Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

«Пермский национальный исследовательский политехнический университет» (ПНИПУ)

Электротехнический факультет

Кафедра информационных технологий и автоматизированных систем

Направление подготовки 09.03.01 «Информатика и вычислительная техника»

**Лабораторная работа по классам № 9**

**по дисциплине**

**«Основы алгоритмизации и программирования»**

**семестр** 2

Выполнил студент гр. ИВТ-21-1б

Балдин Алексей Игоревич

Проверил:

Яруллин Денис Владимирович

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

(оценка) (подпись)

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

(дата)

г. Пермь – 2022

**Цель**

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.

**Исходные данные**

Вариант 5:

(По способам 2 и 3)

Класс- контейнер ВЕКТОР с элементами типа int. Реализовать операции:

[] – доступа по индексу;

int() – определение размера вектора;

\* вектор – умножение элементов векторов a[i]\*b[i];

+ n – переход вправо к элементу с номером n .

**Разбор кода:**

В реализации 2: Суть вывода сообщений об ошибке и завершении программы – через пользовательский класс (Error) и через специальные блоки – try – который принимает условие, throw – который выделяет ошибку и catch – который выводит ошибку.

В реализации 3: Суть сохраняется, но в реализации вместо одного пользовательского класса – приходит иерархия пользовательских классов. Для каждого блока ошибки – свои сообщения – под каждый вид ошибки, создается отдельный класс, унаследованный от класса типа этой ошибки. Также работает через базовый класс (Error) и через специальные блоки – try – который принимает условие, throw – который выделяет ошибку и catch – который выводит ошибку.

Код программы:

**Для реализации 2:**

#include<iostream>

using namespace std;

int max\_size = 6;

class error

{

private:

string msg;

public:

error(string m)

{

msg = m;

}

void what()

{

cout << "Произошла ошибка: " << msg << endl;

}

};

class Vector

{

private:

friend istream& operator>>(istream& in, Vector& tmp);

friend ostream& operator<<(ostream& out, const Vector& tmp1);

int size;

int\* data;

public:

Vector()

{

size = 0;

data = 0;

}

Vector(int size)

{

if (size > max\_size)

{

throw error("Недопустимый размер вектора!");

}

this->size = size;

data = new int[size];

}

Vector(int size, int\* d)

{

if (size > max\_size)

{

throw error("Недопустимый размер вектора!");

}

this->size = size;

data = new int[size];

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

{

data[i] = d[i];

}

}

Vector(const Vector& other)

{

this->size = other.size;

data = new int[size];

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

{

data[i] = other.data[i];

}

}

~Vector()

{

}

int& operator () () { return size; }

Vector operator \*(const Vector& other)

{

Vector tmp(size);

int\* temp = new int[size];

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

{

temp[i] = data[i] \* other.data[i];

}

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

{

tmp.data[i] = temp[i];

}

delete[] temp;

delete[]data;

return tmp;

}

int& operator [](int index)

{

if (index >= 0 && index < size)

{

return data[index];

}

else

{

cout << endl;

throw error("Недопустимый индекс");

}

}

const Vector& operator = (const Vector& v)

{

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

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

size = v.size;

data = new int[size];

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

{

data[i] = v.data[i];

}

return \*this;

}

};

ostream& operator<<(ostream& out, const Vector& tmp1)

{

cout << "{ ";

for (int i = 0; i < max\_size; ++i)

{

out << tmp1.data[i] << " ";

}

cout << "}";

return out;

}

istream& operator>>(istream& in, Vector& tmp)

{

for (int i = 0; i < max\_size; ++i)

{

in >> tmp.data[i];

}

return in;

}

int main()

{

setlocale(LC\_ALL, "rus");

srand(time(0));

try

{

int k;

Vector a(max\_size);

cout << "Введите элементы вектора a: " << endl;

cin >> a;

cout << endl;

cout << "Вектор a: " << a << endl << endl;

cout << "Введите индекс для поиска элемента: ";

cin >> k;

cout << endl << "Число с выбранным индексом: " << a[k] << endl;

cout << endl << "Длина вектора a: " << a() << endl << endl;

Vector b(max\_size);

cout << "Введите элементы вектора b: " << endl;

cin >> b;

cout << endl;

cout << "Вектор b: " << b;

cout << endl;

cout << endl << "Длина вектора b: " << b() << endl;

Vector c = a \* b;

cout << endl << "Произведение векторов равно : " << c << endl;

cout << endl;

cout << "Введите номер элемента, к которому будет осуществлён переход: ";

int n;

cin >> n;

while (n > max\_size || n <= 0)

{

cout << "Номер элемента не может быть больше количества чисел/или равен нулю/или отрицательным " << endl;

cout << "Введите еще раз: ";

cin >> n;

cout << endl;

