Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
Пермский национальный исследовательский политехнический университет
Электротехнический факультет
Кафедра информационных технологий и автоматизированных систем

Лабораторная работа №9

" Обработка исключительных ситуаций "

Вариант: 12

Выполнил студент ИВТ-24-2б:
Шишкин Максим Григорьевич

_____
(дата, подпись)

Проверил доцент кафедры ИТАС:

Полякова Ольга Андреевна

_____
(дата, подпись)

Пермь 2025

Содержание

1 Постановка задачи

1. Реализовать класс, перегрузить для него операции, указанные в варианте.

2. Определить исключительные ситуации.

3. Предусмотреть генерацию исключительных ситуаций.

Вариант: Класс- контейнер СПИСОК с ключевыми значениями типа int. Реализовать операции:

[] – доступа по индексу;

() – определение размера вектора;

+ число – добавляет константу ко всем элементам вектора;

++ - добавление элемента в конец списка.

Вариант реализации: 1 и 3

2 Код на C++

lab9_main.cpp | Vector1.h | Vector1.cpp | Vector3.cpp | Vector3.h | error.h

Lab9_Vector

```cpp
#include <iostream>
#include <windows.h>
#include <locale>
#include "Vector1.h"
#include "Vector3.h"
#include "error.h"
using namespace std;

int main() {
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    setlocale(LC_ALL, "Russian");
    cout << "=== Вариант 1: Стандартные исключения ===\n";
    Vector1 v1;
    v1.push_back(1);
    v1.push_back(2);
    v1.print();
    cout << "Размер: " << v1() << endl;
    try {
        cout << "v[1] = " << v1[1] << endl;
        Vector1 v2 = v1 + 3;
        v2.print();
        cout << v1[10] << endl;
    } catch (const exception& e) {
        cout << "Стандартная ошибка: " << e.what() << endl;
    }

    cout << "\n=== Вариант 3: Пользовательские исключения ===\n";
    Vector3 v3;
    v3.push_back(10);
    v3.push_back(20);
    v3.print();
    cout << "Размер: " << v3() << endl;
    try {
        cout << "v[1] = " << v3[1] << endl;
        Vector3 v4 = v3 + 5;
        v4.print();
        cout << v3[100] << endl;
    } catch (const VectorError& e) {
        cout << "Пользовательская ошибка: " << e.what() << endl;
    }

    return 0;
}
```

lab9_main.cpp | Vector1.h | Vector1.cpp | Vector3.cpp | Vector3.h | error.h

Lab9_Vector

```cpp
#pragma once
#include <vector>
#include <iostream>
#include <string>

using namespace std;

class Vector1 {
private:
    vector<int> data;
public:
    void push_back(int value);
    int& operator[](int index);
    int operator()() const;
    Vector1 operator+(int value) const;
    void print() const;
};
```

lab9_main.cpp | Vector1.h | **Vector1.cpp** | Vector3.cpp | Vector3.h | error.h

Lab9_Vector — Vector1

```cpp
#include "Vector1.h"
#include <stdexcept>
#include <clocale>

void Vector1::push_back(int value) {
    data.push_back(value);
}

int& Vector1::operator[](int index) {
    if (index < 0 || index >= data.size()) {
        throw out_of_range("Индекс вне диапазона: " + to_string(index));
    }
    return data[index];
}

int Vector1::operator()() const {
    return static_cast<int>(data.size());
}

Vector1 Vector1::operator+(int value) const {
    Vector1 result = *this;
    for (auto& el : result.data) {
        el += value;
    }
    return result;
}

void Vector1::print() const {
    cout << "[ ";
    for (int val : data) {
        cout << val << " ";
    }
    cout << "]\n";
}
```

lab9_main.cpp | Vector1.h | Vector1.cpp | **Vector3.cpp** | Vector3.h | error.h

Lab9_Vector — (Глобальная о

```cpp
#include "Vector3.h"
#include <clocale>

void Vector3::push_back(int value) {
    data.push_back(value);
}

int& Vector3::operator[](int index) {
    if (index < 0 || index >= data.size()) {
        throw IndexOutOfRange(index);
    }
    return data[index];
}

int Vector3::operator()() const {
    return data.size();
}

Vector3 Vector3::operator+(int value) const {
    Vector3 result = *this;
    for (auto& el : result.data) {
        el += value;
    }
    return result;
}

void Vector3::print() const {
    cout << "[ ";
    for (int val : data) {
        cout << val << " ";
    }
    cout << "]\n";
}
```

lab9_main.cpp | Vector1.h | Vector1.cpp | Vector3.cpp | Vector3.h | error.h

Lab9_Vector

```cpp
#pragma once
#include <vector>
#include <iostream>
#include "error.h"
using namespace std;

class Vector3 {
private:
    vector<int> data;
public:
    void push_back(int value);
    int& operator[](int index);
    int operator()() const;
    Vector3 operator+(int value) const;
    void print() const;
};
```

