Министерство образования Республики Беларусь

Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра электронных вычислительных машин

ОТЧЕТ

О ЛАБОРАТОРНОЙ РАБОТЕ № 2

Дружественные функции. Перегрузка операций

по дисциплине «Программирование на языках высокого уровня»

Выполнил ст. гр. 450503 А.П. Красько

Проверил асс. каф. ЭВМ И.Г. Скиба

Минск 2025

# 1 ПОСТАНОВКА ЗАДАЧИ

Реализовать класс String для работы со строками символов. Перегрузить операторы <, >, !=, <=, >=. Предоставить конструктор копирования. Определить friend функции для операций ввода-вывода в поток.

# 2 ЛИСТИНГ КОДА

Файл main.cс

#include <functional>

#include "../../lib/consoleUtils.hh"

#include "screens.hh"

#include "string.hh"

using namespace std;

using namespace console\_utils;

using namespace screen\_handlers;

int main(void) {

str::String str1;

str::String str2;

array<function<bool(str::String &, str::String &)>, 4> actions = {

inputStrings, printStrings, checkOperators, [](const str::String &, const str::String &) { return 0; }};

unsigned int response;

do {

printMainScreen();

readT(response, ">", [](unsigned int num) { return num > 0 && num <= 4; });

cout << "\x{1B}[2J\x{1B}[H\n";

} while (actions[response - 1](str1, str2));

return 0;

}

Файл screens.cc

##include <iostream>

#include <print>

#include "string.hh"

#include "../../lib/consoleUtils.hh"

using namespace std;

namespace screen\_handlers {

bool printMainScreen() {

auto [cols, rows] = console\_utils::getConsoleDimensions();

println("{:^{}}", "\x{1B}[48;5;35mLab 2\x{1B}[0m", cols);

println("Please select action:\n");

println(" 1.Input 2 strings");

println(" 2.Print strings");

println(" 3.Check operators");

println(" 4.Exit");

return true;

}

bool inputStrings(str::String &str1, str::String &str2) {

cout << "Please enter first string:";

cin >> str1;

cout << "Please enter second string:";

cin >> str2;

return true;

}

bool printStrings(const str::String &str1, const str::String &str2) {

cout << "First string: " << str1 << std::endl;

cout << "Second string: " << str2 << std::endl;

return true;

}

bool checkOperators(const str::String &str1, const str::String &str2) {

println("str1 < str2: {}", str1 < str2);

println("str1 > str2: {}", str1 > str2);

println("str1 <= str2: {}", str1 <= str2);

println("str1 >= str2: {}", str1 >= str2);

println("str1 != str2: {}", str1 != str2);

return true;

}

} // namespace screen\_handlers

Файл screens.hh

#pragma once

#include "string.hh"

namespace screen\_handlers {

void printMainScreen();

bool inputStrings(str::String &str1, str::String &str2);

bool printStrings(const str::String &str1, const str::String &str2);

bool checkOperators(const str::String &str1, const str::String &str2);

} // namespace screen\_handlers

Файл string.cc

#include "string.hh"

#include <iostream>

namespace str {

String::String(const char\* str) : length\_{0}, dataPtr\_{nullptr} {

for (; str[length\_]; length\_++);

++length\_;

dataPtr\_ = std::make\_unique\_for\_overwrite<char[]>(length\_);

std::ranges::copy(str, str + length\_, dataPtr\_.get());

}

String::String() : length\_{2}, dataPtr\_{std::make\_unique<char[]>(length\_)} {};

String::String(const String& other) : length\_{other.length\_}, dataPtr\_{std::make\_unique<char[]>(length\_)} {

std::ranges::copy(other.dataPtr\_.get(), other.dataPtr\_.get() + length\_, dataPtr\_.get());

};

String::String(String&& other) noexcept : length\_{other.length\_}, dataPtr\_{std::move(other.dataPtr\_)} {};

String& String::operator=(const String& other) {

length\_ = other.length\_;

resizeD\_(length\_);

std::ranges::copy(other.dataPtr\_.get(), other.dataPtr\_.get() + length\_, dataPtr\_.get());

return \*this;

};

size\_t String::getLen() const { return length\_; }

String& String::operator=(String&& other) noexcept {

length\_ = other.length\_;

dataPtr\_ = std::move(other.dataPtr\_);

return \*this;

};

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

if (index >= length\_) throw std::invalid\_argument("Index out of range");

return dataPtr\_[index];

}

const char& String::operator[](size\_t index) const {

if (index >= length\_) throw std::invalid\_argument("Index out of range");

return dataPtr\_[index];

}

void String::resizeD\_(size\_t newLen) {

length\_ = newLen;

auto tmp = std::make\_unique<char[]>(length\_);

dataPtr\_ = std::move(tmp);

