Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное учреждение высшего образования  
**«Пермский национальный исследовательский политехнический университет»**

**(ПНИПУ)**

Электротехнический факультет  
Кафедра «Информационные технологии и автоматизированные системы»

**Отчет о лабораторной работе №9**

Вариант №4

Дисциплина: основы алгоритмизации и программирования

Тема: классы в С++

Выполнил студент ИВТ-22-2б:

Якушев Михаил Витальевич

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата, подпись)

Проверил доцент кафедры ИТАС:

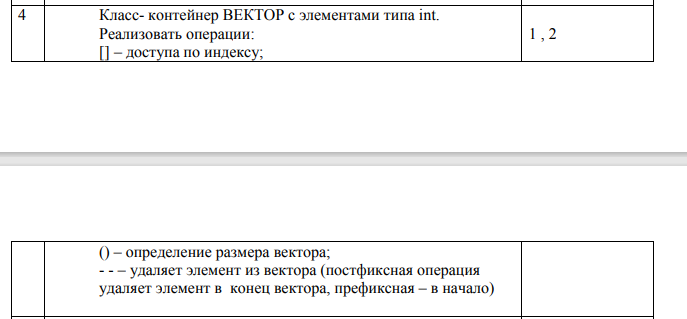
Полякова Ольга Андреевна

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата, подпись)

Пермь 2023

**1 Постановка задачи**

****

**2 Анализ задачи**

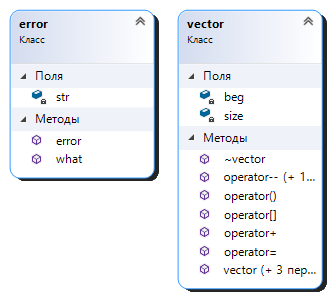
Исключение – это непредвиденное или аварийное событие. Обработаю исключения двумя способами: с помощью иерархии пользовательских классов и с помощью стандартного типа данных.

Блок try используется для проверки возникновения исключения, соответственно в блок try помещается та часть кода, в которой возможно возникновение исключения.

Оператор throw - передает заданный объект в один из операторов catch.

сatch(<тип> <имя объекта>) – при такой форме записи оператор принимает сам объект, и переданный объект может использоваться в самом блоке catch. Во втором случае используются системные исключения.

**3 UML-диаграммы**

****

**4 Код программы**

#include "error.h"

//#include "Vector.h"

#include "Vector1.h"

#include <iostream>

using namespace std;

int main()

{

setlocale(LC\_ALL, "ru");

try {

vector x(5);

vector y;

cout << x;

cin >> x;

cout << "Введите индекс: ";

int i,m;

cin >> i;

cout << x[i] << endl;

cout << "Размер: " << x() <<endl;

cout << "Введите число, на которое нужно увеличить каждый элемент вектора x: ";

cin >> m;

y = x + m;

cout << "Вектор y: " << y;

x--;

cout << x;

--x;

cout << x;

}

/\* catch (error a) {

a.what();

}\*/

catch (int) {

cout << "ERROR!!!" << endl;

}

return 0;

}

**Vector1.h**

#pragma once

#include <iostream>

#include "error.h"

using namespace std;

const int MAX\_SIZE = 30;

class vector {

int size;

int\* beg;

public:

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

vector(int);

vector(int, int\*);

vector(const vector&);

~vector();

vector& operator=(const vector&);

int operator[](int);

vector operator+(int);

vector operator--();

vector operator--(int);

int operator()();

friend ostream& operator<<(ostream& out, const vector&);

friend istream& operator>>(istream& in, vector&);

};

vector::vector(int s) {

if (s > MAX\_SIZE) throw 1;

size = s;

beg = new int[s];

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

beg[i] = 0;

}

};

vector::vector(const vector& a) {

size = a.size;

beg = new int[size];

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

beg[i] = a.beg[i];

}

};

vector::~vector() {

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

};

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

if (s > MAX\_SIZE) throw 1;

size = s;

beg = new int[s];

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

beg[i] = mas[i];

}

};

vector& vector::operator=(const vector& a) {

if (this == &a) return \*this;

if (beg != 0)

delete[]beg;

size = a.size;

beg = new int[size];

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

beg[i] = a.beg[i];

}

return \*this;

};

int vector:: operator()() {

if (size == 0) throw 5;

else

return size;

};

vector vector::operator--(int) {

if (size == 0) throw 5;

if (size == 1) {

size = 0;

delete[]beg;

beg = 0;

return \*this;

