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

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №5
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 с динамическими списочными структурами данных.

Задание: Реализовать с помощью классов динамическую списочную структуру, содержащую объекты классов, связанных наследованием.

Исходные коды модулей проекта:

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

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

class Date {
protected:
    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();

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

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

    // Виртуальные методы
    virtual const char* toString() const;
    virtual void display() const;
    virtual Date* clone() const;  // Виртуальный метод для клонирования

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

```
    virtual bool readFromBinaryFile(const char* filename);

    // Виртуальный метод для сравнения (для поиска)
    virtual bool isEqual(const Date* other) const;
};

#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) {
    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) {
    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;
}

// 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);
}

// Виртуальный метод для клонирования
Date* Date::clone() const {
    return new Date(*this);
}

// Запись в текстовый файл
bool Date::writeToTextFile(const char* filename) const {
    FILE* file = fopen(filename, "w");
    if(file == nullptr) {
        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()) {
        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()) {
        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;
}

// Сравнение объектов
bool Date::isEqual(const Date* other) const {
    if (!other) return false;
    return day == other->day && month == other->month && year == other->year &&
           strcmp(weekDay, other->weekDay) == 0;
}
```

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;
    virtual Date* clone() const override;
    virtual bool isEqual(const Date* other) 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 Date* clone() const override;
    virtual bool isEqual(const Date* other) 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 ==========

DateWithTime::DateWithTime() : Date(), hour(0), minute(0), second(0) {}

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) {}

DateWithTime::DateWithTime(const DateWithTime& other)
    : Date(other), hour(other.hour), minute(other.minute), second(other.second) {}

DateWithTime::~DateWithTime() {}

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);
}

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;
}

Date* DateWithTime::clone() const {
    return new DateWithTime(*this);
}

bool DateWithTime::isEqual(const Date* other) const {
    const DateWithTime* derived = dynamic_cast<const DateWithTime*>(other);
    if (!derived) return false;
    return Date::isEqual(other) && hour == derived->hour &&
           minute == derived->minute && second == derived->second;
}

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);
            }
        }
    }
}

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

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

DateWithEvent::DateWithEvent() : Date() {
    eventName = new char[10];
    strcpy(eventName, "Meeting");
    location = new char[10];
    strcpy(location, "Office");
}

DateWithEvent::DateWithEvent(int d, int m, int y, const char* wd, const char* event, const char* loc)
    : Date(d, m, y, wd) {
    eventName = new char[strlen(event) + 1];
    strcpy(eventName, event);
    location = new char[strlen(loc) + 1];
    strcpy(location, loc);
}

DateWithEvent::DateWithEvent(const DateWithEvent& other)
```

```
    : Date(other) {
    eventName = new char[strlen(other.eventName) + 1];
    strcpy(eventName, other.eventName);
    location = new char[strlen(other.location) + 1];
    strcpy(location, other.location);
}

DateWithEvent::~DateWithEvent() {
    delete[] eventName;
    delete[] location;
}

DateWithEvent& DateWithEvent::operator=(const DateWithEvent& other) {
    if(this != &other) {
        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;
}

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);
}

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;
}

Date* DateWithEvent::clone() const {
    return new DateWithEvent(*this);
}

bool DateWithEvent::isEqual(const Date* other) const {
    const DateWithEvent* derived = dynamic_cast<const DateWithEvent*>(other);
    if (!derived) return false;
    return Date::isEqual(other) && strcmp(eventName, derived->eventName) == 0 &&
           strcmp(location, derived->location) == 0;
}

bool DateWithEvent::writeToTextFile(const char* filename) const {
    FILE* file = fopen(filename, "w");
    if(file == nullptr) {
        return false;
    }
    fprintf(file, "%d %d %d %s %s %s\n", day, month+1, year, weekDay, eventName, location);
    fclose(file);
    return true;
}

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);
}
```

Stack.h

```
#ifndef STACK_H
#define STACK_H

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

class StackNode {
public:
    Date* data;
    StackNode* next;

    StackNode(Date* date) : data(date), next(nullptr) {}
    ~StackNode() { delete data; }
};

class Stack {
private:
    StackNode* top;
    int size;

    // Вспомогательная функция для копирования
    void copyFrom(const Stack& other);

public:
    // Конструкторы и деструктор
    Stack();
    Stack(const Stack& other);
    ~Stack();

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

    // Основные операции стека
    void push(Date* date);
    Date* pop();
    Date* peek() const;

    // Операции по заданию
    void insert(int position, Date* date);  // Вставка по номеру
    void removeAt(int position);            // Удаление по номеру
    int find(Date* date) const;             // Поиск элемента
    void display() const;                   // Просмотр всей структуры

    // Дополнительные методы
    bool isEmpty() const { return top == nullptr; }
    int getSize() const { return size; }
    void clear();
};

#endif
```

Stack..cpp

```
#include "Stack.h"
```

```
#include <cstdio>

// Конструктор по умолчанию
Stack::Stack() : top(nullptr), size(0) {}

// Копирующий конструктор
Stack::Stack(const Stack& other) : top(nullptr), size(0) {
    copyFrom(other);
}

