МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСТИЕТ

Кафедра вычислительной техники

Лабораторная работа №2

по дисциплине «Программирование»

на тему: «Переопределение операций»

Группа: АВТ-441

Студент: Тютрин Влад

Преподаватель: Орлов Илья Викторович

Новосибирск 2025

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc208001212)

[1 Описание разрабатываемого класса 4](#_Toc208001213)

[2 Программная реализация 5](#_Toc208001214)

[3 Тестирование 6](#_Toc208001215)

[ЗАКЛЮЧЕНИЕ 7](#_Toc208001216)

[ПРИЛОЖЕНИЕ 8](#_Toc208001217)

ВВЕДЕНИЕ

**Цель работы:**

Тема: Ознакомиться с особенностями использования дружественных классов и функций, а также возможностью получения законченного нового типа данных, определив для него допустимые операции с помощью перегрузки операторов.

Для класса pаботы со cтpоками перегрузить следующие операции:

сложение 2-х объектов класса;

сложение со строкой типа char\* как замена строки;

операции сравнения строк;

операцию присваивания.

**Вариант:** 10

1. Описание разрабатываемого класса

Разработанный класс `String` представляет собой расширенную реализацию для работы со строками символов, обеспечивающую полный набор операций через перегруженные операторы. Класс инкапсулирует механизмы динамического управления памятью и предоставляет интуитивно понятный интерфейс для манипуляций со строковыми данными. Основной особенностью реализации является интеграция объектно-ориентированного подхода с возможностями перегрузки операций, что позволяет использовать объекты класса в выражениях естественным образом, аналогично работе со встроенными типами данных.

Архитектура класса включает конструкторы для различных сценариев инициализации, деструктор для корректного освобождения ресурсов и набор методов для базовых операций со строками. Особое внимание уделено безопасности операций с памятью - все методы обеспечивают целостность данных и предотвращают утечки памяти. Статический член класса ведет подсчет активных объектов, предоставляя возможность мониторинга использования ресурсов в течение времени выполнения программы.

Перегруженные операторы реализованы как методами класса, так и дружественными функциями, что демонстрирует различные подходы к перегрузке. Операторы арифметического сложения поддерживают комбинации объектов класса со строками в стиле C, операторы сравнения обеспечивают лексикографическое сравнение строк, а оператор присваивания реализует безопасное копирование с учетом самоприсваивания. Дружественные функции для ввода и вывода интегрируют класс с стандартными потоками, обеспечивая удобный интерфейс для взаимодействия с пользователем.

Данная реализация образует законченный тип данных, который может быть использован в реальных проектах для работы со строками, предоставляя при этом все преимущества объектно-ориентированного подхода, включая инкапсуляцию, безопасность типов и удобство использования.

1. Программная реализация

Программная реализация проекта выполнена с соблюдением принципов объектно-ориентированного программирования и представляет собой законченное решение для работы со строковыми данными. Архитектура проекта организована в виде трех модулей, обеспечивающих четкое разделение интерфейса и реализации, что соответствует современным стандартам разработки программного обеспечения.

В основе системы лежит класс String, который инкапсулирует данные и поведение для работы со строками. Заголовочный файл содержит объявление класса с продуманной структурой, включающей приватные поля для хранения указателя на строку и ее длины, а также статическую переменную для подсчета объектов. Публичный интерфейс класса составляет комплекс конструкторов и деструктора, базовые методы для операций со строками и набор перегруженных операторов, реализованных различными способами.

Файл реализации содержит детальную проработку всех методов класса с акцентом на безопасность работы с динамической памятью. Конструкторы выполняют инициализацию объектов с контролем корректности входных данных и соответствующим изменением статического счетчика. Деструктор гарантирует освобождение выделенных ресурсов. Особое внимание уделено реализации оператора присваивания, который обрабатывает случай самоприсваивания и предотвращает утечки памяти.

Перегруженные операторы арифметического сложения поддерживают различные комбинации операндов и реализованы с созданием временных буферов для формирования результирующих строк. Операторы сравнения используют лексикографический порядок и корректно обрабатывают специальные случаи, включая пустые строки. Дружественные функции обеспечивают симметричность операций и интеграцию с потоками ввода-вывода, что значительно расширяет возможности использования класса.

Демонстрационная программа представляет собой комплексный тест, проверяющий все аспекты функционирования класса. Программа последовательно создает объекты различными способами, тестирует операции присваивания, арифметические операции сложения с разными типами операндов, полный набор операций сравнения и комбинированные выражения. Особенностью демонстрации является наглядное отображение работы статического счетчика объектов throughout времени выполнения, что позволяет отслеживать динамику создания и уничтожения объектов.