}

vector tmp(size, beg);

delete[]beg;

size--;

beg = new int[size];

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

beg[i] = tmp.beg[i];

return \*this;

};

vector vector::operator--() {

if (size == 0) throw 5;

if (size == 1) {

size = 0;

delete[]beg;

beg = 0;

return \*this;

}

vector tmp(size, beg);

delete[]beg;

size--;

beg = new int[size];

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

beg[i] = tmp.beg[i + 1];

return \*this;

};

ostream& operator<<(ostream& out, const vector& a) {

if (a.size == 0) out << "Empty\n";

else {

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

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

out << endl;

}

return out;

};

istream& operator>>(istream& in, vector& a) {

cout << "Введите элементы: " << endl;

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

cout << '>';

in >> a.beg[i];

}

return in;

};

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

if (i < 0)throw 2;

if (i >= size)throw 3;

return beg[i];

};

vector vector::operator+(int a) {

vector temp(size);

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

temp.beg[i] = beg[i] + a;

return temp;

};

**Vector.h**

#pragma once

#include <iostream>

#include "error.h"

using namespace std;

const int MAX\_SIZE = 30;

class vector {

int size;

int\* beg;

public:

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

vector(int);

vector(int, int\*);

vector(const vector&);

~vector();

vector& operator=(const vector&);

int operator[](int);

vector operator+(int);

vector operator--();

vector operator--(int);

int operator()();

friend ostream& operator<<(ostream& out, const vector&);

friend istream& operator>>(istream& in, vector&);

};

vector::vector(int s) {

if (s > MAX\_SIZE) throw error("Vector lenght more than MAXSIZE\n");

size = s;

beg = new int[s];

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

beg[i] = 0;

}

};

vector::vector(const vector& a) {

size = a.size;

beg = new int[size];

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

beg[i] = a.beg[i];

}

};

vector::~vector() {

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

};

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

if (s > MAX\_SIZE) throw error("Vector lenght more than MAXSIZE\n");

size = s;

beg = new int[s];

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

beg[i] = mas[i];

}

};

vector& vector::operator=(const vector& a) {

if (this == &a) return \*this;

if (beg != 0)

delete []beg;

size = a.size;

beg = new int[size];

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

beg[i] = a.beg[i];

}

return \*this;

};

int vector:: operator()() {

if (size == 0) throw error("vector is empty");

else

return size;

};

vector vector::operator--(int) {

if (size == 0) throw error("vector is empty");

if (size == 1) {

size = 0;

delete[]beg;

beg = 0;

return \*this;

}

vector tmp(size,beg);

delete[]beg;

size--;

beg = new int[size];

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

beg[i] = tmp.beg[i];

return \*this;

};

vector vector::operator--() {

if (size == 0) throw error("vector is empty");

if (size == 1) {

size = 0;

delete[]beg;

beg = 0;

return \*this;

}

vector tmp(size, beg);

delete[]beg;

size--;

beg = new int[size];

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

beg[i] = tmp.beg[i+1];

return \*this;

};

ostream& operator<<(ostream& out, const vector& a) {

if (a.size == 0) out << "Empty\n";

else {

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

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

out << endl;

}

return out;

};

istream& operator>>(istream& in, vector& a) {

cout << "Введите элементы: " <<endl;

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

cout << '>';

in >> a.beg[i];

}

return in;

};

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

if (i < 0)throw error("index < 0");

if (i >= size)throw error("index > size!");

return beg[i];

};

vector vector::operator+(int a) {

vector temp(size);

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

temp.beg[i] = beg[i] + a;

return temp;

};

**error.h**

#pragma once

#include <iostream>

#include <string>

using namespace std;

class error {

string str;

public:

error(string str) {

this->str = str;

};

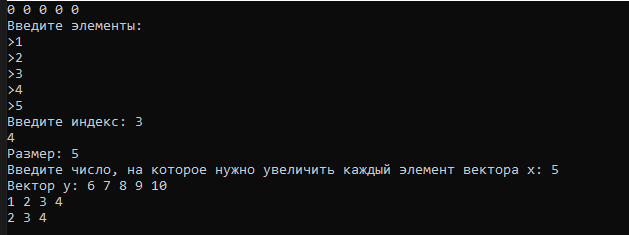
void what() {

cout << str << endl;

};

};

**5 Результаты работы программы**

****

**6 Ответы на вопросы**

1. Что представляет собой исключение в С++?