// Деструктор
Stack::~Stack() {
    clear();
}

// Оператор присваивания
Stack& Stack::operator=(const Stack& other) {
    if (this != &other) {
        clear();
        copyFrom(other);
    }
    return *this;
}

// Вспомогательная функция для копирования
void Stack::copyFrom(const Stack& other) {
    if (other.top == nullptr) return;

    // Создаем временный стек для сохранения порядка
    Stack temp;
    StackNode* current = other.top;

    while (current != nullptr) {
        temp.push(current->data->clone());
        current = current->next;
    }

    // Переносим из временного стека в текущий (восстанавливаем порядок)
    while (!temp.isEmpty()) {
        push(temp.pop());
    }
}

// Добавление элемента в стек
void Stack::push(Date* date) {
    StackNode* newNode = new StackNode(date);
    newNode->next = top;
    top = newNode;
    size++;
}

// Удаление и возврат верхнего элемента
Date* Stack::pop() {
    if (isEmpty()) return nullptr;

    StackNode* temp = top;
    Date* data = temp->data;
    top = top->next;
    temp->data = nullptr;  // Предотвращаем удаление данных в деструкторе StackNode
    delete temp;
    size--;
```

```
    return data;
}

// Просмотр верхнего элемента без удаления
Date* Stack::peek() const {
    if (isEmpty()) return nullptr;
    return top->data;
}

// Вставка по номеру (позиции)
void Stack::insert(int position, Date* date) {
    if (position < 0 || position > size) {
        printf("Ошибка: недопустимая позиция для вставки\n");
        return;
    }

    if (position == 0) {
        push(date);
        return;
    }

    // Находим элемент перед позицией вставки
    StackNode* current = top;
    for (int i = 0; i < position - 1; i++) {
        current = current->next;
    }

    StackNode* newNode = new StackNode(date);
    newNode->next = current->next;
    current->next = newNode;
    size++;
}

// Удаление по номеру (позиции)
void Stack::removeAt(int position) {
    if (position < 0 || position >= size) {
        printf("Ошибка: недопустимая позиция для удаления\n");
        return;
    }

    if (position == 0) {
        pop();
        return;
    }

    // Находим элемент перед удаляемым
    StackNode* current = top;
    for (int i = 0; i < position - 1; i++) {
        current = current->next;
    }

    StackNode* toDelete = current->next;
    current->next = toDelete->next;
    toDelete->data = nullptr;  // Предотвращаем удаление данных в деструкторе
    delete toDelete;
    size--;
}

// Поиск элемента
int Stack::find(Date* date) const {
    if (!date) return -1;
```

```
    StackNode* current = top;
    int position = 0;

    while (current != nullptr) {
        if (current->data->isEqual(date)) {
            return position;
        }
        current = current->next;
        position++;
    }

    return -1;
}

// Просмотр всей структуры
void Stack::display() const {
    if (isEmpty()) {
        printf("Стек пуст\n");
        return;
    }

    printf("Содержимое стека (размер: %d):\n", size);
    StackNode* current = top;
    int position = 0;

    while (current != nullptr) {
        printf("[%d] ", position);
        current->data->display();
        current = current->next;
        position++;
    }
}

// Очистка стека
void Stack::clear() {
    while (!isEmpty()) {
        pop();
    }
}
```

L5.cpp

```
#include "Date.h"
#include "ExtendedDate.h"
#include "Stack.h"
#include <cstdio>

// Функция для демонстрации полиморфизма
void demonstratePolymorphism(Stack& stack) {
    printf("\n=== Демонстрация полиморфизма ===\n");

    StackNode* current = reinterpret_cast<StackNode*>(&stack);

    // Создаем временный указатель для обхода стека
    // В реальной реализации нужно использовать методы стека
    // Для демонстрации создадим временный стек
    Stack tempStack = stack;

    int position = 0;
    while (!tempStack.isEmpty()) {
        Date* date = tempStack.pop();
        printf("Элемент %d (виртуальный вызов display): ", position);
        date->display();  // Полиморфный вызов!
```

```
            printf("Элемент %d (виртуальный вызов toString): %s\n",
                position, date->toString());  // Полиморфный вызов!

            delete date;
            position++;
        }
    }

    int main() {
        printf("=== Демонстрация динамического стека на однонаправленном списке ===\n\n");

        Stack stack;

        // 1. Добавление объектов разных типов в стек
        printf("1. Добавление объектов в стек:\n");
        stack.push(new Date(15, 5, 2024, "Wednesday"));
        stack.push(new DateWithTime(16, 5, 2024, "Thursday", 14, 30, 45));
        stack.push(new DateWithEvent(17, 5, 2024, "Friday", "Экзамен", "Аудитория 101"));
        stack.push(new Date(18, 5, 2024, "Saturday"));

        stack.display();

        // 2. Демонстрация полиморфизма
        demonstratePolymorphism(stack);

