ФГБОУ ВО

Национальный исследовательский университет

«МЭИ»

Лабораторная работа № 4

по курсу

«Технология программирования»

ООП C++. ПЕРЕГРУЗКА ОПЕРАТОРОВ

Вариант 8

Выполнил:

студент группы Аэ-21-23

Нестеров А.С.

Москва, 2024

**1. Постановка задачи**

Задача: Создать класс для представления координат в двумерном пространстве. Класс должен обеспечивать следующие функции: инкремент и декремент координат с помощью унарных операторов, а также возможность изменения знака координат. Операторы инкремента и декремента должны быть реализованы в качестве методов класса и дружественных функций. Исключения должны выбрасываться при некорректном вводе координат.

**2. Разработка структуры приложения**

В данной программе описан класс Coordinates, который включает методы для управления координатами и их отображения.

**2.1 Описание полей и методов класса**

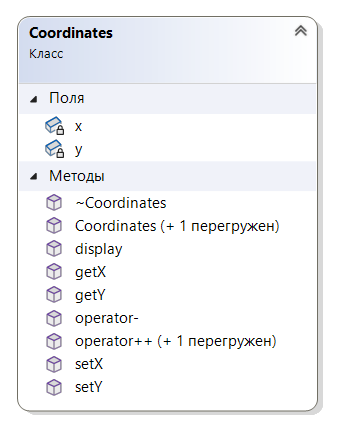
**Класс Coordinates:**

**Поля:**

* int x: координата X.
* int y: координата Y.

**Методы:**

* Coordinates(): конструктор по умолчанию.
* Coordinates(int x\_, int y\_): конструктор с параметрами, устанавливающий координаты и выполняющий валидацию.
* void setX(int x\_): метод для установки координаты X с валидацией.
* void setY(int y\_): метод для установки координаты Y с валидацией.
* int getX() const: метод для получения координаты X.
* int getY() const: метод для получения координаты Y.
* void display() const: метод для отображения координат.
* Coordinates& operator++(): префиксный оператор инкремента.
* Coordinates operator++(int): постфиксный оператор инкремента.
* friend Coordinates& operator--(Coordinates& coord): префиксный оператор декремента как дружественная функция.
* friend Coordinates operator--(Coordinates& coord, int): постфиксный оператор декремента как дружественная функция.
* Coordinates operator-() const: оператор для изменения знака координат.



*Рис. 1. Диаграмма классов*

**2.2 Разработка пользовательского интерфейса**

Категории пользователей: Студенты и преподаватели, желающие протестировать функциональность программы.

**Функции пользователя (интерфейс):**

* Ввод координат X и Y.
* Установка значений координат.
* Инкремент и декремент координат.
* Изменение знака координат.
* Отображение текущих координат.
* Завершение программы.

Интерфейс программы реализован в виде текстового меню, которое позволяет пользователю выбирать операции.

**3. Реализация и тестирование приложения**

**Описание разработанной программы**

Данная программа написана на языке C++ и включает класс Coordinates для управления координатами. Основная задача программы — демонстрация работы с классами, обработка исключений и валидация данных.

