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

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №3
ПО ДИСЦИПЛИНЕ «ПРОГРАММИРОВАНИЕ»

Факультет: Заочное отделение Преподаватель: Дубков Илья Сергеевич

Группа: ДТ-460а

Студент: Дроздов Иван Сергеевич

Вариант: 0

Новосибирск, 2025 г.

Цель работы: Изучить работу потоков ввода-вывода и реализацию перегрузки потоков ввода-вывода на стандартные устройства и в файл для разработанных классов.

Задание: Для класса из лабораторной работы №2 перегрузить операции ввода/вывода, позволяющие осуществлять ввод и вывод в удобной форме объектов классов:

- вывод объекта класса в текстовый файл;
- вывод объекта класса в двоичный файл;
- ввод объекта класса из двоичного файла.

Дополнить демонстрационную программу, продемонстрировав все перегруженные операции.

Исходные коды модулей проекта:

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

#include <iostream>

class Date {
private:
    int day;
    int month;  // 0-11
    int year;
    char* weekDay;

public:
    // Геттеры
    int getDay() const { return day; }
    int getMonth() const { return month; }
    int getYear() const { return year; }
    const char* getWeekDay() const { return weekDay; }

    // Статический массив
    static const int daysInMonth[];

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

    // Основные операции с датами
    void addDays(int numDays);

    // Операторы
```

```
    Date& operator=(const Date& other);
    Date& operator++();
    Date operator++(int);

    // Методы представления
    const char* toString() const;
    void display() const;

    // Вспомогательные методы
    bool isLeapYear() const;
    int getDaysInMonth() const;

    // Операторы ввода/вывода
    friend std::ostream& operator<<(std::ostream& os, const Date& date);
    friend std::istream& operator>>(std::istream& is, Date& date);

    // Дружественные функции для бинарного ввода/вывода
    friend void writeDateToBinaryFile(const Date& date, const char* filename);
    friend Date readDateFromBinaryFile(const char* filename);
};

#endif
```

Date.cpp

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

// 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(0), year(2000) {
    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 - 1), year(y) {
    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) {
    weekDay = new char[strlen(other.weekDay) + 1];
    strcpy(weekDay, other.weekDay);
}

// Деструктор
Date::~Date() {
    delete[] weekDay;
}

// Проверка на високосный год
bool Date::isLeapYear() const {
    return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
}

// Получение количества дней в текущем месяце
int Date::getDaysInMonth() const {
```

```
    if (month == 1) {  // Февраль
        return isLeapYear() ? 29 : 28;
    }
    return daysInMonth[month];
}

// Операция сложения дней (ИСПРАВЛЕННАЯ)
void Date::addDays(int numDays) {
    day += numDays;

    while (day > getDaysInMonth()) {
        day -= getDaysInMonth();
        month++;

        if (month >= 12) {
            month = 0;
            year++;
        }
    }
}

// Оператор присваивания
Date& Date::operator=(const Date& other) {
    if (this != &other) {
        day = other.day;
        month = other.month;
        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) {
    Date temp(*this);
    addDays(1);
    return temp;
}

// Приведение к строке
const char* Date::toString() const {
    static char buffer[20];
    snprintf(buffer, sizeof(buffer), "%02d.%02d.%04d", day, month + 1, year);
    return buffer;
}

// Метод для демонстрации
void Date::display() const {
    printf("Дата: %02d.%02d.%04d, День недели: %s\n", day, month + 1, year, weekDay);
}

// Оператор вывода
std::ostream& operator<<(std::ostream& os, const Date& date) {
```

```
    os << date.day << " " << (date.month + 1) << " " << date.year << " " << date.weekDay;
    return os;
}

// Оператор ввода
std::istream& operator>>(std::istream& is, Date& date) {
    int tempDay, tempMonth, tempYear;
    char tempWeekDay[100];

    is >> tempDay >> tempMonth >> tempYear >> tempWeekDay;

    // Проверка корректности ввода
    if (tempMonth < 1 || tempMonth > 12) {
        is.setstate(std::ios::failbit);
        return is;
    }

    date.day = tempDay;
    date.month = tempMonth - 1;
    date.year = tempYear;

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

    return is;
}
```

