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

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №4
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 механизм наследования и возможности порождения новых типов данных на основе уже существующих классов, изучить определение виртуальных функций и их использование для позднего связывания.

Задание: Для классов предыдущей лабораторной работы реализовать иерархию, изменяя отдельные методы и добавляя члены-данные. В иерархию должно входить 2 производных класса. Один из методов должен быть виртуальным.

Исходные коды модулей проекта:

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

#include <cstdio>
#include <cstring>
#include <fstream>

class Date {
protected:  // Изменено с private на protected для доступа в производных классах
    int day;
    int month;
    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);
    virtual ~Date();  // Виртуальный деструктор для корректного удаления производных классов

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

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

```
    // Виртуальные методы (для позднего связывания)
    virtual const char* toString() const;
    virtual void display() const;  // Виртуальный метод

    // Файловые операции
    virtual bool writeToTextFile(const char* filename) const;
    virtual bool writeToBinaryFile(const char* filename) const;
    virtual bool readFromBinaryFile(const char* filename);
};

#endif

//Date.cpp
#include "Date.h"

// 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 Date 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 Date 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 Date\n");
    weekDay = new char[strlen(other.weekDay) + 1];
    strcpy(weekDay, other.weekDay);
}

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

// Операция сложения дней
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;
            }
```

```
        }
    }
}

// Оператор присваивания
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);
}

// Запись в текстовый файл
bool Date::writeToTextFile(const char* filename) const {
    FILE* file = fopen(filename, "w");
    if(file == nullptr) {
        perror("Ошибка открытия файла для записи текста");
        return false;
    }
    fprintf(file, "%d %d %d %s\n", day, month+1, year, weekDay);
    fclose(file);
    return true;
}

// Запись в бинарный файл
bool Date::writeToBinaryFile(const char* filename) const {
    std::ofstream out(filename, std::ios::binary|std::ios::out);
    if(!out.is_open()) {
        perror("Ошибка открытия файла для бинарной записи");
        return false;
    }
```

```
    out.write((char*)&day, sizeof(day));
    out.write((char*)&month, sizeof(month));
    out.write((char*)&year, sizeof(year));
    size_t len = strlen(weekDay);
    out.write((char*)&len, sizeof(len));
    out.write(weekDay, len);

    out.close();
    return true;
}

// Чтение из бинарного файла
bool Date::readFromBinaryFile(const char* filename) {
    std::ifstream in(filename, std::ios::binary|std::ios::in);
    if(!in.is_open()) {
        perror("Ошибка открытия файла для чтения");
        return false;
    }

    in.read((char*)&day, sizeof(day));
    in.read((char*)&month, sizeof(month));
    in.read((char*)&year, sizeof(year));

    size_t len;
    in.read((char*)&len, sizeof(len));

    delete[] weekDay;
    weekDay = new char[len + 1];
    in.read(weekDay, len);
    weekDay[len] = '\0';

    in.close();
    return true;
}

//DateUtils.h
#ifndef DATE_UTILS_H
#define DATE_UTILS_H

#include "Date.h"
#include "ExtendedDate.h"

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

// Функция для демонстрации полиморфизма
void demonstratePolymorphism(Date* dates[], int count);

#endif

//DateUtils.cpp
#include "DateUtils.h"
#include <cstdio>

// 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()));
}
```

```
// Демонстрация полиморфизма
void demonstratePolymorphism(Date* dates[], int count) {
    printf("\n=== Демонстрация полиморфизма ===\n");
    for(int i = 0; i < count; i++) {
        printf("Объект %d: ", i+1);
        dates[i]->display();  // Виртуальный вызов!
        printf("String представление: %s\n\n", dates[i]->toString());
    }
}

//ExtendedDate.h
#ifndef EXTENDED_DATE_H
#define EXTENDED_DATE_H

#include "Date.h"

// Первый производный класс: Дата с временем
class DateWithTime : public Date {
private:
    int hour;
    int minute;
    int second;

public:
    // Конструкторы
    DateWithTime();
    DateWithTime(int d, int m, int y, const char* wd, int h, int min, int s);
    DateWithTime(const DateWithTime& other);
    ~DateWithTime();

    // Геттеры для времени
    int getHour() const { return hour; }
    int getMinute() const { return minute; }
    int getSecond() const { return second; }

    // Переопределенные виртуальные методы
    virtual void display() const override;
    virtual const char* toString() const override;

    // Собственные методы
    void addSeconds(int seconds);
    void setTime(int h, int m, int s);
};

// Второй производный класс: Дата с событием
class DateWithEvent : public Date {
private:
    char* eventName;
    char* location;

public:
    // Конструкторы
    DateWithEvent();
    DateWithEvent(int d, int m, int y, const char* wd, const char* event, const char* loc);
    DateWithEvent(const DateWithEvent& other);
    ~DateWithEvent();

    // Оператор присваивания
    DateWithEvent& operator=(const DateWithEvent& other);

    // Геттеры
    const char* getEventName() const { return eventName; }
```

```
    const char* getLocation() const { return location; }

    // Переопределенные виртуальные методы
    virtual void display() const override;
    virtual const char* toString() const override;
    virtual bool writeToTextFile(const char* filename) const override;

    // Собственные методы
    void setEvent(const char* event, const char* loc);
};

#endif

//ExtendedDate.cpp
#include "ExtendedDate.h"
#include <cstdio>
#include <cstring>

// ========== DateWithTime реализации ==========

// Конструктор по умолчанию
DateWithTime::DateWithTime() : Date(), hour(0), minute(0), second(0) {
    printf("Конструктор DateWithTime по умолчанию\n");
}

// Конструктор с параметрами