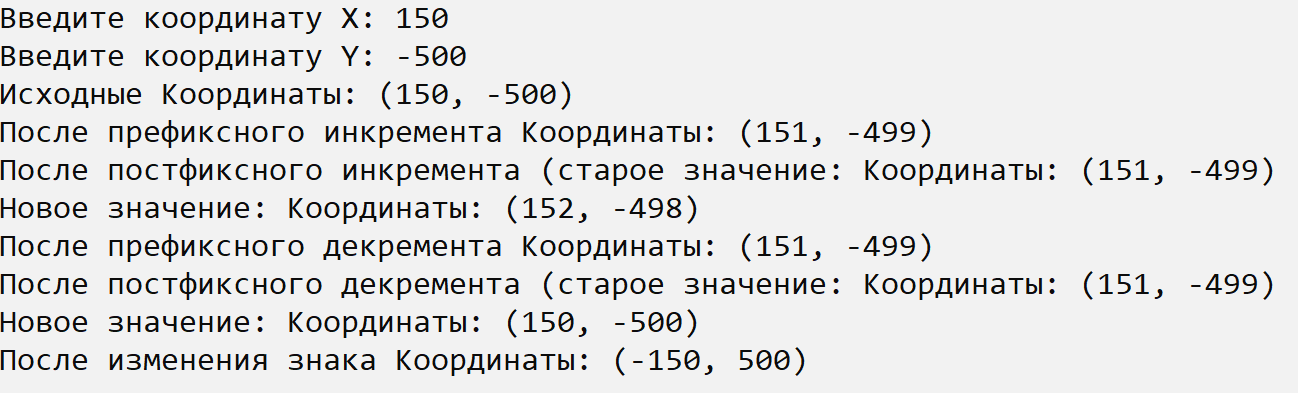
**Тестирование программы**

Тестирование программы включает:

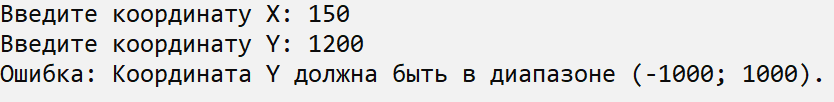
* Нормальные условия: ввод корректных данных и успешное выполнение операций.
* Исключительные условия: ввод некорректных данных, таких как недопустимые координаты.

**Таблица тестирования**

| **№ теста** | **Тестируемая функция** | **Входные данные** | **Ожидаемый результат** | **Цель теста** |
| --- | --- | --- | --- | --- |
| 1 | Coordinates() | - | Координаты: (0, 0) | Проверка конструктора по умолчанию |
| 2 | Coordinates(int, int) | 150, -500 | Координаты: (100, -500) | Проверка корректной инициализации с параметрами |
| 3 | setX(int) | 150 | Успешно устанавливает X на 150 | Проверка установки координаты X |
| 4 | setX(int) | 1200 | Исключение: «Координата X должна быть в диапазоне (-1000; 1000).» | Проверка валидации координаты X |
| 5 | setY(int) | -500 | Успешно устанавливает Y на -500 | Проверка установки координаты Y |
| 6 | operator++() | - | Координаты: (151, -499) | Проверка префиксного инкремента |
| 7 | operator++(int) | - | Координаты: (152, -498) | Проверка постфиксного инкремента |
| 8 | operator—() | - | Координаты: (151, -499) | Проверка префиксного декремента |
| 9 | operator—(int) | - | Координаты: (150, -500) | Проверка постфиксного декремента |
| 10 | operator-() | - | Координаты: (-150, 500) | Проверка изменения знака координат |



*Рис. Тест 2-3,5-10*



*Рис. Тест 4*

**Вывод**

В данной лабораторной работе изучены принципы работы с классами в C++, включая создание, перегрузку операторов и обработку исключений. Класс Coordinates демонстрирует основные концепции объектно-ориентированного программирования и управление памятью. Программа успешно обрабатывает различные сценарии ввода и предоставляет пользователю удобный интерфейс для работы с координатами.

**Код.**

#include <iostream>

#include <windows.h>

#include <stdexcept>

class Coordinates {

private:

int x; // координата X

int y; // координата Y

public:

// Конструктор по умолчанию

Coordinates() = default;

// Конструктор с параметрами

Coordinates(int x\_, int y\_) {

setX(x\_);

setY(y\_);

}

// Деструктор

~Coordinates() {

// Нет динамической памяти для освобождения

}

// Геттеры

int getX() const {

return x;

}

int getY() const {

return y;

}

// Сеттеры с обработкой исключений

void setX(int x\_) {

if (x\_ < -1000 || x\_ > 1000) {

throw std::invalid\_argument("Координата X должна быть в диапазоне (-1000; 1000).");

