Министерство высшего образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет» (ПНИПУ)**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

ОТЧЁТ

по лабораторной работе №9

Тема: «Обработка исключительных ситуаций»

Выполнил

Студент группы РИС-22-2б

Арапов М.В.

Проверил доц. Кафедры ИТАС

Полякова О. А.

Пермь 2023

# Постановка задачи

Класс- контейнер ВЕКТОР с элементами типа int.

Реализовать операции:

[]- доступа по индексу;

++ добавляет элемент в вектор (постфиксная операция добавляет элемент в конец, префиксная в начало)

**Контрольные вопросы**

1. Что представляет собой исключение в С++?

Исключение в C++ представляет собой событие, которое возникает во время выполнения программы и приводит к ее некорректному поведению.

2. На какие части исключения позволяют разделить вычислительный процесс? Достоинства такого подхода?

Исключение позволяет разделить вычислительный процесс на две части: основную и обработку ошибок. Это позволяет улучшить структуру программы, сделать ее более надежной и легче поддерживаемой.

3. Какой оператор используется для генерации исключительной ситуации?

Для генерации исключительной ситуации используется оператор throw.

4. Что представляет собой контролируемый блок? Для чего он нужен?

Контролируемый блок представляет собой участок кода, который может генерировать исключения. Он нужен для того, чтобы обеспечить правильную обработку исключений в программе.

5. Что представляет собой секция-ловушка? Для чего она нужна?

Секция-ловушка представляет собой участок кода, который обрабатывает исключения. Она нужна для того, чтобы обеспечить корректную обработку исключений и предотвратить прерывание работы программы.

6. Какие формы может иметь спецификация исключения в секции ловушке? В каких ситуациях используются эти формы?

Спецификация исключения может иметь форму throw(), throw(type1), throw(type1, type2), где type1 и type2 - типы исключений. Форма throw() означает, что функция не генерирует исключений, форма throw(type1) - что функция может генерировать исключение типа type1, форма throw(type1, type2) - что функция может генерировать исключения типов type1 и type2. Эти формы используются для указания того, какие исключения может генерировать функция.

7. Какой стандартный класс можно использовать для создания собственной иерархии исключений?

Стандартный класс exception можно использовать для создания собственной иерархии исключений.

8. Каким образом можно создать собственную иерархию исключений?

Для создания собственной иерархии исключений необходимо создать новый класс, который будет наследоваться от класса exception.

9. Если спецификация исключений имеет вид: void f1()throw(int,double); то какие исключения может прождать функция f1()?

Функция f1() может прождать исключения типа int и double.

10. Если спецификация исключений имеет вид: void fl()throw(); то какие исключения может прождать функция f1()?

Функция f1() не может прождать никаких исключений.

11. В какой части программы может генерироваться исключение?

Исключение может генерироваться в любой части программы, включая функции, методы классов, конструкторы и деструкторы.

12. Написать функцию, которая вычисляет площадь треугольника по трем сторонам

(формула Герона).

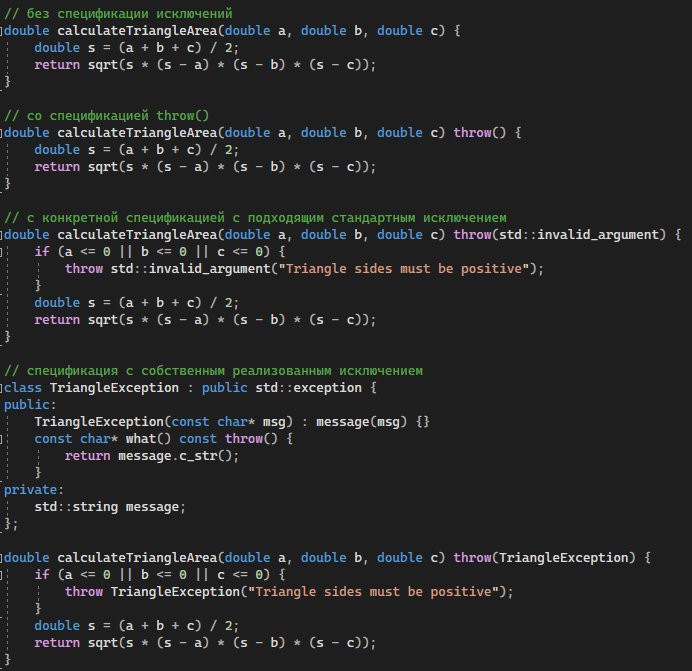
Функцию реализовать в 4 вариантах:

без спецификации исключений;

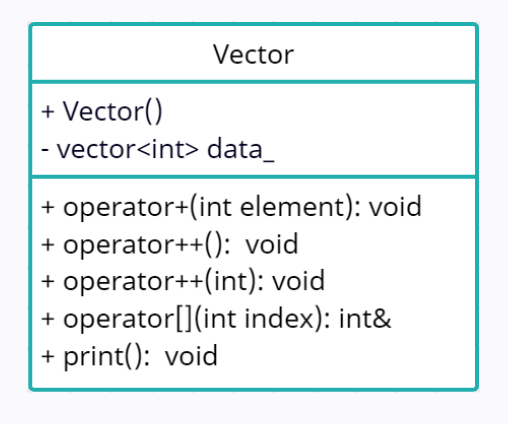
со спецификацией throw();

с конкретной спецификацией с подходящим стандартным исключением;

спецификация с собственным реализованным исключением.



UML таблица



Код программы

#include <iostream>

#include <stdexcept>

#include <vector>

using namespace std;

class Vector {

public:

Vector() {}

void operator+(int element) { // adds an element to the end of the vector

data\_.push\_back(element);

}

void operator++() { // adds an element to the beginning of the vector

int element;

cout << "Enter an element to add to the beginning of the vector: ";

cin >> element;

data\_.insert(data\_.begin(), element);

}

void operator++(int) { // adds an element to the end of the vector

int element;

cout << "Enter an element to add to the end of the vector: ";

cin >> element;

data\_.push\_back(element);

}

int& operator[](int index) { // access by index

if (index < 0 || index >= data\_.size()) {

throw std::out\_of\_range("Index out of range");

}

return data\_[index];

}

void print() {

cout << "Vector: ";

for (const auto& element : data\_) {

cout << element << " ";

}

cout << endl;

}

private:

vector<int> data\_;

};

int main() {

Vector v;

try {

v[0] = 10; // throws an exception

}

catch (const std::exception& e) {

cout << e.what() << endl;

}

int n;

cout << "Enter the number of elements in the vector: ";

cin >> n;

for (int i = 0; i < n; ++i) {

int element;

cout << "Enter element #" << i << ": ";

cin >> element;

v + element;

}

cout << "Enter an element to add to the beginning of the vector: ";

++v;

cout << "Enter an element to add to the end of the vector: ";

v++;

v.print();

cout << "Enter the index of the element to retrieve: ";

int index;

cin >> index;

try {

cout << "Element #" << index << " is " << v[index] << endl;

}

catch (const std::exception& e) {

cout << e.what() << endl;

}

return 0;

}