Министерство образования и науки РФ

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра Информационные технологии и автоматизированные системы

Лабораторная работа № 9

Тема: «Обработка исключительных ситуаций»

Выполнил: студент группы РИС-22-2б

Мизёв В.А. ф

Проверил: доцент кафедры ИТАС

Полякова О.А.

г. Пермь – 2024

*Постановка задачи:*

1. Реализовать класс, перегрузить для него операции, указанные в варианте.
2. Определить исключительные ситуации.
3. Предусмотреть генерацию исключительных ситуаций.

*Задание вариант 1:*

Класс – контейнер ВЕКТОР с элементами типа int

Реализовать операции:

[] – доступ по индексу;

() – определение размера вектора;

+ число – добавляет константу ко всем элементам вектора;

- n – удаляет n элементов из конца вектора.

*Анализ задачи:*

Основываясь на задании необходимо реализовать 2 класса, вектор и vector exception. Класс вектор будет иметь один конструктор по умолчанию, а также соответствующие перегруженные операции из варианта задания. Основные поля класса m\_data – указатель на область памяти, m\_capacity – кол-во выделенной памяти из кучи, m\_size – кол-во элементов в векторе и private функция realloc(), которая позволяет динамически выделять память на базе capacity и копировать данные массива из старой памяти в новую. Класс VectorException будет наследоваться из std::exception, из методов what() и getCode(), где what сообщение об ошибке, getCode, код ошибки.

*Код программы:*

Прикреплён в приложении 1.