        // 3. Вставка по номеру
        printf("\n3. Вставка по номеру 2:\n");
        stack.insert(2, new DateWithTime(19, 5, 2024, "Sunday", 10, 0, 0));
        stack.display();

        // 4. Удаление по номеру
        printf("\n4. Удаление по номеру 1:\n");
        stack.removeAt(1);
        stack.display();

        // 5. Поиск элемента
        printf("\n5. Поиск элемента:\n");
        Date* searchDate = new Date(15, 5, 2024, "Wednesday");
        int foundPosition = stack.find(searchDate);
        if (foundPosition != -1) {
            printf("Элемент найден на позиции: %d\n", foundPosition);
        } else {
            printf("Элемент не найден\n");
        }
        delete searchDate;

        // 6. Работа с верхним элементом
        printf("\n6. Работа с верхним элементом:\n");
        Date* topElement = stack.peek();
        if (topElement) {
            printf("Верхний элемент: ");
            topElement->display();
        }

        // 7. Удаление из стека
        printf("\n7. Удаление из стека (pop):\n");
        Date* popped = stack.pop();
        if (popped) {
            printf("Удаленный элемент: ");
            popped->display();
            delete popped;
```

```
    }
    stack.display();

    // 8. Копирование стека
    printf("\n8. Копирование стека:\n");
    Stack copiedStack = stack;
    printf("Оригинальный стек: ");
    stack.display();
    printf("Скопированный стек: ");
    copiedStack.display();

    // 9. Очистка стека
    printf("\n9. Очистка стека:\n");
    stack.clear();
    printf("После очистки: ");
    stack.display();

    printf("\n=== Демонстрация завершена ===\n");

    return 0;
}
```

Выводы:

В результате выполнения лабораторной работы была успешно реализована динамическая списочная структура, использующая механизмы объектно-ориентированного программирования. На практике были применены принципы наследования для построения иерархии классов, объекты которых хранятся в списке. Работа со структурой потребовала глубокого понимания динамического выделения памяти, использования указателей и организации связей между элементами. Разработанный программный комплекс наглядно демонстрирует эффективность комбинации динамических структур данных и наследования для решения задач управления разнотипными объектами в C++.

1. Что такое виртуальные функции и как они определяются в базовом и производном классах?

Виртуальные функции — это функции-члены класса, объявленные с ключевым словом `virtual`. Они позволяют реализовать **полиморфизм времени выполнения**: когда программа выбирает, какую функцию выполнить, основываясь на типе объекта, а не на типе указателя/ссылки.

- **В базовом классе** виртуальная функция объявляется с ключевым словом `virtual`:

```
class Base {
 public:
    virtual void show() {
        cout << "Base class show()" << endl;
    }
};
```

В производном классе функция переопределяется (обычно с ключевым словом `override` для ясности):

```
class Derived : public Base {
public:
    void show() override {  // override не обязательно, но рекомендуется
        cout << "Derived class show()" << endl;
    }
};
```

Пример использования:

```
Base* obj = new Derived();
obj->show();  // Вызовется Derived::show() благодаря виртуальности
```

2. Что такое чисто виртуальная функция? Как называется класс, содержащий чисто виртуальную функцию?

Чисто виртуальная функция — это виртуальная функция, которая **не имеет реализации** в базовом классе. Она объявляется с помощью `= 0`. Например:

```
virtual void draw() = 0;
```

Класс, содержащий хотя бы одну чисто виртуальную функцию, называется **абстрактным классом**. Нельзя создать объект абстрактного класса. Его производные классы **должны переопределить** все чисто виртуальные функции, иначе они тоже станут абстрактными.

3. Что такое полиморфизм?

Полиморфизм — это свойство системы, позволяющее использовать объекты разных классов через единый интерфейс. В C++ полиморфизм бывает двух типов:

1. **Статический** (во время компиляции) — перегрузка функций и шаблоны.
2. **Динамический** (во время выполнения) — достигается через виртуальные функции и наследование.

Пример динамического полиморфизма:

```
Animal* animal = new Dog();
animal->makeSound();  // Вызовется Dog::makeSound(), даже если тип указателя — Animal
```

---

4. Каков механизм реализации полиморфизма?

Механизм реализации полиморфизма в C++ основан на:

1. **Виртуальных функциях** и **таблицах виртуальных функций (vtable)**.

2. **Указателе на vtable (vptr)**, который неявно добавляется в каждый объект класса с виртуальными функциями.

Как это работает:

- Каждый класс с виртуальными функциями имеет свою `vtable` — массив указателей на виртуальные функции.
- Каждый объект такого класса содержит скрытый указатель `vptr` на `vtable` своего класса.
- При вызове виртуальной функции через указатель/ссылку на базовый класс, программа:
    1. Следует по `vptr` объекта к `vtable`.
    2. Находит нужную функцию в `vtable`.
    3. Вызывает её.

Пример:

```
Base* obj = new Derived();
obj->virtualFunction();  // Через vptr и vtable вызывается Derived::virtualFunction()
```

Это позволяет выбирать версию функции в зависимости от реального типа объекта, даже если он доступен через указатель базового класса.