}

x = x\_;

}

void setY(int y\_) {

if (y\_ < -1000 || y\_ > 1000) {

throw std::invalid\_argument("Координата Y должна быть в диапазоне (-1000; 1000).");

}

y = y\_;

}

// Метод для отображения координат

void display() const {

std::cout << "Координаты: (" << x << ", " << y << ")\n";

}

// Префиксный оператор ++ как функция-член

Coordinates& operator++() {

++(x);

++(y);

return \*this;

}

// Постфиксный оператор ++

Coordinates operator++(int) {

Coordinates temp = \*this; // сохраняем текущее состояние

++(x);

++(y);

return temp; // возвращаем старое состояние

}

// Префиксный оператор -- как дружественная функция

friend Coordinates& operator--(Coordinates& coord);

// Постфиксный оператор -- как дружественная функция

friend Coordinates operator--(Coordinates& coord, int);

// Оператор - для изменения знака

Coordinates operator-() const {

return Coordinates(-(x), -(y));

}

};

Coordinates& operator--(Coordinates& coord) {

--(coord.x);

--(coord.y);

return coord;

}

Coordinates operator--(Coordinates& coord, int) {

Coordinates temp = coord; // сохраняем текущее состояние

--(coord.x);

--(coord.y);

return temp; // возвращаем старое состояние

}

void clearInputBuffer() {

std::cin.clear(); // сбрасываем флаг ошибки

while (std::cin.get() != '\n'); // очищаем буфер ввода до конца строки

}

int main() {

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

system("color F0");

int x, y;

// Ввод координаты X с обработкой исключений

while (true) {

try {

std::cout << "Введите координату X: ";

std::cin >> x;

if (std::cin.fail()) {

throw std::invalid\_argument("Некорректный ввод. Пожалуйста, введите целое число.");

}

clearInputBuffer(); // очищаем буфер ввода

break; // выходим из цикла, если ввод успешен

}

catch (const std::invalid\_argument& e) {

std::cerr << "Ошибка: " << e.what() << std::endl;

clearInputBuffer(); // очищаем буфер ввода

}

}

// Ввод координаты Y с обработкой исключений

while (true) {

try {

std::cout << "Введите координату Y: ";

std::cin >> y;

if (std::cin.fail()) {

throw std::invalid\_argument("Некорректный ввод. Пожалуйста, введите целое число.");

}

clearInputBuffer(); // очищаем буфер ввода

break; // выходим из цикла, если ввод успешен

}

catch (const std::invalid\_argument& e) {

std::cerr << "Ошибка: " << e.what() << std::endl;

clearInputBuffer(); // очищаем буфер ввода

}

}

// Создаем объект с введенными координатами

Coordinates\* coord = nullptr;

try {

coord = new Coordinates(x, y);

}

catch (const std::invalid\_argument& e) {

std::cerr << "Ошибка: " << e.what() << std::endl;

return 1;

}

std::cout << "Исходные ";

coord->display();

// Префиксный инкремент

++(\*coord);

std::cout << "После префиксного инкремента ";

coord->display();

// Постфиксный инкремент

Coordinates oldCoord = (\*coord)++;

std::cout << "После постфиксного инкремента (старое значение: ";

oldCoord.display();

std::cout << "Новое значение: ";

coord->display();

// Префиксный декремент

--(\*coord);

std::cout << "После префиксного декремента ";

coord->display();

// Постфиксный декремент

oldCoord = (\*coord)--;

std::cout << "После постфиксного декремента (старое значение: ";

oldCoord.display();

std::cout << "Новое значение: ";

coord->display();

// Изменение знака координат

Coordinates negated = -(\*coord);

std::cout << "После изменения знака ";

negated.display();

delete coord; // Освобождение памяти

return 0;

}

**1. Постановка задачи**

Задача: Создать класс для целых чисел (long) с возможностью выполнения бинарных операций. Класс должен обеспечивать перегрузку операторов вычитания, сложения, присвоения и сравнений. Оператор вычитания должен быть реализован как метод класса, а оператор сложения — как дружественная функция. Также необходимо реализовать обработку некорректного ввода данных.

