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.cc

```
#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

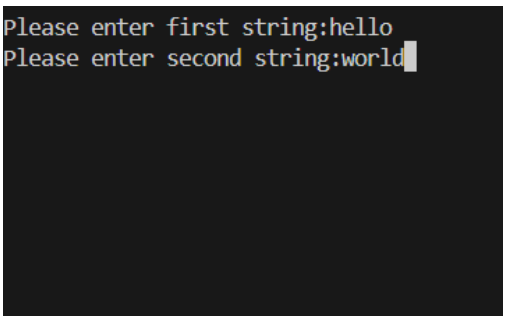


Рисунок 4.2 – Меню ввода

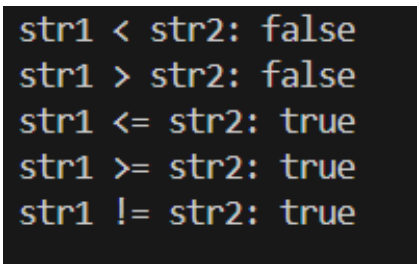


Рисунок 4.3 – Проверка работы операторов


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.