*UML-диаграмма класса:*

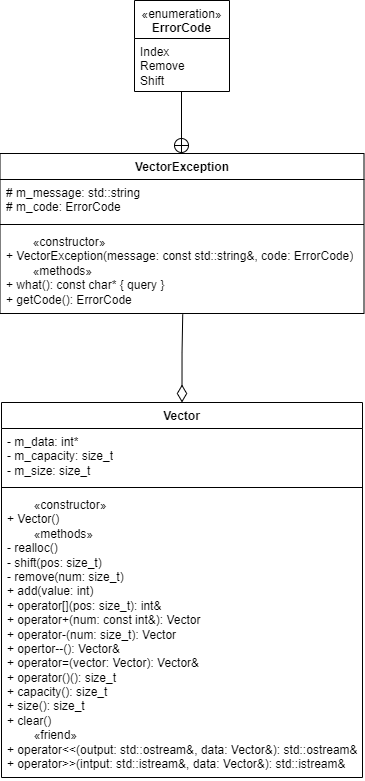


Рисунок 1 - UML-диаграмма классов

*Скриншот работы программы:*

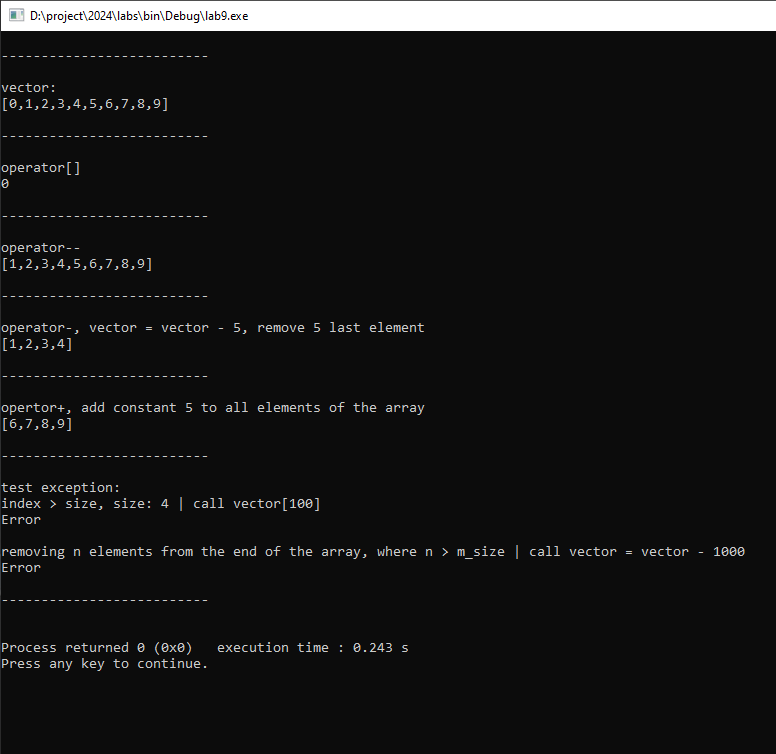


Рисунок 2 – Результат работы программы (стандартные исключения)

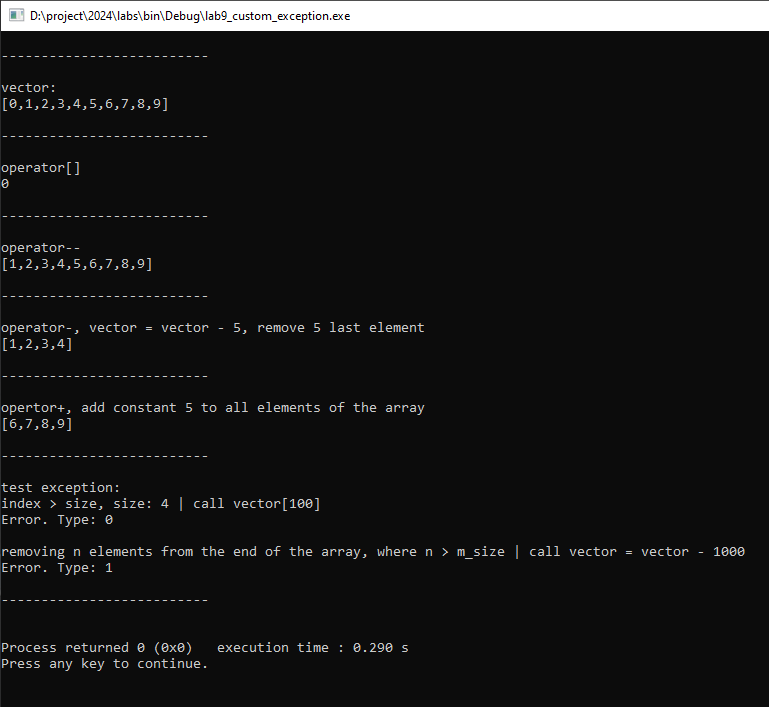


Рисунок 2 Результат работы программы (свой класс с исключениями)

*Контрольные вопросы:*

1. Что представляет собой исключение в С++?

Исключение в C++ — это механизм обработки ошибок, который позволяет программе реагировать на ошибки и исключительные ситуации, которые возникают во время выполнения. Исключения позволяют отделить обработку ошибок от основной логики программы.

#include <iostream>

#include <stdexcept>

void mightGoWrong() {

throw std::runtime\_error("Something went wrong!");

}

2. На какие части исключения позволяют разделить вычислительный процесс?

Достоинства такого подхода?

Вычислительный процесс можно разделить на:

- Блок кода, где может возникнуть исключение (обычно это то, что вызывает исключение).

- Блоки обработки исключений, которые могут реагировать на возникновение исключений (catch).

Достоинства:

- Чистота кода: обработка ошибок не смешивается с основной логикой.

- Упрощение отладки: легче отслеживать и понимать ошибки.

3. Какой оператор используется для генерации исключительной ситуации?

Для генерации исключительной ситуации используется оператор throw.

throw std::runtime\_error("Ошибка!");

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

Контролируемый блок — это блок кода, в котором могут возникать исключения, и он находится под контролем обработчиков исключений (try-catch).

try {

mightGoWrong();

} catch (std::runtime\_error &e) {

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

}

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

Секция-ловушка (catch) — это блок кода, который обрабатывает исключения, выброшенные в соответствующем блоке try. Она нужна, чтобы предотвратить аварийное завершение программы и обработать ошибки.

