МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
Кафедра вычислительной техники

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №2
ПО ДИСЦИПЛИНЕ «ПРОГРАММИРОВАНИЕ»

Факультет: Заочное отделение Преподаватель: Дубков Илья Сергеевич

Группа: ДТ-460а

Студент: Дроздов Иван Сергеевич

Вариант: 0

Новосибирск, 2025 г.

Цель работы: 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.

Задание: Для разработанного класса из лабораторной работы №1 реализовать набор операций для работы с объектами класса: сложение (как метод класса), вычитание (как дружественную функцию), присваивание (как метод класса), инкремент постфиксный и инкремент префиксный (как методы класса), приведение к некоторому типу (как метод класса).
Дополнить демонстрационную программу, продемонстрировав все перегруженные операции.

Исходные коды модулей проекта:

Date.h
```
#ifndef DATE_H
#define DATE_H

class Date {
private:
    int day;
    int month;
    int year;
    char* weekDay;

public:
    // Геттеры для доступа к частным данным
    int getDay() const { return day; }
    int getMonth() const { return month; }
    int getYear() const { return year; }

    // Статический массив с количеством дней в каждом месяце
    static const int daysInMonth[];

    // Конструкторы и деструктор
    Date();
    Date(int d, int m, int y, const char* wd);
    Date(const Date& other);
    ~Date();

    // Методы класса
    void addDays(int numDays);
    Date& operator=(const Date& other);
    Date& operator++();          // Префиксный инкремент
    Date operator++(int dummy);   // Постфиксный инкремент
```

```
    const char* toString() const;
    void display() const;
};

// Функция для вычитания двух дат
int subtractDates(const Date& lhs, const Date& rhs);

#endif
```

Date.cpp

```
#include "Date.h"
#include <cstdio>
#include <cstring>

// Определение статического массива
const int Date::daysInMonth[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

// Конструктор по умолчанию
Date::Date() : day(1), month(1), year(2000) {
    printf("Конструктор по умолчанию\n");
    weekDay = new char[10];
    strcpy(weekDay, "Monday");
}

// Конструктор с параметрами
Date::Date(int d, int m, int y, const char* wd) : day(d), month(m), year(y) {
    printf("Конструктор с параметрами\n");
    weekDay = new char[strlen(wd) + 1];
    strcpy(weekDay, wd);
}

// Копирующий конструктор
Date::Date(const Date& other) : day(other.day), month(other.month), year(other.year) {
    printf("Копирующий конструктор\n");
    weekDay = new char[strlen(other.weekDay) + 1];
    strcpy(weekDay, other.weekDay);
}

// Деструктор
Date::~Date() {
    delete[] weekDay;
    printf("Освобождение памяти\n");
}

// Операция сложения: добавляет указанное количество дней к текущей дате
void Date::addDays(int numDays) {
    while(numDays > 0) {
        if(day + numDays <= daysInMonth[month]) {
            day += numDays;
            break;
        } else {
            int remainingDays = daysInMonth[month] - day + 1;
            numDays -= remainingDays;
            day = 1;

            if(month == 11) {
                month = 0;
                ++year;
            } else {
                ++month;
            }
        }
```

```
    }
}

// Присваивание (копирование объекта)
Date& Date::operator=(const Date& other) {
    if(this != &other) {
        this->day = other.day;
        this->month = other.month;
        this->year = other.year;

        delete[] weekDay;
        weekDay = new char[strlen(other.weekDay) + 1];
        strcpy(weekDay, other.weekDay);
    }

    return *this;
}

// Префиксный инкремент
Date& Date::operator++() {
    addDays(1);
    return *this;
}

// Постфиксный инкремент
Date Date::operator++(int dummy) {
    Date temp(*this);
    addDays(1);
    return temp;
}

// Приведение к строке
const char* Date::toString() const {
    static char buffer[11];
    snprintf(buffer, sizeof(buffer), "%02d-%02d-%04d", day, month+1, year);
    return buffer;
}

// Метод для демонстрации
void Date::display() const {
    printf("Дата: %d.%d.%d, День недели: %s\n", day, month+1, year, weekDay);
}

// Функция для вычитания двух дат
int subtractDates(const Date& lhs, const Date& rhs) {
    return ((lhs.getYear() - rhs.getYear())*365 +
            (lhs.getMonth() - rhs.getMonth())*30 +
            (lhs.getDay() - rhs.getDay()));
}

//L2.cpp
#include "Date.h"
#include <cstdio>

int main() {
    // Тестовая дата
    Date myDate(1, 1, 2023, "Monday");
    myDate.display();

    // Сложение (добавление дней)
    myDate.addDays(365);
    myDate.display();
```

```
    // Присваивание (создание копии)
    Date anotherDate(myDate);
    anotherDate.display();

    // Префиксный инкремент
    ++anotherDate;
    anotherDate.display();

    // Постфиксный инкремент
    Date prevState = anotherDate++;
    prevState.display();
    anotherDate.display();

    // Приведение к строке
    printf("Строковое представление: %s\n", anotherDate.toString());

    // Разность дат
    Date earlierDate(1, 1, 2022, "Sunday");
    int diff = subtractDates(anotherDate, earlierDate);
    printf("Разница в днях: %d\n", diff);

    return 0;
}
```

Выводы:

В рамках выполнения второй лабораторной работы по дисциплине «Программирование» была углублена практика работы с классами в C++. Основной фокус был направлен на изучение механизмов перегрузки операторов и применения дружественных функций, что позволяет определять для пользовательских типов данных поведение, аналогичное встроенным типам языка.

На практике это реализовано путём расширения ранее разработанного класса из лабораторной работы №1. Для него был определён набор ключевых операций, демонстрирующих различные способы перегрузки:

- Сложение и присваивание были реализованы как методы класса, подчеркивая их тесную связь с внутренним состоянием объекта.
- Вычитание организовано через дружественную функцию, что иллюстрирует возможность доступа к приватным членам класса извне при необходимости.
- Реализованы обе формы инкремента (префиксная и постфиксная), наглядно показывающие синтаксические и семантические различия между ними.

- Операция приведения к типу была добавлена как метод класса, обеспечивая механизм явного преобразования объекта к другому типу данных.

В процессе выполнения работы были закреплены следующие концепции:

- Принципы и синтаксис перегрузки операторов для пользовательских типов.
- Назначение, объявление и использование дружественных функций и классов.
- Критерии выбора между реализацией оператора как метода класса или как дружественной функции.
- Особенности реализации унарных (инкремент) и бинарных (сложение, вычитание) операторов.

Результатом работы стала модифицированная демонстрационная программа, которая комплексно представляет функциональность разработанного класса, наглядно иллюстрируя корректность работы всех перегруженных операций и подтверждая создание полноценного и удобного в использовании типа данных.