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

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №6
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 обработки исключительных ситуаций в объектно-ориентированных программах.

Задание: Добавить в классы и демонстрационную программу обработку исключений при возникновении ошибок: недостатка памяти, выхода за пределы диапазона допустимых значений и т.д. Дополнить демонстрационную программу так, чтобы она демонстрировала обработку исключений.

Исходные коды модулей проекта:

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

#include <cstdio>
#include <cstring>
#include <fstream>
#include "Exceptions.h"

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

    // Вспомогательные методы для проверки
    bool isValidDate(int d, int m, int y) const;
    bool isValidWeekDay(const char* wd) const;

public:
    // 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();

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

    // Сеттеры с проверкой
    void setDay(int d);
    void setMonth(int m);
    void setYear(int y);
    void setWeekDay(const char* wd);
```

```
    // 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;

    // 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
    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};

// Проверка корректности даты
bool Date::isValidDate(int d, int m, int y) const {
    if (y < 1900 || y > 2100) return false;
    if (m < 0 || m > 11) return false;
    if (d < 1 || d > daysInMonth[m]) return false;
    return true;
}

// Проверка корректности дня недели
bool Date::isValidWeekDay(const char* wd) const {
    if (!wd || strlen(wd) == 0 || strlen(wd) > 20) return false;
    return true;
}

// Конструктор по умолчанию
Date::Date() : day(1), month(1), year(2000) {
    weekDay = new (std::nothrow) char[10];
    if (!weekDay) {
        throw MemoryException("Не удалось выделить память для weekDay в конструкторе по
умолчанию");
    }
    strcpy(weekDay, "Monday");
}

// Конструктор с параметрами
Date::Date(int d, int m, int y, const char* wd) : day(d), month(m), year(y) {
    if (!isValidDate(d, m, y)) {
        throw InvalidDateException("Некорректная дата в конструкторе с параметрами");
    }

    if (!isValidWeekDay(wd)) {
```

```
        throw InvalidDateException("Некорректный день недели в конструкторе с параметрами");
    }

    weekDay = new (std::nothrow) char[strlen(wd) + 1];
    if (!weekDay) {
        throw MemoryException("Не удалось выделить память для weekDay в конструкторе с
параметрами");
    }
    strcpy(weekDay, wd);
}

// Копирующий конструктор
Date::Date(const Date& other) : day(other.day), month(other.month), year(other.year) {
    weekDay = new (std::nothrow) char[strlen(other.weekDay) + 1];
    if (!weekDay) {
        throw MemoryException("Не удалось выделить память для weekDay в копирующем
конструкторе");
    }
    strcpy(weekDay, other.weekDay);
}

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

// Сеттеры с проверкой
void Date::setDay(int d) {
    if (!isValidDate(d, month, year)) {
        throw InvalidDateException("Некорректный день");
    }
    day = d;
}

void Date::setMonth(int m) {
    if (!isValidDate(day, m, year)) {
        throw InvalidDateException("Некорректный месяц");
    }
    month = m;
}

void Date::setYear(int y) {
    if (!isValidDate(day, month, y)) {
        throw InvalidDateException("Некорректный год");
    }
    year = y;
}

void Date::setWeekDay(const char* wd) {
    if (!isValidWeekDay(wd)) {
        throw InvalidDateException("Некорректный день недели");
    }

    delete[] weekDay;
    weekDay = new (std::nothrow) char[strlen(wd) + 1];
    if (!weekDay) {
        throw MemoryException("Не удалось выделить память для weekDay в setWeekDay");
    }
    strcpy(weekDay, wd);
}

// Операция сложения дней
```

```
void Date::addDays(int numDays) {
    if (numDays < 0) {
        throw InvalidDateException("Количество дней не может быть отрицательным");
    }

    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;

        char* newWeekDay = new (std::nothrow) char[strlen(other.weekDay) + 1];
        if (!newWeekDay) {
            throw MemoryException("Не удалось выделить память в операторе присваивания");
        }

        delete[] weekDay;
        weekDay = newWeekDay;
        strcpy(weekDay, other.weekDay);
    }
    return *this;
}

// Остальные методы остаются без изменений, но добавляем проверки...
// 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) {
        throw FileException("Не удалось открыть файл для записи");
    }
    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()) {
        throw FileException("Не удалось открыть файл для бинарной записи");
    }

    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()) {
        throw FileException("Не удалось открыть файл для чтения");
    }

    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));

    char* newWeekDay = new (std::nothrow) char[len + 1];
    if (!newWeekDay) {
        in.close();
        throw MemoryException("Не удалось выделить память при чтении из файла");
    }

    delete[] weekDay;
```

```
    weekDay = newWeekDay;
    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;
}
```

ExtendenDate.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);
}
```