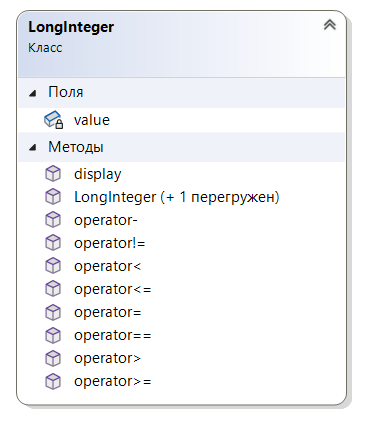
**2. Разработка структуры приложения**

В данной программе описан класс **LongInteger**, который включает методы для управления целыми числами и их отображения.

**2.1 Описание полей и методов класса**

Класс **LongInteger**:

* **Поля:**
  + **long value**: значение целого числа.
* **Методы:**
  + **LongInteger()**: конструктор по умолчанию, инициализирующий значение 0.
  + **LongInteger(long value)**: конструктор с параметром, устанавливающий значение.
  + **LongInteger operator-(const LongInteger& other)**: оператор вычитания, возвращающий результат вычитания двух целых чисел.
  + **friend LongInteger operator+(const LongInteger& lhs, const LongInteger& rhs)**: дружественный оператор сложения.
  + **LongInteger& operator=(const LongInteger& other)**: оператор присвоения.
  + **bool operator==(const LongInteger& other)**: оператор сравнения на равенство.
  + **bool operator!=(const LongInteger& other)**: оператор сравнения на неравенство.
  + **bool operator<(const LongInteger& other)**: оператор сравнения на меньше.
  + **bool operator>(const LongInteger& other)**: оператор сравнения на больше.
  + **bool operator<=(const LongInteger& other)**: оператор сравнения на меньше или равно.
  + **bool operator>=(const LongInteger& other)**: оператор сравнения на больше или равно.
  + **void display() const**: метод для отображения значения.



*Рис. 1. Диаграмма классов*

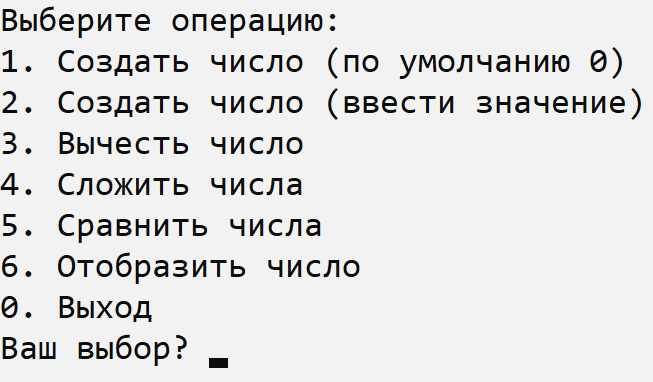
**2.2 Разработка пользовательского интерфейса**

Категории пользователей: Студенты и преподаватели, желающие протестировать функциональность программы.

**Функции пользователя (интерфейс):**

* Создание числа (по умолчанию или с пользовательским значением).
* Выполнение операций вычитания и сложения.
* Сравнение чисел.
* Отображение текущего значения числа.
* Завершение программы.

Интерфейс программы реализован в виде текстового меню, которое позволяет пользователю выбирать операции.



*Рис. 2. Пользовательский интерфейс*

**3. Реализация и тестирование приложения**

**Описание разработанной программы**

Данная программа написана на языке C++ и включает класс **LongInteger** для управления целыми числами. Основная задача программы — демонстрация работы с классами, обработка исключений и валидация данных.