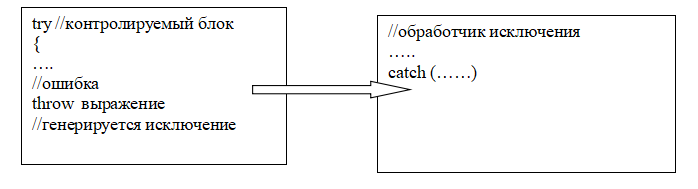
Исключение – это непредвиденное или аварийное событие.

В С++ исключение – это объект, который система должна генерировать при возникновении исключительной ситуации. Генерация такого объекта и создает исключительную ситуацию.

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

Исключения позволяют разделить вычислительный процесс на 2 части:

1. Обнаружение аварийной ситуации (неизвестно как обрабатывать);
2. Обработка аварийной ситуации (неизвестно, где она возникла).



1. Какой оператор используется для генерации исключительной ситуации?

Исключение генерируется оператором

throw <выражение>, где <выражение> -

- либо константа,

- либо переменная некоторого типа,

- либо выражение некоторого типа.

Тип объекта-исключения может быть как встроенным, так и определяемым пользователем. Для представления исключений часто используют пустой класс: class ZeroDevide {};

class NegativeArg {}

Генерация исключения будет выглядеть:

throw ZeroDevide();//вызывается конструктор без параметров

или

throw new ZeroDevide();

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

Исключение надо перехватить и обработать. Для проверки возникновения исключения используется контролируемый блок try{}, с которым связана одна или несколько секций-ловушек catch. То есть контролируемым блоком называется та часть кода, проверяемая блоком try, в которой предполагается возникновение ошибки.

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

После блока try всегда прописывается одна или несколько секций-ловушек catch, которые обычно называют обработчиками исключений.

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

Форма записи секции-ловушки следующая:

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

1)(тип имя)

2)(тип) 3) (…)

Тип – это встроенный тип или тип, определённый программистом.

Формы 1 и 2 обрабатывают конкретные исключения, а форма 3 перехватывает все исключения, такую ловушку надо помещать последней, тогда она будет обрабатывать все исключения, которые ещё не были обработаны.

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

Класс exception определён в стандартной библиотеке следующим образом:

class exception

{

public:

exception() throw();//конструктор без параметров

exception(const exception&) throw();//конструктор копирования

exception& operator= (const exception&) throw();//оператор =

virtual ~exception() throw();//деструктор

virtual const char\* what() const throw();//генерирует сообщение об ошибке

};

1. Каким образом можно создать собственную иерархию исключений?

Для создания собственной иерархии исключений надо:

- подключить библиотеку <exception>

- объявить свой базовый класс- исключение, например class BaseException {};

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

class BaseException {};

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.

Иерархия классов-исключений позволяет вместо нескольких разных блоков-ловушек написать единственный блок с типом аргумента базового класса.

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

Данная функция генерирует те исключения, которые явно указаны в этом списке, то есть int и double.

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

Поскольку спецификация исключений в заголовке не задана, функция может порождать любое исключение

1. В какой части программы может генерироваться исключение?

Исключение генерируется в той части кода, где используется контролируемый блок try(). Само исключение как объект возникает в точке генерации, распознаётся в контролируемом блоке и передаётся в обработчик исключений.

1. Написать функцию, которая вычисляет площадь треугольника по трём 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:

int Geron(int a, int b, int c)

{

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

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

}

- со спецификацией throw():

int Geron(int a, int b, int c) throw()

{

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

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

}

void main()

{

int a, b, c;

cin >> a >> b >> c;

try { Geron(a, b, c); }

catch (...) { cerr << "Возникла ошибка" << endl; }

}

- с конкретной спецификацией с подходящим стандартным исключением:

int Geron(int a, int b, int c) throw(invalid\_argument)

{

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

throw invalid\_argument("Wrong sides");

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

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

}

- спецификация с собственным реализованным исключением:

class NewExcept

{

string msg;

public:

NewExcept(const string& s) : msg(s) {}

const char\* message() const { return msg.c\_str(); }

};

int Geron(int a, int b, int c) throw(NewExcept)

{

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

throw NewExcept("Wrong sides");

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

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

}

int main()

{

int a, b, c;

cin >> a >> b >> c;

try { Geron(a, b, c); }

catch (NewExcept& e)

{

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

}

return 0;

}