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

Спецификация в секции ловушки может иметь следующие формы:

- Поймать исключение по конкретному типу: catch (std::runtime\_error &e)

- Поймать исключение по базовому типу (включая производные): catch (std::exception &e)

- Поймать исключение по универсальному типу: catch (...)

Эти формы используются для обработки исключений определенных типов и более общих случаев.

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

Для создания собственной иерархии исключений можно использовать стандартный класс std::exception.

8. Каким образом можно создать собственную иерархию исключений?

Можно создать пользовательский класс, наследующий от std::exception, и переопределить метод what().

class MyException : public std::exception {

public:

const char\* what() const noexcept override {

return "My custom exception";

}

};

9.Если спецификация исключений имеет вид: void f1()throw(int,double); то какие

исключения может прождать функция f1()?

Функция f1() может выбрасывать исключения типов int и double.

10. Если спецификация исключений имеет вид: void fl() throw(); то какие

может прождать функция f1()?

Функция f1() не может выбрасывать исключения. Выбрасываемые исключения в данном случае не определены.

11. В какой части программы может генерироваться исключение?

Исключение может генерироваться в любой части программы, где происходит выполнение операций, которые могут привести к ошибкам. Это может быть работа с файлами, динамическое выделение памяти, деление на ноль, выход за границы массива и другие ситуации. Обычно такие операции оборачиваются в блоки try, а обработка исключений выполняется в блоках catch.

#include <iostream>

#include <stdexcept>

int main() {

try {

// Попытка выполнить деление на ноль

int a = 5;

int b = 0;

if (b == 0) {

throw std::runtime\_error("Деление на ноль!");

}

std::cout << a / b;

} catch (const std::exception &e) {

std::cout << "Ошибка: " << e.what() << std::endl;

}

return 0;

}

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.

Без спецификации исключений:

#include <cmath>

#include <iostream>

double area(double a, double b, double c) {

double s = (a + b + c) / 2;

return std::sqrt(s \* (s - a) \* (s - b) \* (s - c));

}

Со спецификатором throw();

#include <cmath>

#include <iostream>

double area(double a, double b, double c) throw() {

double s = (a + b + c) / 2;

return std::sqrt(s \* (s - a) \* (s - b) \* (s - c));

}

с конкретной спецификацией с подходящим стандартным исключением

#include <cmath>

#include <iostream>

#include <stdexcept>

double area(double a, double b, double c) throw(std::invalid\_argument) {

if (a <= 0 || b <= 0 || c <= 0) {

throw std::invalid\_argument("Сторона треугольника должна быть положительной!");

}

double s = (a + b + c) / 2;

return std::sqrt(s \* (s - a) \* (s - b) \* (s - c));

}

спецификация с собственным реализованным исключением:

#include <cmath>

#include <iostream>

#include <stdexcept>

class TriangleException : public std::exception {

public:

const char\* what() const noexcept override {

return "Некорректные стороны треугольника!";

}

};

double area(double a, double b, double c) {

if (a <= 0 || b <= 0 || c <= 0) {

throw TriangleException();

}

double s = (a + b + c) / 2;

return std::sqrt(s \* (s - a) \* (s - b) \* (s - c));}

**ПРИЛОЖЕНИЕ 1**

lab9.cpp

//-------------------------------------

#include <iostream>

#include "lab9\_vector.hpp"

//-------------------------------------

#define IDENT\_PRINT printf("\n--------------------------\n\n")

//-------------------------------------

int main() {

Vector vector;

for(size\_t i = 0; i < 10; ++i)

vector.add(i);

IDENT\_PRINT;

std::cout << "vector:" << std::endl;

std::cout << vector << std::endl;

IDENT\_PRINT;

std::cout << "operator[]" << std::endl;

std::cout << vector[0] << std::endl;