**Тестирование программы**

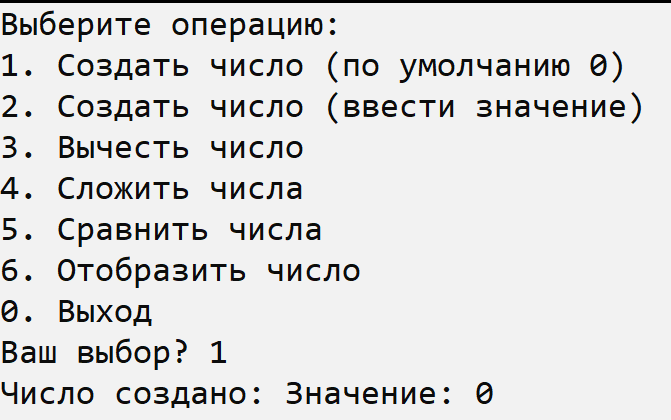
Тестирование программы включает:

* Нормальные условия: ввод корректных данных и успешное выполнение операций.
* Исключительные условия: ввод некорректных данных, таких как недопустимые значения.

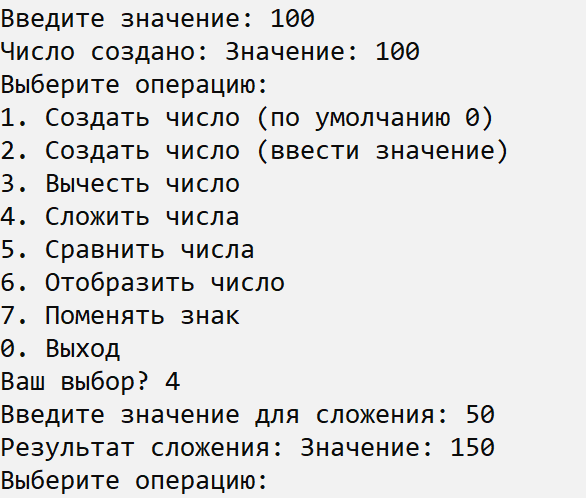
**Таблица тестирования**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№ теста** | **Тестируемая функция** | **Входные данные** | **Ожидаемый результат** | **Цель теста** |
| 1 | LongInteger() | - | Значение: 0 | Проверка конструктора по умолчанию |
| 2 | LongInteger(long) | 100 | LongInteger(100) | Проверка корректной инициализации с параметрами |
| 3 | operator+(LongInteger) | 50 | Значение: 150 | Проверка сложения |
| 4 | operator-(LongInteger) | 30 | Значение: 70 | Проверка вычитания |
| 5 | operator==(LongInteger) | 100 | num1 равно num2 (==) | Проверка сравнения на равенство |
| 6 | operator!=(LongInteger) | 50 | num1 не равно num2 (!=) | Проверка сравнения на неравенство |
| 7 | operator<(LongInteger) | | 150 | num1 меньше num2 (<) | Проверка сравнения на меньше |
| 8 | operator>(LongInteger) | 50 | num1 больше num2 (>) | Проверка сравнения на больше |
| 9 | operator<=(LongInteger) | 100 | num1 не больше num2 (<=) | Проверка сравнения на меньше или равно |
| 10 | operator>=(LongInteger) | 100 | num1 не меньше num2 (>=) | Проверка сравнения на больше или равно |