Exeptions.h

```
#ifndef EXCEPTIONS_H
#define EXCEPTIONS_H

#include <cstdio>
#include <cstring>

// Базовый класс для исключений
class Exception {
protected:
    char* message;
public:
    Exception(const char* msg) {
        message = new char[strlen(msg) + 1];
```

```
        strcpy(message, msg);
    }

    virtual const char* what() const {
        return message;
    }

    virtual ~Exception() {
        delete[] message;
    }
};

// Исключение для ошибок памяти
class MemoryException : public Exception {
public:
    MemoryException(const char* msg = "Ошибка выделения памяти") : Exception(msg) {}
};

// Исключение для выхода за границы
class OutOfRangeException : public Exception {
public:
    OutOfRangeException(const char* msg = "Выход за границы диапазона") : Exception(msg) {}
};

// Исключение для неверных дат
class InvalidDateException : public Exception {
public:
    InvalidDateException(const char* msg = "Неверная дата") : Exception(msg) {}
};

// Исключение для ошибок файлов
class FileException : public Exception {
public:
    FileException(const char* msg = "Ошибка работы с файлом") : Exception(msg) {}
};

// Исключение для пустого стека
class EmptyStackException : public Exception {
public:
    EmptyStackException(const char* msg = "Стек пуст") : Exception(msg) {}
};

#endif
```

Stack.h

```
#ifndef STACK_H
#define STACK_H

#include "Date.h"
#include "ExtendedDate.h"
#include "Exceptions.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;

    void copyFrom(const Stack& other);

public:
    Stack();
    Stack(const Stack& other);
    ~Stack();

    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);
    void removeAt(int position);
    int find(Date* date) const;
    void display() const;

    // 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) {
        Date* cloned = current->data->clone();
        if (!cloned) {
            throw MemoryException("Не удалось клонировать объект при копировании стека");
        }
        temp.push(cloned);
        current = current->next;
    }

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

// Добавление элемента в стек
void Stack::push(Date* date) {
    if (!date) {
        throw InvalidDateException("Попытка добавить нулевой указатель в стек");
    }

    StackNode* newNode = new (std::nothrow) StackNode(date);
    if (!newNode) {
        throw MemoryException("Не удалось выделить память для нового узла стека");
    }

    newNode->next = top;
    top = newNode;
    size++;
}

// Удаление и возврат верхнего элемента
Date* Stack::pop() {
    if (isEmpty()) {
        throw EmptyStackException("Попытка извлечения из пустого стека");
    }

    StackNode* temp = top;
    Date* data = temp->data;
    top = top->next;
    temp->data = nullptr;
    delete temp;
    size--;

    return data;
}

// Просмотр верхнего элемента без удаления
Date* Stack::peek() const {
    if (isEmpty()) {
        throw EmptyStackException("Попытка просмотра пустого стека");
    }
    return top->data;
}

// Вставка по номеру
void Stack::insert(int position, Date* date) {
    if (!date) {
```

```
        throw InvalidDateException("Попытка вставки нулевого указателя");
    }

    if (position < 0 || position > size) {
        throw OutOfRangeException("Неверная позиция для вставки");
    }

    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 (std::nothrow) StackNode(date);
    if (!newNode) {
        throw MemoryException("Не удалось выделить память для нового узла при вставке");
    }

    newNode->next = current->next;
    current->next = newNode;
    size++;
}

// Удаление по номеру
void Stack::removeAt(int position) {
    if (position < 0 || position >= size) {
        throw OutOfRangeException("Неверная позиция для удаления");
    }

    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;
    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()) {
        Date* data = pop();
        delete data;
    }
}
```

L6.cpp

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

// Функция для демонстрации различных исключений
void demonstrateExceptions() {
    printf("\n=== Демонстрация обработки исключений ===\n");

    // 1. Исключение при создании неверной даты
    printf("1. Попытка создания неверной даты:\n");
    try {
        Date invalidDate(32, 13, 2024, "InvalidDay");
        printf("Ошибка: исключение не было брошено!\n");
    } catch (const InvalidDateException& e) {
        printf("Поймано исключение: %s\n", e.what());
    } catch (const Exception& e) {
        printf("Поймано общее исключение: %s\n", e.what());
    }

    // 2. Исключение при работе с пустым стеком
    printf("\n2. Попытка извлечения из пустого стека:\n");
    try {
```

```
        Stack emptyStack;
        emptyStack.pop();
        printf("Ошибка: исключение не было брошено!\n");
    } catch (const EmptyStackException& e) {
        printf("Поймано исключение: %s\n", e.what());
    } catch (const Exception& e) {
        printf("Поймано общее исключение: %s\n", e.what());
    }