DateWithTime::DateWithTime(int d, int m, int y, const char* wd, int h, int min, int s)
    : Date(d, m, y, wd), hour(h), minute(min), second(s) {
    printf("Конструктор DateWithTime с параметрами\n");
}

// Копирующий конструктор
DateWithTime::DateWithTime(const DateWithTime& other)
    : Date(other), hour(other.hour), minute(other.minute), second(other.second) {
    printf("Копирующий конструктор DateWithTime\n");
}

// Деструктор
DateWithTime::~DateWithTime() {
    printf("Освобождение памяти DateWithTime\n");
}

// Переопределенный метод display
void DateWithTime::display() const {
    printf("Дата и время: %d.%d.%d %02d:%02d:%02d, День недели: %s\n",
           day, month+1, year, hour, minute, second, weekDay);
}

// Переопределенный метод toString
const char* DateWithTime::toString() const {
    static char buffer[20];
    snprintf(buffer, sizeof(buffer), "%02d-%02d-%04d %02d:%02d:%02d",
             day, month+1, year, hour, minute, second);
    return buffer;
}

// Добавление секунд
void DateWithTime::addSeconds(int seconds) {
    second += seconds;
    while(second >= 60) {
        second -= 60;
        minute++;
```

```
        if(minute >= 60) {
            minute = 0;
            hour++;
            if(hour >= 24) {
                hour = 0;
                addDays(1);
            }
        }
    }
}

// Установка времени
void DateWithTime::setTime(int h, int m, int s) {
    hour = h;
    minute = m;
    second = s;
}

// ========== DateWithEvent реализации ==========

// Конструктор по умолчанию
DateWithEvent::DateWithEvent() : Date() {
    printf("Конструктор DateWithEvent по умолчанию\n");
    eventName = new char[10];
    strcpy(eventName, "Meeting");
    location = new char[10];
    strcpy(location, "Office");
}

// Конструктор с параметрами
DateWithEvent::DateWithEvent(int d, int m, int y, const char* wd, const char* event, const char* loc)
    : Date(d, m, y, wd) {
    printf("Конструктор DateWithEvent с параметрами\n");
    eventName = new char[strlen(event) + 1];
    strcpy(eventName, event);
    location = new char[strlen(loc) + 1];
    strcpy(location, loc);
}

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

// Деструктор
DateWithEvent::~DateWithEvent() {
    delete[] eventName;
    delete[] location;
    printf("Освобождение памяти DateWithEvent\n");
}

// Оператор присваивания
DateWithEvent& DateWithEvent::operator=(const DateWithEvent& other) {
    if(this != &other) {
        // Вызываем оператор присваивания базового класса
        Date::operator=(other);
```

```
        delete[] eventName;
        delete[] location;

        eventName = new char[strlen(other.eventName) + 1];
        strcpy(eventName, other.eventName);
        location = new char[strlen(other.location) + 1];
        strcpy(location, other.location);
    }
    return *this;
}

// Переопределенный метод display
void DateWithEvent::display() const {
    printf("Дата события: %d.%d.%d, День недели: %s\n", day, month+1, year, weekDay);
    printf("Событие: %s, Место: %s\n", eventName, location);
}

// Переопределенный метод toString
const char* DateWithEvent::toString() const {
    static char buffer[100];
    snprintf(buffer, sizeof(buffer), "%02d-%02d-%04d | Событие: %s | Место: %s",
             day, month+1, year, eventName, location);
    return buffer;
}

// Переопределенный метод записи в текстовый файл
bool DateWithEvent::writeToTextFile(const char* filename) const {
    FILE* file = fopen(filename, "w");
    if(file == nullptr) {
        perror("Ошибка открытия файла для записи текста");
        return false;
    }
    fprintf(file, "%d %d %d %s %s %s\n", day, month+1, year, weekDay, eventName, location);
    fclose(file);
    return true;
}

// Установка события
void DateWithEvent::setEvent(const char* event, const char* loc) {
    delete[] eventName;
    delete[] location;

    eventName = new char[strlen(event) + 1];
    strcpy(eventName, event);
    location = new char[strlen(loc) + 1];
    strcpy(location, loc);
}

//L4.cpp
#include "Date.h"
#include "ExtendedDate.h"
#include "DateUtils.h"

int main() {
    printf("=== Демонстрация наследования и полиморфизма ===\n\n");

    // 1. Создание объектов разных типов
    printf("1. Создание объектов:\n");
    Date basicDate(15, 5, 2024, "Wednesday");
    DateWithTime dateTime(15, 5, 2024, "Wednesday", 14, 30, 45);
    DateWithEvent dateEvent(15, 5, 2024, "Wednesday", "День рождения", "Кафе");
```

```
    // 2. Демонстрация виртуальных функций через массив указателей на базовый класс
    printf("\n2. Демонстрация полиморфизма:\n");
    Date* dates[3];
    dates[0] = &basicDate;
    dates[1] = &dateTime;
    dates[2] = &dateEvent;

    demonstratePolymorphism(dates, 3);

    // 3. Копирование объектов родственных типов
    printf("3. Копирование объектов:\n");
    DateWithTime copiedDateTime = dateTime;  // Копирующий конструктор
    printf("Скопированный объект DateWithTime: ");
    copiedDateTime.display();

    DateWithEvent copiedDateEvent;
    copiedDateEvent = dateEvent;  // Оператор присваивания
    printf("Присвоенный объект DateWithEvent: ");
    copiedDateEvent.display();

    // 4. Работа с виртуальными функциями файловых операций
    printf("\n4. Файловые операции:\n");
    basicDate.writeToTextFile("basic_date.txt");
    dateTime.writeToTextFile("datetime_date.txt");
    dateEvent.writeToTextFile("event_date.txt");
    printf("Объекты записаны в текстовые файлы\n");

    // 5. Демонстрация специфических методов производных классов
    printf("\n5. Специфические методы производных классов:\n");

    // DateWithTime методы
    dateTime.addSeconds(125);  // Добавляем 2 минуты 5 секунд
    printf("После добавления 125 секунд: ");
    dateTime.display();

    // DateWithEvent методы
    dateEvent.setEvent("Собеседование", "Офис компании");
    printf("После изменения события: ");
    dateEvent.display();

    // 6. Демонстрация работы с базовыми методами через производные классы
    printf("\n6. Наследование базовых методов:\n");
    dateTime.addDays(7);  // Метод базового класса
    printf("DateWithTime после добавления 7 дней: ");
    dateTime.display();

    ++dateEvent;  // Оператор инкремента базового класса
    printf("DateWithEvent после инкремента: ");
    dateEvent.display();

    return 0;
}
```