lab9_main.cpp | Vector1.h | Vector1.cpp | Vector3.cpp | Vector3.h | error.h

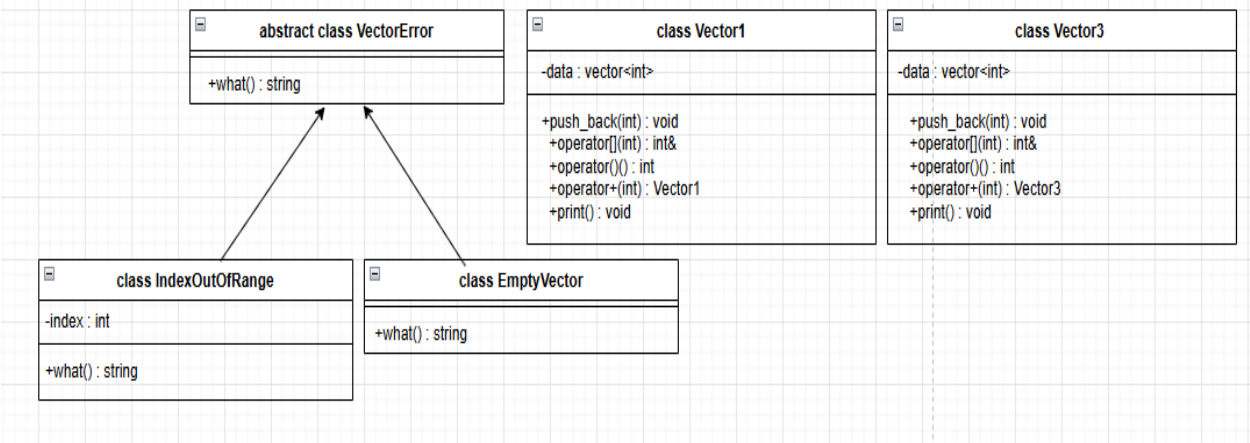
Lab9_Vector | VectorError

```cpp
#pragma once
#include <string>
#include <exception>
using namespace std;

class VectorError : public exception {
public:
    virtual const char* what() const noexcept = 0;
};

class IndexOutOfRange : public VectorError {
    int index;
    string msg;
public:
    IndexOutOfRange(int i) : index(i) {
        msg = "Индекс вне диапазона: " + to_string(i);
    }
    const char* what() const noexcept override {
        return msg.c_str();
    }
};

class EmptyVector : public VectorError {
public:
    const char* what() const noexcept override {
        return "Вектор пустой";
    }
};
```

3 Результаты работы

```
=== Вариант 1: Стандартные исключения ===
[ 1 2 ]
Размер: 2
v[1] = 2
[ 4 5 ]
Стандартная ошибка: Индекс вне диапазона: 10

=== Вариант 3: Пользовательские исключения ===
[ 10 20 ]
Размер: 2
v[1] = 20
[ 15 25 ]
Пользовательская ошибка: Индекс вне диапазона: 100
```

4 UML-диаграмма классов



5 Ответы на контрольные вопросы

1 Исключение в C++ — это объект, сигнализирующий об ошибочной ситуации во время выполнения программы. Оно передаётся в обработчик исключений через механизм throw.

2 Исключения позволяют разделить процесс на:

- основной поток исполнения
- обработку ошибок Достоинства: повышается читаемость, надёжность и удобство отладки, код обработки ошибок не смешивается с основным.

3 Для генерации исключения используется оператор throw.

4 Контролируемый блок — это блок try { … }, в котором может возникнуть исключение. Он нужен для выявления и перехвата исключений.

5 Секция-ловушка (catch(…)) — это блок, следующий за try, предназначен для перехвата исключений. Нужна для обработки ошибок и предотвращения аварийного завершения программы.

6 Формы спецификации:

- catch (int e) — ловит исключение определённого типа
- catch (...) — универсальный обработчик Используются в зависимости от того, известно ли тип исключения заранее или нет.

7 Можно использовать стандартный класс std::exception в качестве базового.
8 Создание собственной иерархии:

```
class MyException : public std::exception {
public:
  const char* what() const noexcept override {
    return "My custom exception";
  }
};
```

9 При void f1() throw(int, double); функция f1() может генерировать только исключения типов int и double
10 При void f1() throw(); функция не может генерировать никакие исключения (устаревшая спецификация C++98, в C++11 заменена на noexcept)
11 Исключения могут генерироваться в любой части программы, где обнаружена ошибка: в функциях, конструкторах, методах и т.д.
12 Пример функции для площади треугольника по формуле Герона в 4-х вариантах:
Вариант без спецификации:

```
double TriangleArea(double a, double b, double c) { double p = (a + b + c) / 2; if (a + b <= c || a + c <= b || b + c <= a) throw "Invalid triangle sides"; return sqrt(p * (p - a) * (p - b) * (p - c)); }
```

Вариант со спецификацией `throw()`:

double TriangleArea2(double a, double b, double c) throw() { double p = (a + b + c) / 2; if (a + b <= c || a + c <= b || b + c <= a) return 0; return sqrt(p * (p - a) * (p - b) * (p - c)); }

С конкретной спецификацией `throw(std::invalid_argument)`:

#include double TriangleArea3(double a, double b, double c) throw(std::invalid_argument) { if (a + b <= c || a + c <= b || b + c <= a) throw std::invalid_argument("Стороны не образуют треугольник"); double p = (a + b + c) / 2; return sqrt(p * (p - a) * (p - b) * (p - c)); }

С собственной реализацией исключения:

class TriangleException : public std::exception { public: const char* what() const noexcept override { return "Неверные стороны треугольника"; } };

double TriangleArea4(double a, double b, double c) { if (a + b <= c || a + c <= b || b + c <= a) throw TriangleException(); double p = (a + b + c) / 2; return sqrt(p * (p - a) * (p - b) * (p - c)); }

6 Ссылка на github

ссылка на github - https://github.com/MAKSPOWERO/mas1