IDENT\_PRINT;

std::cout << "operator--" << std::endl;

--vector;

std::cout << vector << std::endl;

IDENT\_PRINT;

std::cout << "operator-, vector = vector - 5, remove 5 last element" << std::endl;

vector = vector - 5;

std::cout << vector << std::endl;

IDENT\_PRINT;

std::cout << "opertor+, add constant 5 to all elements of the array" << std::endl;

vector = vector + 5;

std::cout << vector << std::endl;

IDENT\_PRINT;

std::cout << "test exception:" << std::endl;

#ifdef CUSTOM\_EXCEPTION

try {

std::cout << "index > size, size: " << vector() << " | call " << "vector[100]" << std::endl;

std::cout << vector[100] << std::endl;

} catch (VectorException& e) {

std::cout << "Error. Type: " << e.getCode() << std::endl;

}

std::cout << std::endl;

try {

std::cout << "removing n elements from the end of the array, where n > m\_size" << " | call " << "vector = vector - 1000" << std::endl;

vector = vector - 1000;

} catch (VectorException& e) {

std::cout << "Error. Type: " << e.getCode() << std::endl;

}

#else

try {

std::cout << "index > size, size: " << vector() << " | call " << "vector[100]" << std::endl;

std::cout << vector[100] << std::endl;

} catch (std::exception& e) {

std::cout << "Error" << std::endl;

}

std::cout << std::endl;

try {

std::cout << "removing n elements from the end of the array, where n > m\_size" << " | call " << "vector = vector - 1000" << std::endl;

vector = vector - 1000;

} catch (std::exception& e) {

std::cout << "Error" << std::endl;

}

#endif

IDENT\_PRINT;

return 0;

}

//-------------------------------------

lab9\_vector.hpp

//-------------------------------------

#ifndef LAB9\_VECTOR\_HPP\_INCLUDED

#define LAB9\_VECTOR\_HPP\_INCLUDED

//-------------------------------------

#include <iostream>

#include "lab9\_vector\_exception.hpp"

//-------------------------------------

//uncomment to check for other exceptions

#define CUSTOM\_EXCEPTION

//-------------------------------------

#define BASE\_SIZE 4

//-------------------------------------

class Vector {

public:

Vector();

~Vector();

void add(int value);

int& operator[](size\_t pos);

Vector operator+(const int& num);

Vector operator-(size\_t num);

Vector& operator--();

Vector& operator=(Vector vector);

size\_t operator()(){ return m\_size; }

size\_t capacity() { return m\_capacity; }

size\_t size() { return m\_size; }

void clear();

friend std::ostream& operator<<(std::ostream& output, Vector& data);

friend std::istream& operator>>(std::istream& input, Vector& data);

private:

int\* m\_data;

size\_t m\_capacity,

m\_size;

void realloc();

void shift(size\_t pos);

void remove(size\_t num);

};

//-------------------------------------

#endif // LAB9\_VECTOR\_HPP\_INCLUDED

//-------------------------------------

lab9\_vector.cpp

//-------------------------------------

#include "lab9\_vector.hpp"

//-------------------------------------

Vector::Vector() {

m\_size = 0;

m\_capacity = 0;

this->realloc();

}

//-------------------------------------

Vector::~Vector() {

this->clear();

}

//-------------------------------------

void Vector::realloc() {

if(m\_size < m\_capacity)

return;

if(m\_capacity == 0) {

m\_capacity = BASE\_SIZE;

m\_data = new int[m\_capacity];

return;

}

m\_capacity\*=2;

int\* array = new int[m\_capacity];

for(size\_t i = 0; i < m\_size; ++i)

array[i] = m\_data[i];

delete []m\_data;

m\_data = array;

}

//-------------------------------------

void Vector::shift(size\_t pos) {

#ifdef CUSTOM\_EXCEPTION

if(m\_size == 0)

throw VectorException("m\_size == 0", VectorException::Shift);

else if(pos > m\_size)

throw VectorException("shift pos > m\_size", VectorException::Shift);

