Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

Дисциплина: «Основы алгоритмизации и программирования»

Тема: «Обработка исключительных ситуаций»

Выполнил работу

Студент группы РИС-23-3Б

Епин Т. Е.

Проверил

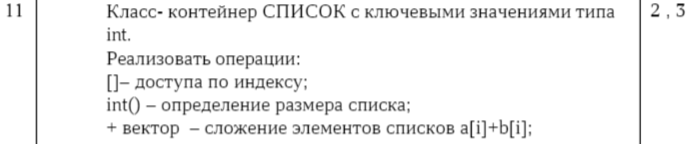
Доцент кафедры ИТАС

Полякова О.А.

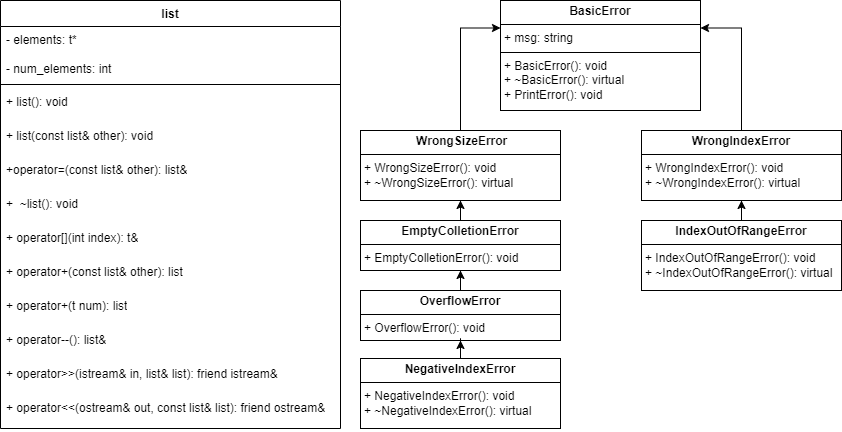
Г. Пермь-2024

**Постановка задачи:**

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

****

**Uml диаграмма:**

****

**Код программы:**

**List.h**

#pragma once

#include "error.h"

#include <string>

#include <iostream>

using namespace std;

const int MAX\_SIZE = 30;

// структура узла списка

struct Node

{

Node\* next = nullptr;

Node\* prev = nullptr;

int Data;

};

// шаблонный класс списка

template<typename t>

class list

{

public:

// конструктор

list() : elements(nullptr), num\_elements(0) {}

// конструктор копирования

list(const list& other) : elements(new t[other.num\_elements]), num\_elements(other.num\_elements)

{

for (int i = 0; i < num\_elements; i++)

elements[i] = other.elements[i];

}

// оператор присваивания

list& operator=(const list& other)

{

if (this != &other)

{

delete[] elements;

elements = new t[other.num\_elements];

num\_elements = other.num\_elements;

for (int i = 0; i < num\_elements; i++)

elements[i] = other.elements[i];

}

return \*this;

}

// деструктор

~list() { delete[] elements; }

// доступ по индексу

t& operator[](int index)

{

if (index >= num\_elements) throw IndexOutOfRangeError();

if (index < 0) throw NegativeIndexError();

return elements[index];

}

list& operator--()

{

num\_elements--;

return \*this;

}

// ввод списка

friend istream& operator>>(istream& in, list& list)

{

int num\_elements;

in >> num\_elements;

if (num\_elements > MAX\_SIZE) throw OverflowError();

if (num\_elements == 0) throw EmptyColletionError();

list.elements = new t[num\_elements];

list.num\_elements = num\_elements;

for (int i = 0; i < num\_elements; i++)

in >> list.elements[i];

return in;

}

// вывод списка

friend ostream& operator<<(ostream& out, const list& list)

{

for (int i = 0; i < list.num\_elements; i++)

out << list.elements[i] << " ";

return out;

}

// перегрузка +

list<t> operator+(const list<t>& other) {

list<t> res;

res.num\_elements = num\_elements + other.num\_elements;

res.elements = new t[res.num\_elements];

for (int i = 0; i < num\_elements; i++) {

res.elements[i] = elements[i];

}

for (int i = 0; i < other.num\_elements; i++) {

res.elements[i + num\_elements] = other.elements[i];

}

return res;

}

private:

t\* elements;

int num\_elements;

};

**Error.h**

#pragma once

#include <string>

#include <iostream>

using namespace std;

class BasicError {

public:

BasicError() {}

virtual ~BasicError() {}

void PrintError() {

cout << msg << endl;

}

protected:

string msg;

};

class WrongSizeError : public BasicError {

public:

WrongSizeError() { msg = "Ошибка неверного размера: "; }

virtual ~WrongSizeError() {}

};

class EmptyColletionError : public WrongSizeError {

public:

EmptyColletionError() { msg += "Список пуст"; }

};

class OverflowError : public WrongSizeError {

public:

OverflowError() { msg += "Список не может содержать более 30 элементов"; }

};

class WrongIndexError : public BasicError {

public:

WrongIndexError() { msg = "Ошибка индекса: "; }

virtual ~WrongIndexError() {}

};

class IndexOutOfRangeError : public WrongIndexError {

public:

IndexOutOfRangeError() { msg += "Индекс больше размера списка"; }

~IndexOutOfRangeError() {}

};

class NegativeIndexError : public WrongIndexError {

public:

NegativeIndexError() { msg += "Индекс отрицательный"; }

~NegativeIndexError() {}

};

**Main.cpp**

#include <iostream>

#include"list.h"

#include"error.h"

int main()

{

system("chcp 1251");

int a = 0;

while (true) {

try

{

// создание списков

list<int> list1;

list<int> list2;

cout << endl;

// ввод списков

cout << "Введите количество элементов первого списка: ";

cin >> list1;

cout << "Введите количество элементов второго списка: ";

cin >> list2;

cout << endl;

// сложение списков

list<int> result = list1 + list2;

cout << "Сумма списков: " << result << endl;

cout << endl;

}

catch (IndexOutOfRangeError e) {

e.PrintError();

continue;

}

catch (NegativeIndexError e) {

e.PrintError();

continue;

}

catch (BasicError& err)

{

err.PrintError();

char c;

cout << "Хотите начать заново? (1/0): ";

cin >> c;

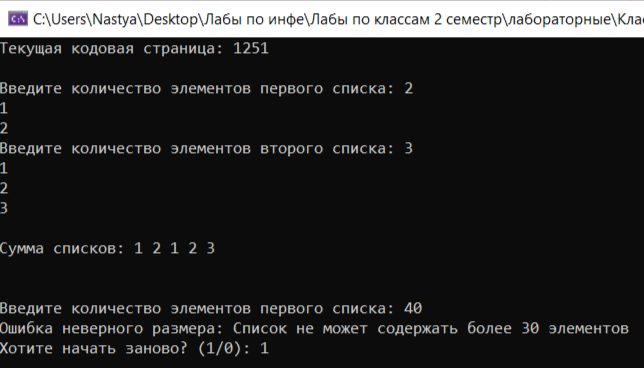
if (c == '0') break;

}

}

}

**Результат работы программы:**

****

**Ответы на контрольные вопросы:**

1. Что представляет собой исключение в С++?

Исключение в С++ представляет собой специальный объект, который генерируется в процессе выполнения программы при возникновении ошибки или иной исключительной ситуации.

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

Исключения позволяют разделить вычислительный процесс на две части: блок кода, который может вызвать исключение, и блок кода, который обрабатывает это исключение. Достоинством такого подхода является возможность более точной обработки ошибок и упрощение кода за счет выноса обработки ошибок в отдельные блоки.

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, в котором может произойти исключение. Для того чтобы контролировать и обрабатывать это исключение, необходимо обернуть этот блок кода в try-блок.

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

Секция-ловушка представляет собой блок кода, который обрабатывает исключение, сгенерированное в try-блоке. Она нужна для того, чтобы определить, какие действия необходимо выполнить при возникновении исключения.

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

Спецификация исключения может иметь форму типа данных или множества типов данных, которые могут быть сгенерированы в try-блоке. В ситуациях, когда необходимо обрабатывать несколько типов исключений, используется множественная форма спецификации. В ситуациях, когда необходимо обработать любое исключение, используется форма без указания типа данных.

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

Стандартный класс, который можно использовать для создания собственной иерархии исключений, называется std::exception.

8. Каким образом можно создать собственную иерархию исключений?

Для создания собственной иерархии исключений необходимо создать класс, наследующийся от класса std::exception или его производных классов, и определить в нем необходимые методы и поля.

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

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

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

Функция f1() не ожидает никаких исключений.

11. В какой части программы может генерироваться исключение?

Исключение может генерироваться в любой части программы, но обычно это происходит в тех участках кода, где есть потенциальная возможность возникновения ошибок или исключительных ситуаций. Например, это могут быть операции с файлами, работа с сетью, выделение памяти и т.д.

12. Написать функцию, которая вычисляет площадь треугольника по трем сторонам (формула Герона).

Функцию реализовать в 4 вариантах:

- без спецификации исключений;

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

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

return (sqrt(p \* (p - a) \* (p - b) \* (p - c)));

}

- со спецификацией throw();

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

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

return sqrt(p \* (p - a) \* (p - b) \* (p - c));

}

- с конкретной спецификацией с подходящим стандартным исключением;

#include <stdexcept>

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

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

throw std::invalid\_argument("Invalid triangle sides");

}

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

return sqrt(p \* (p - a) \* (p - b) \* (p - c));

}

- спецификация с собственным реализованным исключением.

#include <exception>

class InvalidTriangleException : public std::exception {

public:

virtual const char\* what() const throw() {

return "Invalid triangle sides";

}

};

double triangleArea(double a, double b, double c) throw(InvalidTriangleException) {

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

throw InvalidTriangleException();

}

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

return sqrt(p \* (p - a) \* (p - b) \* (p - c));

}