trimChar(tmp);  
 tmp += 'a';  
 }  
 if (tmp >= 'A' && tmp <= 'Z') {  
 tmp -= 'A';  
 tmp += amount;  
 tmp = trimChar(tmp);  
 tmp += 'A';  
 }  
 new\_string[i] = tmp;  
 }  
 return NString(new\_string);  
}  
  
NString NString::replace\_numbers(const char new\_symbol) const {  
 size\_t size = this->size() + 1;  
 char \*new\_string = new char[size];  
 for (int i = 0; i < size; i++) {  
 char tmp = this->get()[i];  
 if (tmp >= '0' && tmp <= '9') {  
 tmp = new\_symbol;  
 }  
 new\_string[i] = tmp;  
 }  
 return NString(new\_string);  
}  
  
NString NString::operator+(const NString &right) const {  
 const char \*result\_str = cstr\_concatenation(this->get(), right.get());  
 return NString(result\_str);  
}  
  
char NString::trimChar(char c) {  
 if (c >= 26) c -= 26;  
 else if (c < 0) c += 26;  
 return c;  
}

**Тестовые примеры**

****