DateUtils.h

```
#ifndef DATE_UTILS_H
#define DATE_UTILS_H

#include "Date.h"

// Утилиты для работы с датами
int subtractDates(const Date& lhs, const Date& rhs);

#endif
```

DateUtils.cpp

```
#include "DateUtils.h"

// Функция для вычитания двух дат (упрощенная)
int subtractDates(const Date& lhs, const Date& rhs) {
    // Простая реализация - разница в днях без учета месяцев разной длины
    int days1 = lhs.getYear() * 365 + lhs.getMonth() * 30 + lhs.getDay();
    int days2 = rhs.getYear() * 365 + rhs.getMonth() * 30 + rhs.getDay();
    return days1 - days2;
}
```

L3.cpp

```
#include "Date.h"
#include "DateUtils.h"
#include <cstdio>
#include <cstring>
#include <fstream>
#include <iostream>

// Функция для записи Date в бинарный файл (ИСПРАВЛЕННАЯ)
void writeDateToBinaryFile(const Date& date, const char* filename) {
    std::ofstream file(filename, std::ios::binary);
    if (file.is_open()) {
```

```
        // Прямой доступ к приватным полям через дружественную функцию
        file.write(reinterpret_cast<const char*>(&date.day), sizeof(int));
        file.write(reinterpret_cast<const char*>(&date.month), sizeof(int));
        file.write(reinterpret_cast<const char*>(&date.year), sizeof(int));

        // Записываем строку БЕЗ нулевого терминатора
        int length = strlen(date.weekDay);
        file.write(reinterpret_cast<const char*>(&length), sizeof(int));
        file.write(date.weekDay, length);  // Только символы, без '\0'

        file.close();
        printf("Объект записан в бинарный файл '%s'\n", filename);

        // Отладочная информация
        printf("Записано: day=%d, month=%d, year=%d, weekDay='%s' (length=%d)\n",
               date.day, date.month, date.year, date.weekDay, length);
    } else {
        printf("Ошибка записи в файл '%s'\n", filename);
    }
}

// Функция для чтения Date из бинарного файла (ИСПРАВЛЕННАЯ)
Date readDateFromBinaryFile(const char* filename) {
    std::ifstream file(filename, std::ios::binary);
    Date date;

    if (file.is_open()) {
        // Прямой доступ к приватным полям через дружественную функцию
        file.read(reinterpret_cast<char*>(&date.day), sizeof(int));
        file.read(reinterpret_cast<char*>(&date.month), sizeof(int));
        file.read(reinterpret_cast<char*>(&date.year), sizeof(int));

        int length;
        file.read(reinterpret_cast<char*>(&length), sizeof(int));

        // Проверяем корректность длины
        if (length > 0 && length < 100) {
            delete[] date.weekDay;
            date.weekDay = new char[length + 1];
            file.read(date.weekDay, length);
            date.weekDay[length] = '\0';  // Добавляем терминатор
        } else {
            // Значение по умолчанию при ошибке
            delete[] date.weekDay;
            date.weekDay = new char[10];
            strcpy(date.weekDay, "Unknown");
        }

        file.close();
        printf("Объект загружен из бинарного файла '%s'\n", filename);
        printf("Прочитано: day=%d, month=%d, year=%d, weekDay='%s'\n",
               date.day, date.month, date.year, date.weekDay);
    } else {
        printf("Ошибка чтения файла '%s'\n", filename);
    }

    return date;
}

// Функция для вывода hex-дампа файла
void printHexDump(const char* filename) {
    std::ifstream file(filename, std::ios::binary);
```

```
if (file.is_open()) {
    printf("Hex-дамп файла '%s':\n", filename);
    char buffer[256];
    std::streamsize totalBytes = 0;

    while (file.read(buffer, 16)) {
        std::streamsize bytesRead = file.gcount();

        // Вывод адреса
        printf("%08lx: ", totalBytes);

        // Hex значения
        for (int i = 0; i < bytesRead; i++) {
            printf("%02x ", (unsigned char)buffer[i]);
            if (i == 7) printf(" ");
        }

        // Заполнение пробелами
        for (int i = bytesRead; i < 16; i++) {
            printf("   ");
            if (i == 7) printf(" ");
        }

        printf(" ");

        // ASCII значения
        for (int i = 0; i < bytesRead; i++) {
            if (buffer[i] >= 32 && buffer[i] <= 126) {
                printf("%c", buffer[i]);
            } else {
                printf(".");
            }
        }

        printf("\n");
        totalBytes += bytesRead;
    }

    // Обработка последней неполной строки
    std::streamsize bytesRead = file.gcount();
    if (bytesRead > 0) {
        printf("%08lx: ", totalBytes);
        for (int i = 0; i < bytesRead; i++) {
            printf("%02x ", (unsigned char)buffer[i]);
            if (i == 7) printf(" ");
        }
        for (int i = bytesRead; i < 16; i++) {
            printf("   ");
            if (i == 7) printf(" ");
        }
        printf(" ");
        for (int i = 0; i < bytesRead; i++) {
            if (buffer[i] >= 32 && buffer[i] <= 126) {
                printf("%c", buffer[i]);
            } else {
                printf(".");
            }
        }
        printf("\n");
        totalBytes += bytesRead;
    }
```

```
        file.close();
        printf("Всего байт: %ld\n", totalBytes);
    } else {
        printf("Не удалось открыть файл '%s' для чтения\n", filename);
    }
}

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

    // Запись в текстовый файл
    std::ofstream outFile("date.txt");
    if (outFile.is_open()) {
        outFile << myDate;
        outFile.close();
        printf("Объект записан в текстовый файл 'date.txt'\n");
    }

    // Чтение из текстового файла
    Date loadedDate;
    std::ifstream inFile("date.txt");
    if (inFile.is_open()) {
        inFile >> loadedDate;
        inFile.close();
        printf("Объект загружен из текстового файла 'date.txt'\n");
        loadedDate.display();
    }

    // Работа с бинарными файлами
    printf("\n--- Работа с бинарными файлами ---\n");
    writeDateToBinaryFile(myDate, "date.bin");

    // Выводим hex-дамп
    printHexDump("date.bin");

    // Чтение из бинарного файла
    Date binaryDate = readDateFromBinaryFile("date.bin");
    printf("Дата из бинарного файла: ");
    binaryDate.display();

    // Демонстрация работы утилит
    Date earlierDate(1, 1, 2022, "Sunday");
    int diff = subtractDates(myDate, earlierDate);
    printf("Разница в днях: %d\n", diff);

    return 0;
}
```

