Министерство высшего 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»

ОТЧЁТ

Лабораторная работа №9

Обработка исключительных ситуаций

Выполнил

Студент группы РИС-22-2б

Вековшинин Д. А.

Проверила

Доцент кафедры ИТАС

Полякова О. А.

Пермь 2023

Постановка задачи

1. Реализовать класс, перегрузить для него операции, указанные в варианте.

2. Определить исключительные ситуации.

3. Предусмотреть генерацию исключительных ситуаций

Вариант 10: Класс- контейнер ВЕКТОР с элементами типа int.

Реализовать операции:

[] – доступа по индексу;

() – определение размера вектора; - – разность векторов;

-- – удаление элемента из вектора.

Словесное описание исключительных ситуаций.

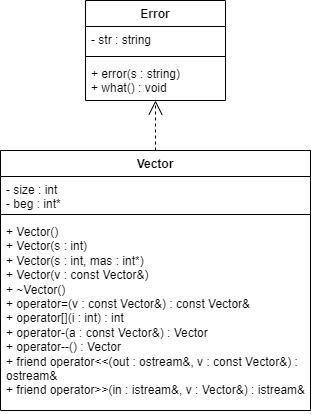
Исключительные ситуации генерируются:

1 – в конструкторе с параметром при попытке создать вектор больше максимального размера;

2, 3 – в операции [] при попытке обратиться к элементу с номером меньше 0 или больше текущего размера вектора;

4 – в операции -- при попытке удалить элемент из пустого вектора.

UML-диаграмма



Определение класса Error

class error //класс ошибка

{

string str;

public:

//конструктор, инициирует атрибут str сообщением об ошибке

error(string s) { str = s; }

void what() { cout << str << endl; } //выводит значение атрибута str

};

Описание класса-контейнера

class Vector

{

int size;

int\* beg;

public:

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

Vector(int s);

Vector(int s, int\* mas);

Vector(const Vector& v);

~Vector();

const Vector& operator=(const Vector& v);

int operator[](int i);

Vector operator-(const Vector& a);

Vector operator--();

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

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

};

Определение компонентных функций для класса-контейнера.

Vector::Vector(int s)

{

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

size = s;

beg = new int[s];

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

beg[i] = 0;

}

Vector::Vector(const Vector& v)

{

size = v.size;

beg = new int[size];

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

beg[i] = v.beg[i];

}

Vector::~Vector()

{

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

}

Vector::Vector(int s, int\* mas)

{

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

size = s;

beg = new int[size];

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

beg[i] = mas[i];

}

const Vector& Vector::operator =(const Vector& v)

{

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

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

size = v.size;

beg = new int[size];

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

beg[i] = v.beg[i];

return\*this;

}

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

{

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

else

{

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

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

out << endl;

}

return out;

}

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

{

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

{

cout << ">";

in >> v.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-(const Vector& a)

{

if (a.size > size) throw error("Second vector is empty");

Vector temp(size);

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

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

return temp;

}

Vector Vector::operator --()

{

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

if (size == 1)

{

size = 0;

delete[]beg;

beg = 0;

return \*this;

};

Vector temp(size, beg);

delete[]beg;

size--;

beg = new int[size];

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

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

return\*this;

}

Функция main()

int main()

{

try

{

Vector z;

cout << "Size of vector?" << endl;

int size;

cin >> size;

Vector x(size);

cout << "Enter " << size << " elements of vector X" << endl;

cin >> x;

cout << "X: " << x;

cout << "Nomer?" << endl;

int i;

cin >> i;

cout << x[i] << endl;

Vector y(size);

cout << "Enter " << size << " elements of vector Y" << endl;

cin >> y;

cout << "Y: " << y;

cout << "Nomer?" << endl;

int j;

cin >> j;

cout << y[j] << endl;

z = x - y;

cout << "Z: " << z << endl;

while(true)

{

cout << "X: " << x << endl;

--x;

}

}

catch (error e)

{

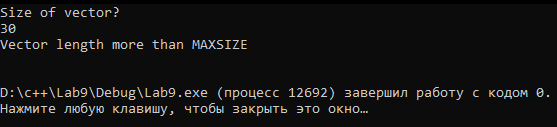
e.what();

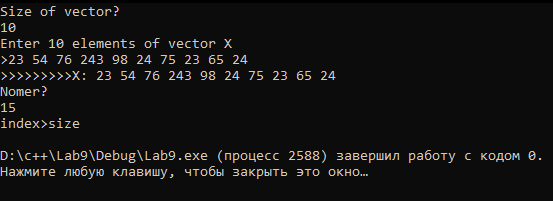
}

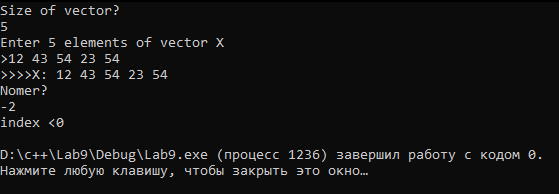
return 0;