#else

if(m\_size == 0)

throw std::runtime\_error("m\_size == 0");

else if(pos > m\_size)

throw std::runtime\_error("shift pos > m\_size");

#endif // BASE\_THROW

m\_size -= pos;

if(m\_size == 0) {

this->clear();

return;

}

int\* array = new int[m\_capacity];

for(size\_t i = 0; i < m\_size; ++i)

array[i] = m\_data[i+pos];

delete []m\_data;

m\_data = array;

}

//-------------------------------------

void Vector::remove(size\_t num) {

#ifdef CUSTOM\_EXCEPTION

if(m\_size == 0)

throw VectorException("m\_size == 0", VectorException::Remove);

else if(num > m\_size)

throw VectorException("num > m\_size", VectorException::Remove);

#else

if(m\_size == 0)

throw std::runtime\_error("m\_size == 0");

else if(num > m\_size)

throw std::runtime\_error("num > m\_size");

#endif //BASE\_EXCEPTION

m\_size -= num;

}

//-------------------------------------

void Vector::add(int value) {

this->realloc();

m\_data[m\_size] = value;

m\_size++;

}

//-------------------------------------

int& Vector::operator[](size\_t pos) {

#ifdef CUSTOM\_EXCEPTION

if(m\_size <= pos)

throw VectorException("m\_size <= pos", VectorException::Index);

#else

if(m\_size <= pos)

throw std::range\_error("m\_size <= pos");

#endif // BASE\_EXCEPTION

return m\_data[pos];

}

//-------------------------------------

Vector Vector::operator+(const int& num) {

Vector vector;

for(size\_t i = 0; i < this->size(); ++i)

vector.add((\*this)[i]+num);

return vector;

}

//-------------------------------------

Vector Vector::operator-(size\_t num) {

/\*Vector vector = \*this;

vector.shift(num);\*/

Vector vector = \*this;

vector.remove(num);

return vector;

}

//-------------------------------------

Vector& Vector::operator--() {

shift(1);

return \*this;

}

//-------------------------------------

Vector& Vector::operator=(Vector vector) {

if(&vector == this)

return \*this;

this->clear();

for(size\_t i = 0; i < vector.size(); ++i)

this->add(vector[i]);

return \*this;

}

//-------------------------------------

void Vector::clear() {

if(m\_data != nullptr)

delete []m\_data;

m\_size = 0;

m\_capacity = 0;

}

//-------------------------------------

std::ostream& operator<<(std::ostream& output, Vector& data) {

output << "[";

for(size\_t i = 0; i < data.size(); ++i) {

output << data[i];

if(data.size()-1 != i)

output << ",";

}

output << "]";

return output;

}

//-------------------------------------

std::istream& operator>>(std::istream& input, Vector& data) {

std::cout << "add " << data.size() << " elements: ";

int a;

input >> a;

data.add(a);

return input;

}

//-------------------------------------

lab9\_vector\_exception.hpp

//-------------------------------------

#ifndef LAB9\_VECTOR\_EXCEPTION\_HPP\_INCLUDED

#define LAB9\_VECTOR\_EXCEPTION\_HPP\_INCLUDED

//-------------------------------------

#include <exception>

//-------------------------------------

class VectorException : public std::exception {

public:

enum ErrorCode {

Index = 0,

Remove,

Shift,

};

public:

explicit VectorException(const std::string& message, ErrorCode code) {

m\_message = message;

m\_code = code;

}

virtual ~VectorException() noexcept { }

virtual const char\* what() const noexcept {

return m\_message.c\_str();

}

ErrorCode getCode() noexcept {

return m\_code;

}

protected:

std::string m\_message;

ErrorCode m\_code;

};

//-------------------------------------

#endif // LAB9\_VECTOR\_EXCEPTION\_HPP\_INCLUDED

//-------------------------------------