date.txt
1 1 2023 Monday

date.bin
```
00000000: 0100 0000 0000 0000 e707 0000 0600 0000  ................
00000010: 4d6f 6e64 6179 0a                        Monday.
```

Где

01 00 00 00 — день 1

00 00 00 00 — месяц 0 в формате 0-11(Январь)

e7 07 00 00 — год 2023

06 00 00 00 — длинна строки — 6

4d 6f 6e 64 61 79 - «Monday»

Выводы:

В ходе выполнения третьей лабораторной работы по дисциплине «Программирование» было проведено углублённое изучение системы ввода-вывода языка C++ и механизмов её интеграции с пользовательскими типами данных. Основной практической задачей являлась реализация перегрузки операторов для потокового ввода и вывода, что обеспечивает удобный и интуитивно понятный интерфейс для сериализации объектов в различные форматы.

На основе ранее разработанного класса был реализован полный цикл работы с потоками:

- **Перегрузка оператора вывода** для записи объекта в текстовый файл в удобочитаемом формате, что позволяет легко просматривать и анализировать данные.
- **Перегрузка того же оператора для бинарного вывода**, обеспечивающая компактное и эффективное сохранение состояния объекта в файл без преобразований.
- **Перегрузка оператора ввода** для бинарного чтения, позволяющая корректно восстановить состояние объекта непосредственно из двоичного потока данных.

В процессе выполнения работы были закреплены и практически применены следующие ключевые аспекты:

- Принципы работы с потоками ввода-вывода стандартной библиотеки.

- Синтаксис и семантика перегрузки операторов для пользовательских классов, в том числе необходимость объявления их дружественными функциями.

- Критические различия между текстовым и бинарным форматами данных, а также особенности работы с ними на уровне потоков.

- Организация устойчивого доступа к файлам и механизмы контроля целостности данных при сериализации и десериализации.

Итогом работы стало существенное расширение демонстрационной программы, которая наглядно подтвердила корректность всех реализованных операций. Программа демонстрирует полный жизненный цикл объекта: от его создания и вывода в текстовом и бинарном представлении до последующего чтения и восстановления его состояния из бинарного файла, что подчеркивает создание целостного и автономного типа данных.