}

cout << endl;

int g = n;

n--;

cout << "Элемент под " << g << " номером: " << c[n];

cout << endl << endl;

}

catch (error ER)

{

ER.what();

}

return 0;

}

**Для реализации 3:**

#include<iostream>

#include <string>

using namespace std;

int max\_size = 6;

class Error

{

public:

virtual void what() {};

};

class NumberError :public Error //ошибка в индексе вектора

{

protected:

string msg;

public:

NumberError() { msg = "Ошибка номера: "; }

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

};

class IndexError :public Error //ошибка в индексе вектора

{

protected:

string msg;

public:

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

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

};

class SizeError :public Error //ошибка в размере вектора

{

protected:

string msg;

public:

SizeError() { msg = "Ошибка Размера: "; }

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

};

class MaxSizeError :public SizeError //превышение максимального размера

{

protected:

string msg\_;

public:

MaxSizeError()

{

SizeError();

msg\_ = "Недопустимый размер вектора!";

}

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

};

class IndexError1 :public IndexError //индекс меньше нуля

{

protected:

string msg\_;

public:

IndexError1()

{

IndexError();

msg\_ = "Индекс меньше нуля! ";

}

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

};

class IndexError2 :public IndexError //индекс больше текущего размера вектора

{

protected:

string msg\_;

public:

IndexError2()

{

IndexError();

msg\_ = "Индекс больше размера вектора ";

}

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

};

class NumberError1 :public NumberError

{

protected:

string msg\_;

public:

NumberError1()

{

NumberError();

msg\_ = "Номер элемента не может быть больше количества чисел ";

}

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

};

class NumberError2 :public NumberError

{

protected:

string msg\_;

public:

NumberError2()

{

NumberError();

msg\_ = "Номер элемента не может быть отрицательным ";

}

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

};

class NumberError3 :public NumberError

{

protected:

string msg\_;

public:

NumberError3()

{

NumberError();

msg\_ = "Номер элемента не может быть равен нулю ";

}

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

};

class Vector

{

private:

friend istream& operator>>(istream& in, Vector& tmp);

friend ostream& operator<<(ostream& out, const Vector& tmp1);

int size;

int\* data;

public:

Vector()

{

size = 0;

data = 0;

}

Vector(int size)

{

if (size > max\_size)

{

throw MaxSizeError();

}

this->size = size;

data = new int[size];

}

Vector(int size, int\* d)

{

this->size = size;

data = new int[size];

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

{

data[i] = d[i];

}

}

Vector(const Vector& other)

{

this->size = other.size;

data = new int[size];

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

{

data[i] = other.data[i];

}

}

~Vector()

{

}

int& operator () () { return size; }

Vector operator \*(const Vector& other)

{

Vector tmp(size);

int\* temp = new int[size];

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

{

temp[i] = data[i] \* other.data[i];

}

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

{

tmp.data[i] = temp[i];

}

delete[] temp;

delete[]data;

return tmp;

}

int& operator [](int index)

{

if (index < 0) throw IndexError1();

if (index >= size) throw IndexError2();

return data[index];

}

const Vector& operator = (const Vector& v)

{

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

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

size = v.size;

data = new int[size];

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

{

data[i] = v.data[i];

}

return \*this;

}

};

ostream& operator<<(ostream& out, const Vector& tmp1)

{

cout << "{ ";

for (int i = 0; i < max\_size; ++i)

{

out << tmp1.data[i] << " ";

}

cout << "}";

return out;

}

istream& operator>>(istream& in, Vector& tmp)

{

for (int i = 0; i < max\_size; ++i)

{

in >> tmp.data[i];

}

return in;

}

int main()

{

setlocale(LC\_ALL, "rus");

srand(time(0));

try

{

int k;

Vector a(max\_size);

cout << "Введите элементы вектора a: " << endl;

cin >> a;

cout << endl;

cout << "Вектор a: " << a << endl << endl;

cout << "Введите индекс для поиска элемента: ";

cin >> k;

cout << endl << "Число с выбранным индексом: " << a[k] << endl;

cout << endl << "Длина вектора a: " << a() << endl << endl;

Vector b(max\_size);

cout << "Введите элементы вектора b: " << endl;

cin >> b;

cout << endl;

cout << "Вектор b: " << b;

cout << endl;

cout << endl << "Длина вектора b: " << b() << endl;

Vector c = a \* b;

cout << endl << "Произведение векторов равно : " << c << endl;

cout << endl;

cout << "Введите номер элемента, к которому будет осуществлён переход: ";

int n;

cin >> n;

cout << endl;

if (n > max\_size)

{

throw NumberError1();

cout << endl;

}

else if (n < 0)