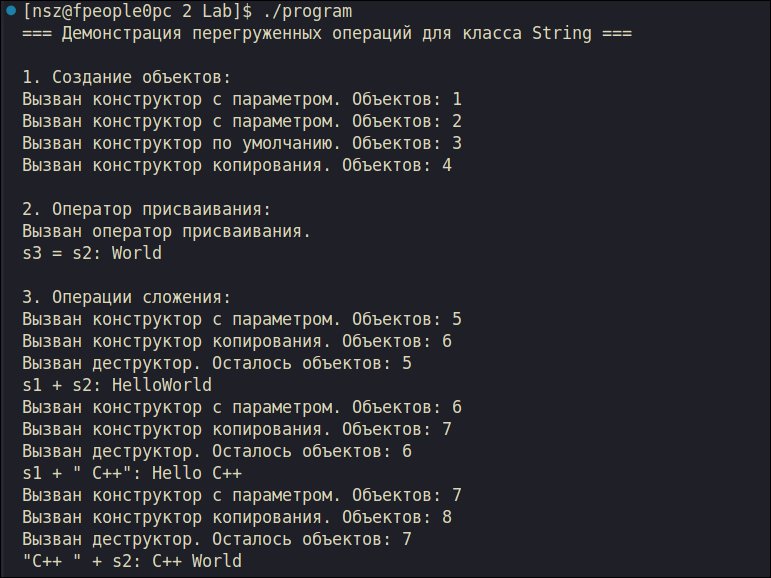
Проект успешно компилируется стандартными средствами и демонстрирует стабильную работу при различных сценариях использования. Тестирование включает проверку граничных условий, обработку пустых строк и работу с данными значительного объема. Реализация обеспечивает защиту от типичных ошибок работы с памятью и представляет собой надежное решение для обработки строковых данных в приложениях.

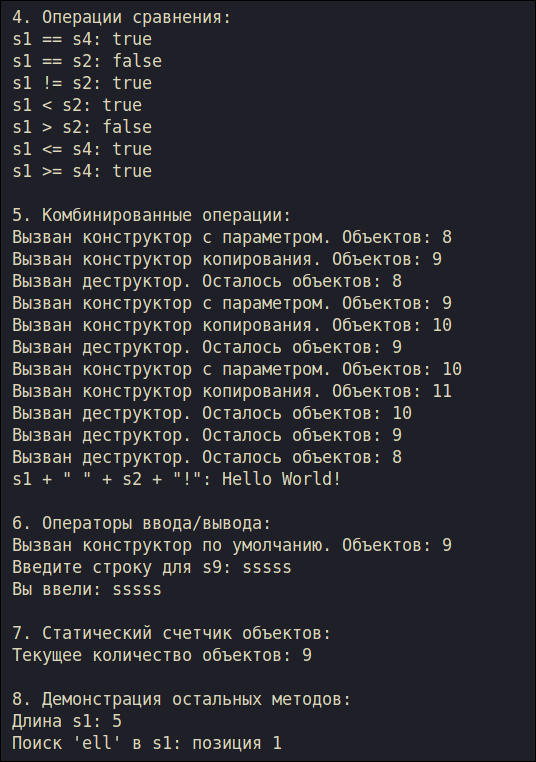
1. Тестирование

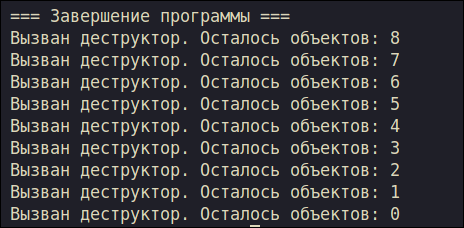
Ожидаемые результаты:

Мы ждем полную реализацию задания, с выводом всех промежуточных процессов, для отладки.

Результат:







ЗАКЛЮЧЕНИЕ

В процессе выполнения лабораторной работы был успешно разработан и реализован класс String для работы со строками, который полностью соответствует поставленным задачам. Основной акцент был сделан на реализации механизма перегрузки операций, что позволило создать интуитивно понятный интерфейс для работы со строковыми данными.

**Проблемы, возникшие при разработке:**  
Основные сложности возникли при реализации оператора присваивания, где требовалось обеспечить корректную обработку случая самоприсваивания и предотвратить утечки памяти. Также требовалось тщательно проработать логику работы с пустыми строками во всех операциях, чтобы избежать неопределенного поведения. При реализации дружественных функций возникла необходимость обеспечения согласованности с методами класса.

