Министерство образования и науки Российской Федерации  
Федеральное государственное автономное образовательное учреждение высшего образования

«ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

(ПНИПУ)

Электротехнический факультет

Кафедра Информационные технологии и автоматизированные системы

Лабораторные работы по

«Основам алгоритмизации и программирования»

вариант № 15

за 2 семестр

Выполнил:

студент группы РИС-21-1бз

Тимолянов Григорий Константинович

Проверила:

Доцент кафедры ИТАС,

к.т.н. Полякова О.А.

2024

###### **Лабораторная работа №9**

###### **Обработка исключительных ситуаций**

**Цель:** 1) Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.

2) Разработка программы, обрабатывающей исключительные ситуации

###### **Постановка задачи**

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

Класс- контейнер МНОЖЕСТВО с значениями типа int.

Реализовать операции:

[]– доступа по индексу;

int() – определение размера списка;

\* – пересечение множеств;

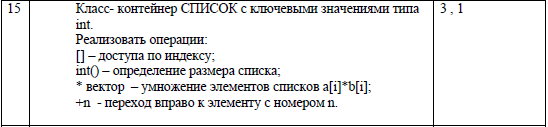
-- - удаление элемента из множества.

###### **Словесное описание исключительных ситуаций.**

Исключительные ситуации генерируются:

1. При создании контейнера нулевого или отрицательного размера.
2. При попытке обратиться к индексу элемента, номер которого отрицательный или больше размера контейнера, то есть при выходе за любые его границы.

###### Описание классов.



**Решение:**

Файл Исключительные ситуации.cpp (вариант 1)

#include "Listing.h"

#include <iostream>

using namespace std;

int main()

{

setlocale(LC\_ALL, "Rus");

int i;

// Обработка исключительных ситуаций

try {

cout << "Пустой список из двух элементов: \n";

// Список с 2 элементами

Listing x(2);

// Пустой список

Listing y;

cout << x << "\nВведите номер элемента: \n";

cin >> i;

cout << "\nВывод элемента под номером " << i << ":\n";

cout << x[i] << endl;

y = x \* 1;

cout << y;

}

catch(int) {

cout << "Ошибка!\n";

}

return 0;

}

Файл Listing.cpp (вариант реализации 1)

#include "Listing.h"

// Генерация исключения при текущий размер > максимальный размер

Listing::Listing(int s) {

if (s > MAX\_SIZE) { throw 1; }

size = s;

beg = new int[s];

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

beg[i] = 0;

}

}

Listing::Listing(const Listing &l) {

size = l.size;

beg = new int[size];

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

beg[i] = l.beg[i];

}

}

Listing::~Listing() {

if (beg != 0) {

delete[] beg;

}

}

Listing::Listing(int s, int \*mas) {

if (s > MAX\_SIZE) { throw 1; }

size = s;

beg = new int[size];

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

beg[i] = mas[i];

}

}

const Listing& Listing::operator = (const Listing& l) {

if (this == &l) { return \*this; }

if (beg != 0) { delete[]beg; }

size = l.size;

beg = new int[size];

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

beg[i] = l.beg[i];

}

return \*this;

}

ostream& operator << (ostream& out, const Listing& l) {

if (l.size == 0) { out << "Пусто\n"; }

else {

for (int i = 0; i < l.size; i++) {

out << l.beg[i] << " ";

}

out << endl;

}

return out;

}

istream& operator >> (istream& in, Listing& l) {

for (int i = 0; i < l.size; i++) {

cout << ">";

in >> l.beg[i];

}

return in;

}

// Если индекс отрицательный, генерируется исключение

int Listing::operator [] (int i) {

if (i < 0) {

throw 2;

}

}

Listing Listing::operator\*(int a) {

Listing temp(size, beg);

size++;

beg = new int[size];

beg[0] = temp.beg[0] \* temp.beg[a];

return \*this;

}

Файл Listing.h (вариант реализации 1)