Выводы:

В ходе выполнения лабораторной работы по программированию было изучено создание иерархий классов с использованием механизма наследования в C++. Основной задачей являлась разработка системы классов, где производные классы расширяли функциональность базового класса через добавление новых свойств и переопределение методов.

На основе ранее созданного класса была построена объектно-ориентированная иерархия, включающая:

- Базовый класс с общими свойствами и поведением
- Два производных класса с специализированной функциональностью
- Виртуальные методы для обеспечения полиморфного поведения

В процессе работы были практически освоены следующие ключевые аспекты:

- Принципы построения иерархий классов через простое и множественное наследование
- Механизмы контроля доступа к членам классов при наследовании
- Особенности вызова конструкторов и деструкторов в иерархии классов
- Реализация виртуальных функций для динамического полиморфизма
- Разрешение проблем множественного наследования через виртуальное наследование

Итогом работы стало создание расширенной системы классов, демонстрирующей различные аспекты наследования. Программа подтвердила корректность реализации полиморфного поведения объектов через механизм виртуальных функций, что обеспечило гибкость и расширяемость разработанной объектной модели.

1. Что такое наследование? Объясните механизм наследования в C++

Наследование - это механизм объектно-ориентированного программирования, который позволяет создавать новый класс на основе существующего класса. Новый класс (производный) наследует свойства и методы существующего класса (базового).

Механизм наследования в C++:

```
class Base {
public:
    int publicVar;
protected:
    int protectedVar;
private:
    int privateVar;
};

class Derived : public Base {
    // Наследует publicVar и protectedVar
    // privateVar недоступен
};
```

2. Какое бывает наследование?