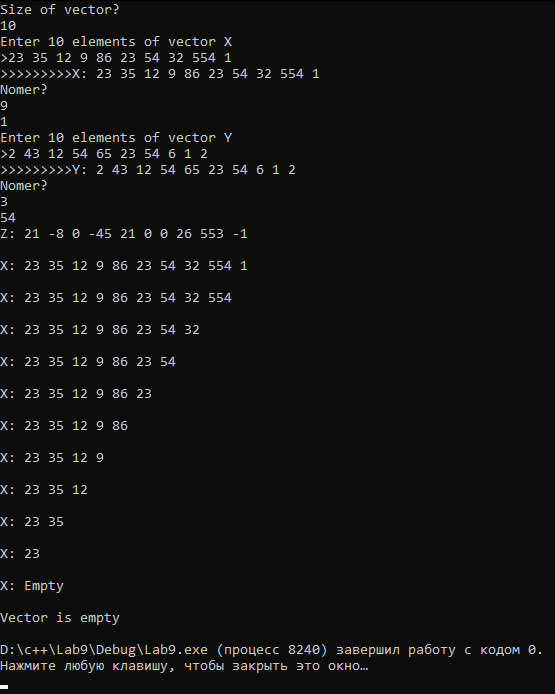
}

Результаты работы программы









Ответы на контрольные вопросы

1. Что представляет собой исключение в С++?

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).

Достоинства такого подхода:

1) удобно использовать в программе, которая состоит из нескольких модулей;

2) не требуется возвращать значение в вызывающую функцию

1. Какой оператор используется для генерации исключительной ситуации?

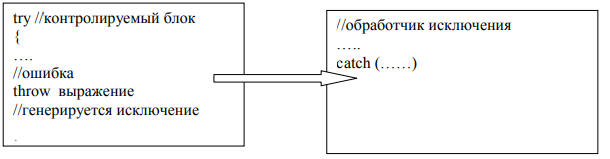
Исключение генерируется оператором

throw <выражение>, где <выражение> -

* либо константа,
* либо переменная некоторого типа,
* либо выражение некоторого типа

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

Контролируемый блок try{}, с которым связана одна или несколько секций-ловушек catch, используется для проверки возникновения исключения.



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

catch( спецификация исключения )

Секция-ловушка – это блок для обработки исключения.

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

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.

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

Exception

Класс exception определен 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 объявить свой базовый класс- исключение, 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.

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 порождать функция f1()?

Функция может вызвать исключения типов int и double

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

Функция не вызывает исключений.

1. В какой части программы может генерироваться исключение?

В контролируемом блоке try

try

{

Vector z;

cout << "Size of vector?" << endl;

int size;

cin >> size;

Vector x(size);

cout << "Enter " << size << " elements of vector X" << endl;

cin >> x;

cout << "X: " << x;

cout << "Nomer?" << endl;

int i;

cin >> i;

cout << x[i] << endl;

Vector y(size);

cout << "Enter " << size << " elements of vector Y" << endl;

cin >> y;

cout << "Y: " << y;

cout << "Nomer?" << endl;

int j;

cin >> j;

cout << y[j] << endl;

z = x - y;

cout << "Z: " << z << endl;

while(true)

{

cout << "X: " << x << endl;

--x;

}

}

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.

1)

double heron(double a, double b, double c)

{

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

double s = sqrt((p \* (p - a)) \* (p \* (p - b)) \* (p \* (p - c)));

return s;

}

2)

double heron(double a, double b, double c)

{

if (a == 0 || b == 0 || c == 0) throw;

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

double s = sqrt((p \* (p - a)) \* (p \* (p - b)) \* (p \* (p - c)));

return s;

}

3)

double heron(double a, double b, double c)

{

if (a == 0 || b == 0 || c == 0) throw exception("Ни одна из сторон не может быть равна 0");

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

double s = sqrt((p \* (p - a)) \* (p \* (p - b)) \* (p \* (p - c)));

return s;

}

4)

class error //класс ошибка

{

string str;

public:

//конструктор, инициирует атрибут str сообщением об ошибке

error(string s) { str = s; }

void what() { cout << str << endl; } //выводит значение атрибута str

};

double heron(double a, double b, double c)

{

if (a == 0 || b == 0 || c == 0) throw error("Ни одна из сторон не может быть равна 0");

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

double s = sqrt((p \* (p - a)) \* (p \* (p - b)) \* (p \* (p - c)));

return s;

}