**Достигнутые результаты:**  
Разработанный класс представляет собой законченный тип данных, который может использоваться в выражениях естественным образом, аналогично встроенным типам. Реализация обеспечивает безопасность работы с памятью, демонстрирует различные подходы к перегрузке операторов (методы класса и дружественные функции) и предоставляет удобный интерфейс для работы со строками.

**Ограничения и возможные улучшения:**  
В текущей реализации не реализованы операторы составного присваивания (+=), индексации ([]) и приведения типов. Для промышленного использования класс мог бы быть дополнен поддержкой Unicode, методами для поиска и замены подстрок, а также оптимизацией производительности через использование move-семантики.

**Вывод:**  
Результат работы полностью удовлетворяет требованиям задания. Были успешно изучены и применены на практике механизмы перегрузки операций, работы с динамической памятью, использования статических членов класса и дружественных функций. Разработанный класс демонстрирует все ключевые аспекты объектно-ориентированного программирования и может служить основой для более сложных строковых классов.

ПРИЛОЖЕНИЕ

String.h

#ifndef STRING\_H

#define STRING\_H

#include <iostream>

class String {

private:

char\* str; // Указатель на строку

int length; // Длина строки

static int count; // Статический счетчик объектов

public:

// Конструкторы и деструктор

String(); // Конструктор по умолчанию

String(const char\* s); // Конструктор с параметром

String(const String& other); // Конструктор копирования

~String(); // Деструктор

// Методы работы со строками

void setString(const char\* s); // Изменение строки

void print() const; // Вывод строки

int findSubstring(const char\* substr) const; // Поиск подстроки

const char\* getString() const { return str ? str : ""; } // Получение строки

int getLength() const { return length; } // Получение длины

// Перегруженные операторы (методы класса)

String operator+(const String& other) const; // Сложение двух объектов

String operator+(const char\* other) const; // Сложение с char\*

String& operator=(const String& other); // Присваивание

bool operator==(const String& other) const; // Сравнение ==

bool operator!=(const String& other) const; // Сравнение !=

bool operator<(const String& other) const; // Сравнение <

bool operator>(const String& other) const; // Сравнение >

bool operator<=(const String& other) const; // Сравнение <=

bool operator>=(const String& other) const; // Сравнение >=

// Дружественные функции

friend String operator+(const char\* lhs, const String& rhs); // char\* + String

friend std::ostream& operator<<(std::ostream& os, const String& obj); // Вывод

friend std::istream& operator>>(std::istream& is, String& obj); // Ввод

// Статический метод

static int getCount(); // Получение количества объектов

};

#endif

String.cpp

#include "String.h"

#include <cstring>

#include <iostream>

// Инициализация статической переменной

int String::count = 0;

// Конструктор по умолчанию

String::String() : str(nullptr), length(0) {

count++;

std::cout << "Вызван конструктор по умолчанию. Объектов: " << count << std::endl;

}

// Конструктор с параметром

String::String(const char\* s) {

if (s) {

length = strlen(s);

str = new char[length + 1];

strcpy(str, s);

} else {

length = 0;

str = nullptr;

}

count++;

std::cout << "Вызван конструктор с параметром. Объектов: " << count << std::endl;

}

// Конструктор копирования

String::String(const String& other) {

if (other.str) {

length = other.length;

str = new char[length + 1];

strcpy(str, other.str);

} else {

length = 0;

str = nullptr;

}

count++;

std::cout << "Вызван конструктор копирования. Объектов: " << count << std::endl;

}

// Деструктор

String::~String() {

delete[] str;

count--;

std::cout << "Вызван деструктор. Осталось объектов: " << count << std::endl;

}

// Изменение строки

void String::setString(const char\* s) {

delete[] str;

if (s) {

length = strlen(s);

str = new char[length + 1];

strcpy(str, s);

} else {

length = 0;

str = nullptr;

}

}

// Вывод строки

void String::print() const {

if (str) std::cout << "Строка: " << str << " (длина: " << length << ")" << std::endl;

else std::cout << "Строка пуста." << std::endl;

}

// Поиск подстроки

int String::findSubstring(const char\* substr) const {

if (!str || !substr) return -1;

char\* pos = strstr(str, substr);

return pos ? pos - str : -1;

}

// Оператор сложения двух объектов (String + String)

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

if (!str && !other.str) return String();

if (!str) return String(other.str);

if (!other.str) return String(str);

char\* newStr = new char[length + other.length + 1];

strcpy(newStr, str);

strcat(newStr, other.str);