- **Простое наследование** - один производный класс от одного базового
- **Множественное наследование** - один производный класс от нескольких базовых
- **Многоуровневое наследование** - цепочка наследования (A → B → C)

- **Иерархическое наследование** - несколько классов от одного базового
- **Гибридное наследование** - комбинация различных типов

3. Как реализуется простое, множественное наследование?

Простое наследование:

```
class Animal {
   // базовый класс
};

class Dog : public Animal {
   // производный класс
};
```

Множественное наследование:

```
class A {
   // первый базовый класс
};

class B {
   // второй базовый класс
};

class C : public A, public B {
   // наследует от A и B
};
```

4. Какой класс называется базовым, а какой производным?

- **Базовый класс** (родительский) - класс, от которого наследуют

- **Производный класс** (дочерний) - класс, который наследует от базового

5. Как определяется доступ к членам базового класса в производном классе?

Доступ определяется **спецификатором наследования** и **уровнем доступа члена**:

Уровень доступа в базовом классе	public наследование	protected наследование	private наследование
public	public	protected	private
protected	protected	protected	private
private	недоступен	недоступен	недоступен

6. Что такое защищенные члены класса?

Защищенные члены (protected) - это члены класса, которые доступны:

- Внутри самого класса
- В производных классах
- **Не доступны** извне через объекты класса

```
class Base {
protected:
   int protectedVar;  // Доступен в производных классах
};
```

7. Как влияют спецификаторы public, protected, private на статус наследования?

- **public**: "является" отношением, сохраняет уровни доступа
- **protected**: "реализовано посредством", понижает public → protected

- **private**: "реализовано посредством", все члены становятся private

8. Какие члены класса наследуются?

Наследуются **все члены** базового класса, кроме:

- Конструкторов и деструкторов
- Оператора присваивания
- Дружественных функций
- Закрытых (private) членов

9. Вызов конструкторов при наследовании

Порядок вызова конструкторов:

1. Конструкторы базовых классов (в порядке объявления)
2. Конструкторы членов-объектов (в порядке объявления)
3. Конструктор производного класса

```
class Base {
public:
   Base() { cout << "Base constructor\n"; }
};

class Derived : public Base {
public:
   Derived() : Base() {  // Вызов конструктора базового класса
      cout << "Derived constructor\n";
   }
};
```

10. Написание конструкторов в производном классе

```
class Base {
    int x;
public:
    Base(int a) : x(a) {}
};

class Derived : public Base {
    int y;
public:
    // Вызов конструктора базового класса в списке инициализации
    Derived(int a, int b) : Base(a), y(b) {}

    // Если базовый класс имеет конструктор по умолчанию,
    // он вызывается автоматически
    Derived() : y(0) {}  // Base() вызывается автоматически
};
```

11. Что такое абстрактный класс? Могут ли существовать экземпляры абстрактного класса?

Абстрактный класс - класс, содержащий хотя бы одну чистую виртуальную функцию.

```
class AbstractClass {
public:
    virtual void pureVirtual() = 0;  // Чистая виртуальная функция
};

// НЕЛЬЗЯ:
// AbstractClass obj;  // Ошибка компиляции!
```

```
// МОЖНО:
class Concrete : public AbstractClass {
public:
    void pureVirtual() override {
        // Реализация
    }
};

Concrete obj;  // OK
```

Экземпляры абстрактного класса создавать НЕЛЬЗЯ.

12. Какие проблемы возникают при множественном наследовании? Как они разрешаются?

Проблемы:

1. **Неоднозначность** при вызове методов с одинаковыми именами
2. **"Ромбовидная" проблема наследования** - дублирование базового класса

Пример неоднозначности:

```
class A { public: void f() {} };
class B { public: void f() {} };
class C : public A, public B {};

C obj;
// obj.f();  // Ошибка: неоднозначность
obj.A::f();  // OK - указание конкретного класса
obj.B::f();  // OK
```

"Ромбовидная" проблема:

```
class Base { public: int x; };
class A : public Base {};
class B : public Base {};
class C : public A, public B {};

C obj;
// obj.x = 10;  // Ошибка: неоднозначность
obj.A::x = 10;  // OK
obj.B::x = 10;  // OK (но это разные переменные!)
```

Решение - виртуальное наследование:

```
class Base { public: int x; };
class A : virtual public Base {};  // Виртуальное наследование
class B : virtual public Base {};  // Виртуальное наследование
class C : public A, public B {};

C obj;
obj.x = 10;  // OK - теперь x общий для всей иерархии
```

Дополнительные решения:

- Использование пространств имен
- Переименование методов
- Композиция вместо наследования