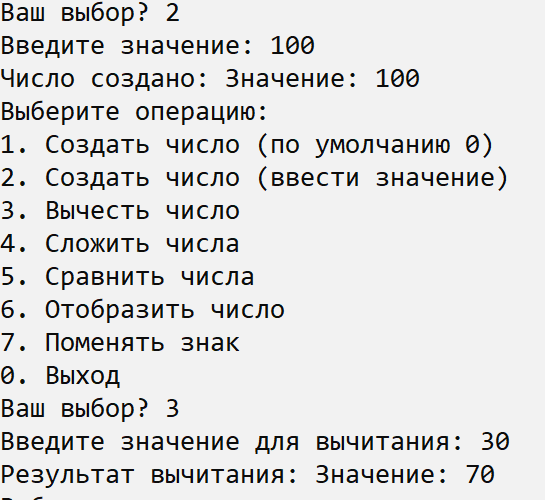
Приложения



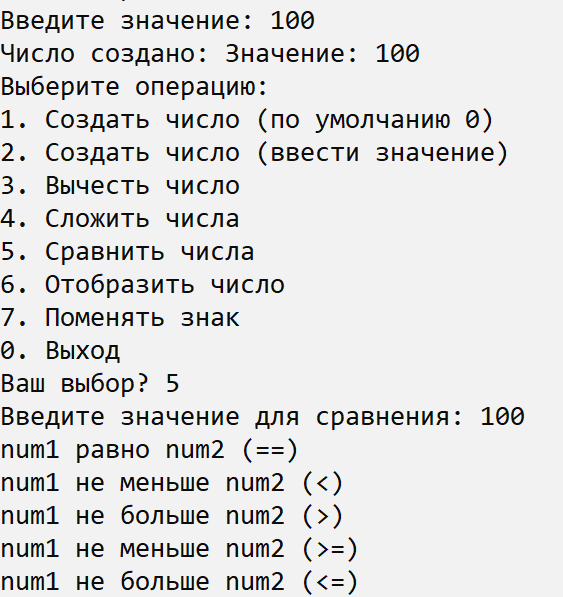
*Рис. 3. Тест 1*



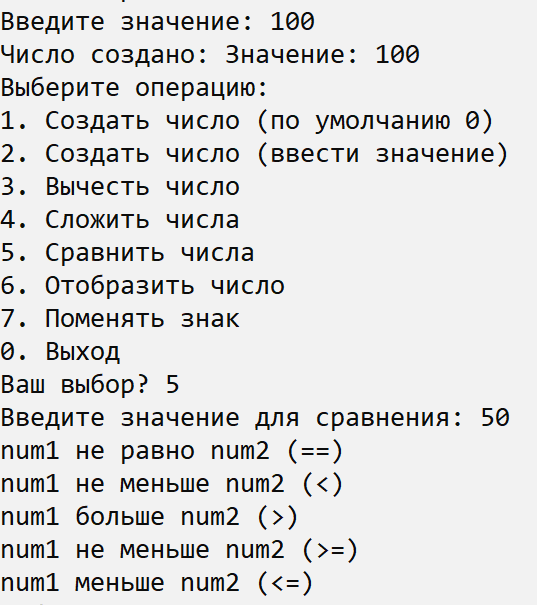
*Рис. 4. Тест 2,3*



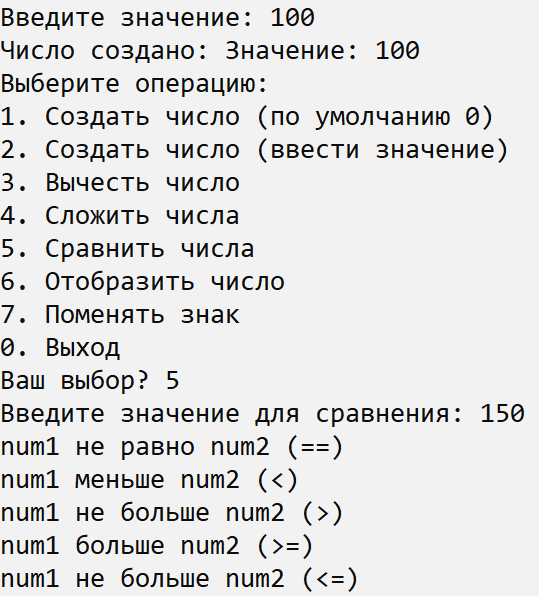
*Рис. 5. Тест 4*



*Рис. 6. Тест 5, 9, 10*



*Рис. 6. Тест 6,8*



*Рис. 7. Тест 7*