}

void String::resize\_(size\_t newLen) {

length\_ = newLen;

auto tmp = std::make\_unique<char[]>(length\_);

std::ranges::copy(dataPtr\_.get(), dataPtr\_.get() + length\_, tmp.get());

dataPtr\_ = std::move(tmp);

}

void String::readFromStream\_(std::istream& is) {

char tmp;

size\_t counter = 0;

while (is.get(tmp) && tmp != '\n') {

if (counter >= length\_ - 1) {

resize\_(length\_ \* 2);

}

dataPtr\_[counter] = tmp;

counter++;

}

resize\_(counter + 1);

dataPtr\_[counter] = '\0';

is.clear();

}

bool operator!=(const String& str1, const String& str2) {

return str1.length != str2.length\_;

}

void printString(const String& str) { std::cout << str.dataPtr\_.get(); }

void readString(String& str) { str.readFromStream\_(std::cin); }

} // namespace str

Файл string.hh

#pragma once

#include <memory>

namespace str {

class String {

private:

size\_t length\_;

std::unique\_ptr<char[]> dataPtr\_;

void resizeD\_(size\_t newLen);

void resize\_(size\_t newLen);

void readFromStream\_(std::istream& is);

friend std::ostream& operator<<(std::ostream& os, const String& obj) {

os << obj.dataPtr\_.get();

return os;

};

friend std::istream& operator>>(std::istream& is, String& obj) {

obj.readFromStream\_(is);

return is;

}

public:

explicit String(const char\* str);

String();

String(const String& other);

String(String&& other) noexcept;

size\_t getLen() const;

String& operator=(const String& other);

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

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

const char& operator[](size\_t index) const;

auto operator<=>(const String& other) const { return length\_ <=> other.length\_; }

friend void printString(const String& str);

friend void readString(String& str);

friend bool operator!=(const String& str1, const String& str2);

};

void printString(const String& str);

void readString(String& str);

} // namespace str

Файл consoleUtils.cc

#include <iostream>

#ifdef \_\_linux\_\_

#include <sys/ioctl.h>

#include <unistd.h>

#endif

#ifdef \_WIN32

#include <windows.h>

#endif

namespace console\_utils {

std::pair<int, int> getConsoleDimensions() {

#ifdef \_WIN32

CONSOLE\_SCREEN\_BUFFER\_INFO csbi;

GetConsoleScreenBufferInfo(GetStdHandle(STD\_OUTPUT\_HANDLE), &csbi);

return std::make\_pair(csbi.srWindow.Right - csbi.srWindow.Left + 1, csbi.srWindow.Bottom - csbi.srWindow.Top + 1);

#endif

#ifdef \_\_linux\_\_

struct winsize w;

ioctl(STDOUT\_FILENO, TIOCGWINSZ, &w);

return std::make\_pair(w.ws\_col, w.ws\_row);

#endif

}

} // namespace console\_utils

Файл consoleUtils.hh

#pragma once

#include <iostream>

#include <limits>

namespace console\_utils {

std::pair<int, int> getConsoleDimensions();

template <typename T, typename CT>

void readT(T& data, const std::string& massage, CT bound) {

std::cout << massage;

while (((std::cin >> data).fail()) || !bound(data)) {

std::cout << "Invalid input. Reread input requierments\n";

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), ' ');

std::cout << massage;

}

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

}

template <typename T>

void readT(T& data, const std::string& massage) {

std::cout << massage;

while ((std::cin >> data).fail()) {

std::cout << "Invalid input. Reread input requierments\n";

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), ' ');

std::cout << massage;

}

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

}

template <typename T, typename CT>

void readT(T& data, const std::string& message, CT bound, const std::string& errmess) {

std::cout << message;

while (((std::cin >> data).fail()) || !bound(data)) {

std::cout << errmess;

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

std::cout << message;

}

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

}

} // namespace console\_utils

**3 РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**



Рисунок 4.1 – Главное меню

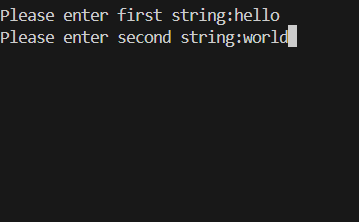


Рисунок 4.2 – Меню ввода

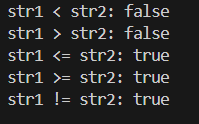


Рисунок 4.3 – Проверка работы операторов



Рисунок 4.4 – Вывод строк

**4 ЗАКЛЮЧЕНИЕ**

В ходе выполнения лабораторной работы был успешно реализован класс String для работы со строками, включая перегрузку операторов сравнения (<, >, <=, >=, !=), конструктор копирования, а также дружественные функции для ввода и вывода в поток. Программа предоставляет интерактивный интерфейс для тестирования функциональности класса, демонстрируя корректность работы всех реализованных операций.