#pragma once

#include <iostream>

#include <string>

#include <Windows.h>

using namespace std;

const int MAX\_SIZE = 30;

class Listing {

// Дружественные функции ввода-вывода

friend ostream& operator << (ostream& out, const Listing& l);

friend istream& operator >> (istream& in, Listing& l);

public:

int size;

int\* beg;

Listing() { size = 0; beg = 0; }

Listing(int s);

Listing(int s, int\* mas);

Listing(const Listing& l);

~Listing();

const Listing& operator = (const Listing& l);

int operator[] (int i);

Listing operator\*(int a);

};

Файл Исключ. сит.cpp (вариант реализации 2)

#include "Listing.h"

#include "Error.h"

#include <iostream>

using namespace std;

int main()

{

setlocale(LC\_ALL, "Rus");

try {

cout << "Список с 2 пустыми элементами:\n";

Listing x(2);

Listing y;

cout << x;

cout << "\nВведите номер элемента:\n";

int i;

cin >> i;

cout << x[i] << endl;

y = x + 3;

cout << y;

//x \* 1;

cout << x;

}

catch (Error& e) {

e.what();

}

return 0;

}

Файл Listing.cpp (вариант реализации 2)

#include "Listing.h"

#include "Error.h"

#include <iostream>

using namespace std;

Listing::Listing(int s) {

if (s > MAX\_SIZE) throw MaxSizeError();

size = s;

beg = new int[s];

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

beg[i] = 0;

}

}

Listing::Listing(const Listing& l) {

size = l.size;

beg = new int[size];

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

beg[i] = l.beg[i];

}

}

Listing::~Listing() {

if (beg != 0) {

delete[] beg;

}

}

Listing::Listing(int s, int\* mas) {

size = s;

beg = new int[size];

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

beg[i] = mas[i];

}

}

const Listing& Listing::operator = (const Listing& l) {

if (this == &l) { return \*this; }

if (beg!= 0) { delete[] beg; }

size = l.size;

beg = new int[size];

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

beg[i] = l.beg[i];

}

return \*this;

}

ostream& operator << (ostream& out, const Listing& l) {

if (l.size == 0) { out << "Пусто\n"; }

else {

for (int i = 0; i < l.size; i++) {

out << l.beg[i] << " ";

}

out << endl;

}

return out;

}

istream& operator >> (istream& in, Listing& l) {

for (int i = 0; i < l.size; i++) {

cout << ">";

in >> l.beg[i];

}

return in;

}

int Listing::operator [] (int i) {

if (i < 0) {

throw IndexError1();

}

if (i >= size) {

throw IndexError2();

}

return beg[i];

}

Listing Listing::operator+(int a) {

if (size + 1 == MAX\_SIZE) {

throw MaxSizeError();

}

Listing temp(size + 1, beg);

temp.beg[size] = a;

return temp;

}

Listing Listing::operator\*(int a) {

Listing temp(size, beg);

size++;

beg = new int[size];

beg[0] = temp.beg[0] \* temp.beg[a];

return \*this;

}

Файл Listing.h (вариант реализации 3)

#pragma once

#include <iostream>

#include <string>

#include <Windows.h>

using namespace std;

const int MAX\_SIZE = 30;

class Listing {

int size;

int\* beg;

friend ostream& operator << (ostream& out, const Listing& l);

friend istream& operator >> (istream& in, Listing& l);

public:

Listing() { size = 0; beg = 0; };

Listing(int s);

Listing(int s, int\* mas);

Listing(const Listing& l);

~Listing();

const Listing& operator = (const Listing& l);

int operator[] (int i);

Listing operator+(int a);

Listing operator\*(int a);

};

Error.h (вариант реализации 3)

#pragma once

#include <string>

#include <iostream>

using namespace std;

// Базовая структура

struct Error {

virtual void what() {};

};

// Ошибка в индексе вектора

struct IndexError : public Error {

IndexError() { msg = "Ошибка индекса\n"; }

virtual void what() { cout << msg; }

protected:

string msg;