String result(newStr);

delete[] newStr;

return result;

}

// Оператор сложения с char\* (String + char\*)

String String::operator+(const char\* other) const {

if (!str && !other) return String();

if (!str) return String(other);

if (!other) return String(str);

int otherLen = strlen(other);

char\* newStr = new char[length + otherLen + 1];

strcpy(newStr, str);

strcat(newStr, other);

String result(newStr);

delete[] newStr;

return result;

}

// Оператор присваивания

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

if (this != &other) {

delete[] str;

if (other.str) {

length = other.length;

str = new char[length + 1];

strcpy(str, other.str);

} else {

length = 0;

str = nullptr;

}

}

std::cout << "Вызван оператор присваивания." << std::endl;

return \*this;

}

// Операторы сравнения

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

if (!str && !other.str) return true;

if (!str || !other.str) return false;

return strcmp(str, other.str) == 0;

}

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

return !(\*this == other);

}

bool String::operator<(const String& other) const {

if (!str && !other.str) return false;

if (!str) return true;

if (!other.str) return false;

return strcmp(str, other.str) < 0;

}

bool String::operator>(const String& other) const {

if (!str && !other.str) return false;

if (!str) return false;

if (!other.str) return true;

return strcmp(str, other.str) > 0;

}

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

return !(\*this > other);

}

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

return !(\*this < other);

}

// Дружественная функция: char\* + String

String operator+(const char\* lhs, const String& rhs) {

if (!lhs && !rhs.str) return String();

if (!lhs) return String(rhs.str);

if (!rhs.str) return String(lhs);

int lhsLen = strlen(lhs);

char\* newStr = new char[lhsLen + rhs.length + 1];

strcpy(newStr, lhs);

strcat(newStr, rhs.str);

String result(newStr);

delete[] newStr;

return result;

}

// Дружественная функция: вывод

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

if (obj.str) os << obj.str;

else os << "";

return os;

}

// Дружественная функция: ввод

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

char buffer[1000];

is.getline(buffer, 1000);

obj.setString(buffer);

return is;

}

// Получение количества объектов

int String::getCount() {

return count;

}

main.cpp

#include "String.h"

#include <iostream>

using namespace std;

int main() {

cout << "=== Демонстрация перегруженных операций для класса String ===" << endl << endl;

// Создание объектов

cout << "1. Создание объектов:" << endl;

String s1("Hello");

String s2("World");

String s3;

String s4 = s1; // Конструктор копирования

cout << endl << "2. Оператор присваивания:" << endl;

s3 = s2; // Оператор присваивания

cout << "s3 = s2: " << s3 << endl;

cout << endl << "3. Операции сложения:" << endl;

// String + String

String s5 = s1 + s2;

cout << "s1 + s2: " << s5 << endl;

// String + char\*

String s6 = s1 + " C++";

cout << "s1 + \" C++\": " << s6 << endl;

// char\* + String

String s7 = "C++ " + s2;

cout << "\"C++ \" + s2: " << s7 << endl;

cout << endl << "4. Операции сравнения:" << endl;

cout << "s1 == s4: " << (s1 == s4 ? "true" : "false") << endl;

cout << "s1 == s2: " << (s1 == s2 ? "true" : "false") << endl;

cout << "s1 != s2: " << (s1 != s2 ? "true" : "false") << endl;

cout << "s1 < s2: " << (s1 < s2 ? "true" : "false") << endl;

cout << "s1 > s2: " << (s1 > s2 ? "true" : "false") << endl;

cout << "s1 <= s4: " << (s1 <= s4 ? "true" : "false") << endl;

cout << "s1 >= s4: " << (s1 >= s4 ? "true" : "false") << endl;

cout << endl << "5. Комбинированные операции:" << endl;

String s8 = s1 + " " + s2 + "!";

cout << "s1 + \" \" + s2 + \"!\": " << s8 << endl;

cout << endl << "6. Операторы ввода/вывода:" << endl;

String s9;

cout << "Введите строку для s9: ";

cin >> s9;

cout << "Вы ввели: " << s9 << endl;

cout << endl << "7. Статический счетчик объектов:" << endl;

cout << "Текущее количество объектов: " << String::getCount() << endl;

cout << endl << "8. Демонстрация остальных методов:" << endl;

cout << "Длина s1: " << s1.getLength() << endl;

cout << "Поиск 'ell' в s1: позиция " << s1.findSubstring("ell") << endl;

cout << endl << "=== Завершение программы ===" << endl;

return 0;

}