    // 3. Исключение при неверной позиции
    printf("\n3. Попытка вставки по неверной позиции:\n");
    try {
        Stack stack;
        stack.insert(5, new Date());  // Неверная позиция
        printf("Ошибка: исключение не было брошено!\n");
    } catch (const OutOfRangeException& e) {
        printf("Поймано исключение: %s\n", e.what());
    } catch (const Exception& e) {
        printf("Поймано общее исключение: %s\n", e.what());
    }

    // 4. Исключение при работе с файлом
    printf("\n4. Попытка записи в неверный файл:\n");
    try {
        Date date;
        date.writeToTextFile("/invalid/path/file.txt");
        printf("Ошибка: исключение не было брошено!\n");
    } catch (const FileException& e) {
        printf("Поймано исключение: %s\n", e.what());
    } catch (const Exception& e) {
        printf("Поймано общее исключение: %s\n", e.what());
    }

    // 5. Исключение при установке неверного дня
    printf("\n5. Попытка установки неверного дня:\n");
    try {
        Date date;
        date.setDay(32);  // Неверный день
        printf("Ошибка: исключение не было брошено!\n");
    } catch (const InvalidDateException& e) {
        printf("Поймано исключение: %s\n", e.what());
    } catch (const Exception& e) {
        printf("Поймано общее исключение: %s\n", e.what());
    }
}

// Функция для демонстрации нормальной работы с обработкой исключений
void demonstrateNormalWork() {
    printf("\n=== Демонстрация нормальной работы с обработкой ошибок ===\n");

    Stack stack;

    try {
        // Добавляем корректные данные
        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", "Собрание", "Офис"));

        printf("Стек успешно создан:\n");
        stack.display();

        // Корректные операции
```

```
        printf("\nКорректное извлечение:\n");
        Date* popped = stack.pop();
        if (popped) {
            printf("Извлеченный элемент: ");
            popped->display();
            delete popped;
        }

        printf("\nСтек после извлечения:\n");
        stack.display();

        // Корректная вставка
        printf("\nКорректная вставка по позиции 1:\n");
        stack.insert(1, new Date(18, 5, 2024, "Saturday"));
        stack.display();

    } catch (const Exception& e) {
        printf("Произошла ошибка: %s\n", e.what());
    }
}

// Функция для демонстрации полиморфизма исключений
void demonstrateExceptionPolymorphism() {
    printf("\n=== Демонстрация полиморфизма исключений ===\n");

    Exception* exceptions[4];

    exceptions[0] = new MemoryException("Особая ошибка памяти");
    exceptions[1] = new OutOfRangeException("Особая ошибка диапазона");
    exceptions[2] = new InvalidDateException("Особая ошибка даты");
    exceptions[3] = new FileException("Особая ошибка файла");

    for (int i = 0; i < 4; i++) {
        try {
            printf("Бросаем исключение %d: ", i + 1);
            throw exceptions[i];
        } catch (const MemoryException& e) {
            printf("Обработана MemoryException: %s\n", e.what());
        } catch (const OutOfRangeException& e) {
            printf("Обработана OutOfRangeException: %s\n", e.what());
        } catch (const InvalidDateException& e) {
            printf("Обработана InvalidDateException: %s\n", e.what());
        } catch (const FileException& e) {
            printf("Обработана FileException: %s\n", e.what());
        } catch (const Exception& e) {
            printf("Обработана общая Exception: %s\n", e.what());
        }
    }

    // Освобождаем память
    for (int i = 0; i < 4; i++) {
        delete exceptions[i];
    }
}

int main() {
    printf("=== Демонстрация обработки исключительных ситуаций ===\n");

    try {
        // Демонстрация различных типов исключений
        demonstrateExceptions();
```

```
        // Демонстрация нормальной работы
        demonstrateNormalWork();

        // Демонстрация полиморфизма исключений
        demonstrateExceptionPolymorphism();

        // Дополнительная демонстрация: обработка в цепочке
        printf("\n=== Демонстрация вложенной обработки ===\n");
        try {
            Stack stack;
            // Создаем ситуацию, которая может вызвать несколько исключений
            stack.push(new Date(31, 2, 2024, "Wednesday"));  // Неверная дата
        } catch (const InvalidDateException& e) {
            printf("Внутреннее исключение: %s\n", e.what());
            // Бросаем новое исключение
            throw FileException("Не удалось добавить элемент в стек из-за неверной даты");
        } catch (const Exception& e) {
            printf("Внешнее исключение: %s\n", e.what());
        }