{

throw NumberError2();

cout << endl;

}

else if (n == 0)

{

throw NumberError3();

cout << endl;

}

int g = n;

n--;

cout << "Элемент под " << g << " номером: " << c[n];

cout << endl << endl;

}

catch (Error &ER)

{

ER.what();

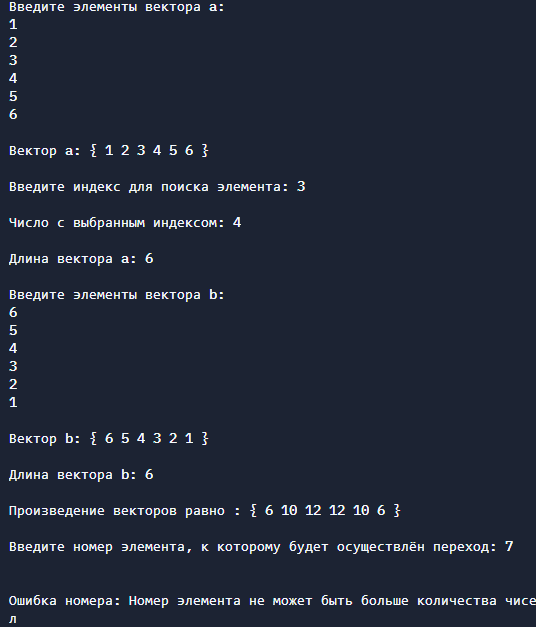
}

return 0;

}

Результат программы:

(По методу 3)



(По методу 2)

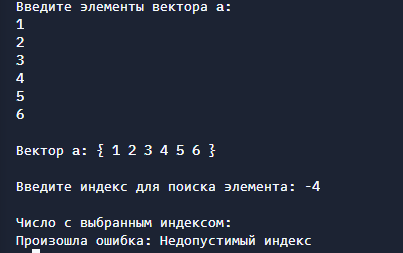


Диаграмма к задаче:

По методу 2:

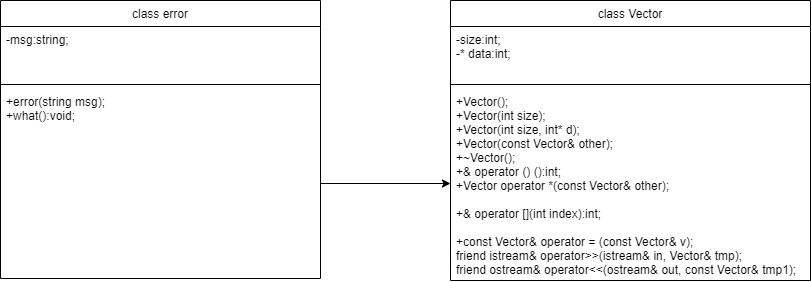


Рисунок 1

По методу 3:

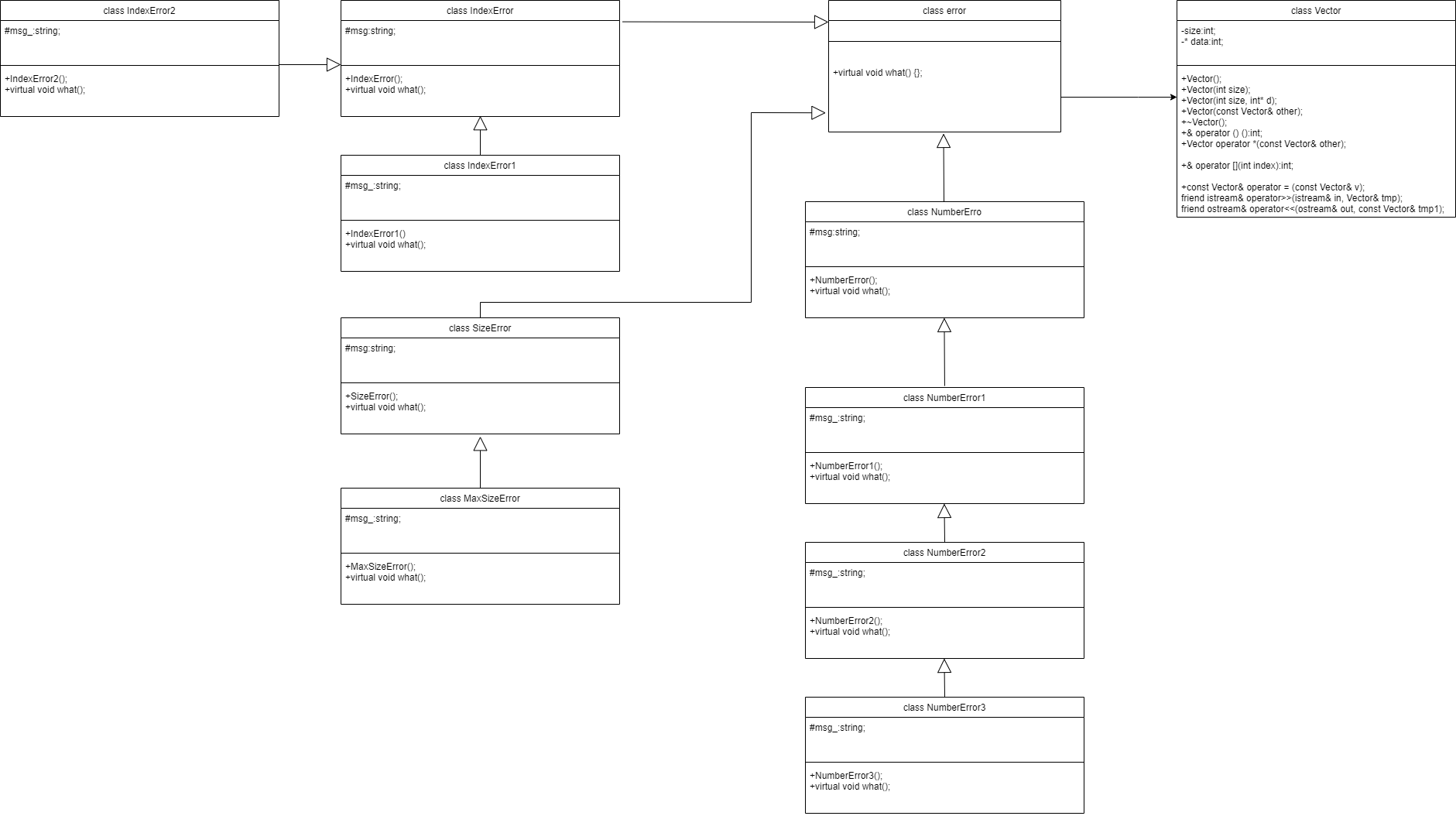


Рисунок 2

**Контрольные вопросы:**

1) то представляет собой исключение в С++?

2) На какие части исключения позволяют разделить вычислительный процесс? 3) 3) Достоинства такого подхода?

4) Какой оператор используется для генерации исключительной ситуации?

5) Что представляет собой контролируемый блок? Для чего он нужен?

6) Что представляет собой секция-ловушка? Для чего она нужна?

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

8) Каким образом можно создать собственную иерархию исключений?

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

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

11) В какой части программы может генерироваться исключение?

**Ответы на вопросы**

1)  Понятие исключения. В языке C++ исключение – это специальный объект класса или значение базового типа, который описывает (определяет) конкретную исключительную ситуацию и соответствующим образом обрабатывается.

2) Исключения позволяют логически разделить вычислительный процесс на две части. — обнаружение аварийной ситуации и ее обработка. Это важно не только для лучшей структуризации программы. ... Раскручиванием стека называется процесс освобождения памяти из-под локальных переменных и возврата управления вызывающей функции. Когда функция завершается, происходит естественное раскручивание стека. Тот же самый механизм используется и при обработке исключений.

3) Для их возбуждения используется оператор throw. Тип выражения, указанного в операторе throw, определяет тип исключительной ситуации, а значение может быть передано обработчику прерываний. Этот механизм, заявленный как стандартный, представляется весьма экзотическим без использования механизма классов.

4) Для перехвата исключений в списке инициализации конструктора. Контролируемый блок-функция является телом функции, а не находится внутри другой функции как обычный контролируемый блок.

Контролируемый блок идентифицируется ключевым словом try. Каждый обработчик исключении начинается со служебного слова catch. Общая схема размещения указанных блоков ... Если в контролируемом блоке формируется исключение, то делается попытка найти среди последующих обработчиков соответствующий исключению обработчик и передать ему управление.

5)Секция-ловушка с многоточием — это крайняя мера: если уж мы в нее попали, то в программе произошло что-то совсем непредусмотренное. Поэтому в такой секции обычно выводят сообщение о непредвиденном исключении и завершают работу программы.

6) 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, определенный 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 еще не были обработаны.

7) Корневой класс исключения в стандартной иерархии называется exception. Он определен в стандартном заголовочном файле и является базовым для всех исключений, возбуждаемых функциями из стандартной библиотеки. Класс exception имеет следующий интерфейс: namespace std. {class exception. public: exception() throw(); exception( const exception & ) throw()

8) При создании собственных исключений заканчивайте имя класса пользовательского исключения словом Exception и реализуйте три общих конструктора.

9) int и double

10) Всё подряд

11) Эти исключения генерируются не в произвольной точке программы, а в точке, где они указаны как throw («Ошибка»)