Что такое поток ввода-вывода?

Поток (stream) в C++ — это абстракция, представляющая последовательность данных, передаваемую от источника (например, клавиатуры, файла) к приемнику (например, экрану, файлу) или наоборот. Потоки скрывают детали низкоуровневых операций, предоставляя унифицированный интерфейс для работы с данными.

- **Что такое файл? В каких форматах сохраняется информация в файлах?**

  Файл — это именованная область данных на носителе информации. В C++ информация в файлах может сохраняться в двух основных форматах:

  - **Текстовый формат:** Данные хранятся как последовательности символов (байт), интерпретируемые как текст. Например, числа преобразуются в их строковое представление.
  - **Бинарный (двоичный) формат:** Данные сохраняются в том виде, в каком они находятся в памяти компьютера (последовательность байт), без преобразований.

- **Зачем нужно переопределять операции ввода/вывода >> и << для объектов классов?**

  Переопределение (перегрузка) операторов `>>` и `<<` позволяет использовать их для ввода и вывода объектов пользовательских классов так же удобно, как и для встроенных типов (например, `int`, `string`). Это делает код более читаемым и естественным.

- **Какие потоки используются при работе с файлами? Где они определены?**

  Для работы с файлами используются классы:

  - `ifstream` — для чтения из файла (input file stream).
  - `ofstream` — для записи в файл (output file stream).

- `fstream` — для одновременного чтения и записи (file stream).

Все они определены в заголовочном файле `<fstream>`.

- **С какими атрибутами можно открыть файл?**

Основные атрибуты (режимы) открытия файла:

- `ios::in` — открыть для чтения.
- `ios::out` — открыть для записи (усекает файл, если он существует).
- `ios::app` — открыть для добавления в конец файла (append).
- `ios::ate` — установить указатель в конец файла при открытии (at end).
- `ios::trunc` — усечь файл до нулевой длины (если файл существует).
- `ios::binary` — открыть в бинарном режиме.
- **Как открыть файл по чтению, по записи, по чтению и записи?**
- **Чтение:** `ifstream file("имя_файла");` или `fstream file("имя_файла", ios::in);`
- **Запись:** `ofstream file("имя_файла");` или `fstream file("имя_файла", ios::out);`
- **Чтение и запись:** `fstream file("имя_файла", ios::in | ios::out);`
- **Как прочитать информацию из файла?**
- **Для текстовых файлов:** Используйте оператор `>>` для форматированного чтения или `getline()` для чтения строки.

```
ifstream file("input.txt");
string line;
```

```
while (getline(file, line)) {
    // Обработка строки line
}
```

Для бинарных файлов: Используйте метод `read()`.

```
file.read((char*)&variable, sizeof(variable));
```

Как записать информацию в файл?

- **Для текстовых файлов:** Используйте оператор `<<`.

```
ofstream file("output.txt");
file << "Привет, мир!" << endl;
```

Для бинарных файлов: Используйте метод `write()`.

```
file.write((char*)&variable, sizeof(variable));
```

Как закрыть файл?

Файл можно закрыть с помощью метода `close()`. Однако обычно в этом нет необходимости, так как при разрушении объекта потока (например, при выходе из области видимости) файл закрывается автоматически.

```
file.close();
```

- **Что означает файл последовательного доступа?**

  Это файл, данные из которого читаются (или в который записываются) строго последовательно, от начала к концу. Чтобы просить данные из середины файла, нужно сначала прочитать все предыдущие.

- **Что означает файл прямого доступа?**

  Это файл, который позволяет обращаться к любой его позиции для чтения или записи напрямую, без чтения предыдущих данных. В C++ это достигается с помощью перемещения указателя позиции в файле.

- **Как переместить внутренний указатель в файле?**

Для этого используются методы:

- `seekg(смещение, режим)` — для установки позиции чтения (get).
- `seekp(смещение, режим)` — для установки позиции записи (put).

Режимы:

- `ios::beg` — от начала файла.
- `ios::cur` — от текущей позиции.
- `ios::end` — от конца файла.

Пример:

file.seekg(100, ios::beg); // Переместить указатель чтения на 100 байт от начала