    } catch (const Exception& e) {
        printf("Исключение перехвачено в main: %s\n", e.what());
    }

    printf("\n=== Программа завершена успешно ===\n");
    return 0;
}
```

Выводы:

В ходе выполнения лабораторной работы был изучен механизм обработки исключительных ситуаций в объектно-ориентированном программировании. На практике реализована система обработки ошибок для ранее разработанных классов, включая контроль критических ситуаций, таких как недостаток памяти, выход за границы допустимых диапазонов и другие потенциально аварийные случаи.

В процессе работы были освоены основные принципы обработки исключений:

- организация блоков try-catch для перехвата и обработки исключений;
- использование стандартных исключений библиотеки STL;
- создание специализированных типов исключений для конкретных ошибочных сценариев.

Демонстрационная программа была дополнена сценариями, которые наглядно показывают работу механизма исключений в различных условиях. Реализованная система обработки ошибок повысила надежность программы, обеспечивая корректное выполнение даже при возникновении исключительных ситуаций и демонстрируя профессиональный подход к разработке устойчивого программного обеспечения.

1. Что такое исключительная ситуация?

Исключительная ситуация (исключение) — это непредвиденная ошибка или исключительное условие, которое возникает во время выполнения программы и нарушает нормальный поток управления. Примеры: деление на ноль, обращение к несуществующей памяти, открытие несуществующего файла.

2. Каков механизм обработки исключения?

Механизм обработки исключений в C++ состоит из трех основных компонентов:

1. **throw** — генерация исключения
2. **try** — блок кода, где может возникнуть исключение
3. **catch** — блок обработки исключения

Механизм:

- Код, который может вызвать ошибку, помещается в блок `try`
- При возникновении исключения используется `throw` для "бросания" исключения
- Блок `catch` перехватывает и обрабатывает исключение

3. Какой тип может быть у ошибки?

Тип исключения может быть **любым**:

- **Базовые типы**: `int`, `char`, `const char*` и др.
- **Объекты классов**: стандартные (`std::exception`, `std::runtime_error`) или пользовательские
- **Указатели**

Рекомендуется использовать классы, производные от `std::exception`.

4. Что будет происходить, если класс не содержит обработчик исключения?

Если в текущем блоке `try` нет подходящего обработчика `catch`:

1. Происходит **поиск обработчика в вызывающих функциях** (раскрутка стека)
2. Если обработчик не найден во всей цепочке вызовов, вызывается функция `std::terminate()`
3. **Программа аварийно завершается**

5. Для чего нужны блоки try? catch? Оператор throw?

- **try** — определяет блок кода, в котором могут возникать исключения
- **catch** — обрабатывает исключения, возникшие в блоке `try`
- **throw** — генерирует исключение

Пример:

```
try {
    if (error) {
        throw std::runtime_error("Произошла ошибка!");
    }
}
catch (const std::exception& e) {
    cout << "Ошибка: " << e.what() << endl;
}
```

6. Сколько блоков catch может быть после блока try? Имеет ли значение их порядок?

После блока `try` может быть **неограниченное количество блоков `catch`**.

Порядок имеет критическое значение:

- Обработчики проверяются **сверху вниз**
- Первый подходящий обработчик выполняется, остальные игнорируются
- **Специфичные исключения должны обрабатываться раньше общих**

Пример:

```
try {
    // код
}
catch (const MyException& e) {  // Специфичное исключение
    // обработка
}
catch (const std::exception& e) {  // Более общее исключение
    // обработка
}
catch (...) {  // Самый общий обработчик - должен быть последним
    // обработка любых исключений
}
```

7. Что означает «генерация повторного исключения»?

Повторная генерация исключения — это когда обработчик `catch` перехватывает исключение, но затем снова генерирует его (с помощью `throw` без аргументов) для обработки на более высоком уровне.

Пример:

```
try {
    try {
        throw std::runtime_error("Ошибка");
    }
    catch (const std::exception& e) {
        cout << "Частичная обработка" << endl;
        throw;  // Повторная генерация того же исключения
    }
}
catch (const std::exception& e) {
    cout << "Полная обработка: " << e.what() << endl;
}
```

8. Что означает «раскрутка стека»? Когда она возникает?

Раскрутка стека (stack unwinding) — это процесс автоматического уничтожения всех локальных объектов в цепочке вызовов функций при возникновении исключения.

Когда возникает:

- При выбросе исключения
- Когда система ищет подходящий обработчик `catch`

Что происходит:

1. Выполнение текущей функции прекращается

2. Локальные объекты уничтожаются (вызываются их деструкторы)

3. Процесс повторяется для вызывающих функций, пока не найдется обработчик исключения

Важно: Деструкторы всех локальных объектов вызываются автоматически, что обеспечивает корректное освобождение ресурсов (принцип RAII).