};

// Ошибка в размере вектора

struct SizeError : public Error {

SizeError() { msg = "Ошибка длины\n"; }

virtual void what() { cout << msg; }

protected:

string msg;

};

// Превышение макс. размера

struct MaxSizeError : public SizeError {

MaxSizeError() { SizeError(); msg = "Длина > MAXSIZE\n"; }

virtual void what() { cout << msg << msg\_; }

protected:

string msg\_;

};

// Удаление из пустого вектора

struct EmptySizeError : public SizeError {

EmptySizeError() { SizeError(); msg\_ = "Вектор пустой\n"; }

virtual void what() { cout << msg << msg\_; }

protected:

string msg\_;

};

// Индекс < 0

struct IndexError1 : public IndexError {

IndexError1() { IndexError(); msg\_ = "Индекс <0\n"; }

virtual void what() { cout << msg << msg\_; }

protected:

string msg\_;

};

// Индекс больше текущего размера вектора

struct IndexError2 :public IndexError {

IndexError2() { IndexError(); msg\_ = "Индекс > длины\n"; }

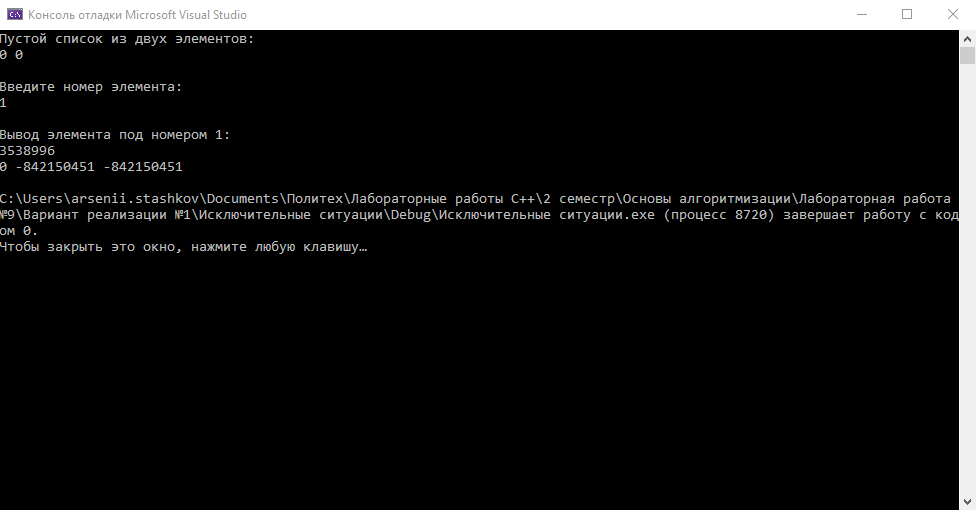
virtual void what() { cout << msg << msg\_; }

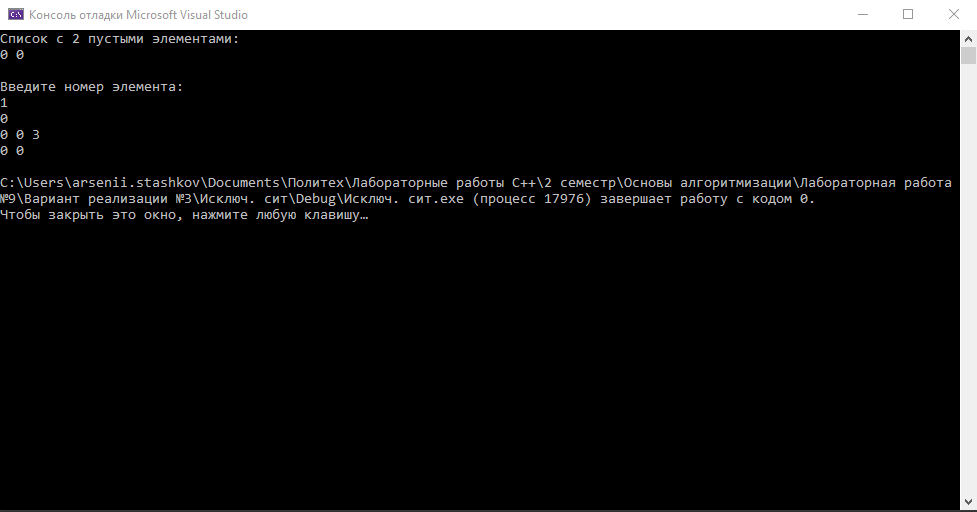
protected:

string msg\_;

};

**Результат вариант 1:**



**Результат вариант 2:**

###### **Ответы на контрольные вопросы.**

1. Что представляет собой исключение в С++?

Исключение – это непредвиденное или аварийное событие. В С++ генерируется объект при возникновении исключительной ситуации.

1. На какие части исключения позволяют разделить вычислительный процесс?

* обнаружение аварийной ситуации
* обработка аварийной ситуации

Достоинства такого подхода?

* удобно использовать в программе, которая состоит из нескольких модулей;
* не требуется возвращать значение в вызывающую функцию

1. Какой оператор используется для генерации исключительной ситуации?

Используется оператор throw

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

Используется для проверки возникновения исключения try { } , с которым связаны одна или несколько ловушек исключений catch

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

catch. Перехватывает и обрабатывает возникшее исключение.

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

Форма записи : catch (спецификация исключения), где спецификация исключения может иметь три формы:

* (тип имя)
* (тип)
* (…)

Первая форма- объект передается в блок обработки, чтобы его каким-то образом там использовать, например, для вывода информации в сообщении об ошибке.

Третья фора – перехватывает все исключения, которые не попали в первые две.

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

Можно определять собственные исключения, унаследовав их от класса exception.

1. Каким образом можно создать собственную иерархию исключений?

Для создания собственной иерархии исключений надо объявить свой базовый классисключение, например:

class BaseException{};

Остальные классы будут наследниками этого класса, аналогично тому, как это сделано в иерархии стандартных исключений:

class Child\_Exception1:public BaseException{};

class Child\_Exception2:public BaseException{};

Класс BaseException можно унаследовать от стандартного класса exception

class BaseException: public exception{};

Наследование от стандартных классов позволит использовать метод what для вывода сообщений об ошибках.

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

Порождает только те исключения, которые есть в спецификации.

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

Спецификация пуста, значит никаких.

1. В какой части программы может генерироваться исключение?

В любой.

1. 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:

double areaTriangle(const double& a, const double& b, const double& c)

{

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

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

}

Со спецификацией throw():

double areaTriangle(const double& a, const double& b, const double& c)

{

if (a <= 0.0 || b <= 0.0 || c <= 0.0)

{

throw 1;

}

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

if (halfPerimeter <= 0)

{

throw 2;

}

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

}

C конкретной спецификацией с подходящим стандартным исключением:

double areaTriangle(const double& a, const double& b, const double& c)

{

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

if (halfPerimeter <= 0)

{

throw std::invalid\_argument("Error! halfPerimeter <= 0");

}

double x = halfPerimeter \* (halfPerimeter - a) \* (halfPerimeter - b) \* (halfPerimeter - c);

if (x <= 0)

{

throw std::invalid\_argument("x must be >= 0");

}

return std::sqrt(x);

}

Спецификация с собственным реализованным исключением:

class Error

{

public:

void what()

{

std::cout << "Error! Negative number";

}

};

double areaTriangle(const double& a, const double& b, const double& c)

{

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

if (halfPerimeter <= 0)

{

throw Error();

}

double x = halfPerimeter \* (halfPerimeter - a) \* (halfPerimeter - b) \* (halfPerimeter - c);

if (x <= 0)

{

throw Error();

}

return std::sqrt(x);

}