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: «Информатика»

Тема: Программа управляемая событиями

Семестр 2

Выполнил работу

Студент группы РИС-22-1Б

Тарасов C.В.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

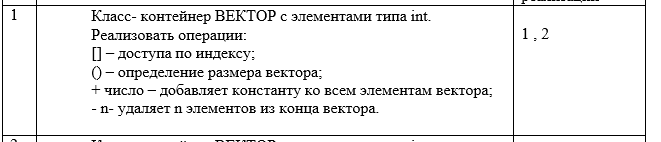
Г. Пермь-2023

**Постановка задачи (общая и конкретного варианта).**

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

Определить исключительные ситуации.

Предусмотреть генерацию исключительных ситуаций.



**Словесное описание исключительных ситуаций.**

Исключительные ситуации генерируются:

1 – в операции [] при попытке ввести i<0 и i>= размера массива

2 – в операции – при попытке удаления элемента из пустого вектора и при попытке удаления n элементов из вектора, размер которого size<n

**Определение класса error или иерархии пользовательских классов для определения исключительных ситуаций (если есть необходимость).**

class error //класс ошибка

{

string str;

public:

//конструктор, инициирует атрибут str сообщением об ошибке

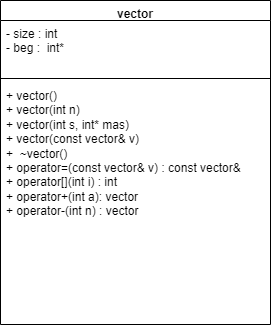
error(string s){str=s;}

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

};

**1 реализация**

**Описание класса-контейнера.**

****

class vector

{

int size;

int\* beg;

public:

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

vector(int n);

vector(int s, int\* mas);

vector(const vector& v);

~vector();

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

int operator[](int i);

vector operator+(int a);

vector operator-(int n);

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

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

**Определение компонентных функций для класса-контейнера.**

vector::vector(int n)

{

size = n;

beg = new int[n];

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

{

beg[i] = 0;

}

}

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

{

size = s;

beg = new int[size];

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

beg[i] = mas[i];

}

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;

}

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;

}

int vector::operator[](int i)

{

if (i < 0)throw 2;

if (i >= size) throw 3;

return beg[i];

}

vector vector::operator+(int a)

{

if (size != 0)

{

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

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

}

return \*this;

}

//-n - удаляет n элементов из конца вектора.

//5 – в операции – при попытке удалить элемент из пустого вектора.

vector vector::operator-(int n)

{

if (size == 0) throw 4;

else if (size < n)

{

/\*size = 0;

delete[] beg;

beg = 0;\*/

throw 5;

}

else if (size == n)

{

size = 0;

delete[] beg;

beg = 0;

return \*this;

}

else if (size>n){

size = size - n;

return \*this;

}

}

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

{

if (v.size == 0) out << "Пусто."<<endl;

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++)

{

in >> v.beg[i];

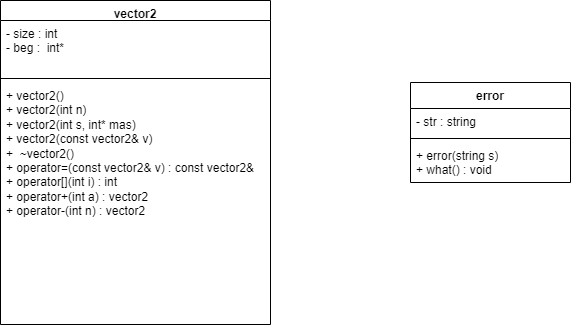
}

return in;

}

**2 реализация**

**Описание класса-контейнера.**

****

class vector2

{

int size;

int\* beg;

public:

vector2() { size = 0; beg = 0; };

vector2(int n);

vector2(int s, int\* mas);

vector2(const vector2& v);

~vector2();

const vector2& operator=(const vector2& v);

int operator[](int i);

vector2 operator+(int a);

vector2 operator-(int n);

friend ostream& operator<<(ostream& out, const vector2& v);

friend istream& operator>>(istream& in, vector2& v);

};

**Определение компонентных функций для класса-контейнера.**

vector2::vector2(int n)

{

size = n;

beg = new int[n];

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

{

beg[i] = 0;

}

}

vector2::vector2(int s, int\* mas)

{

size = s;

beg = new int[size];

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

beg[i] = mas[i];

}

vector2::vector2(const vector2& v)

{

size = v.size;

beg = new int[size];

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

{

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

}

}

vector2::~vector2()

{

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

}

const vector2& vector2::operator=(const vector2& 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;

}

int vector2::operator[](int i)

{

if (i < 0)throw error("ERROR. [i] , i<0 ");

if (i >= size) throw error("ERROR. [i] , i>=size ");

return beg[i];

}

vector2 vector2::operator+(int a)

{

if (size != 0)

{

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

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

}

return \*this;

}

//-n - удаляет n элементов из конца вектора.

//5 – в операции – при попытке удалить элемент из пустого вектора.

vector2 vector2::operator-(int n)

{

if (size == 0) throw error("ERROR. -n , size==0 ");

else if (size < n)

{

/\*size = 0;

delete[] beg;

beg = 0;\*/

throw error("ERROR. - n, size < n ");

}

else if (size == n)

{

size = 0;

delete[] beg;

beg = 0;

return \*this;

}

else if (size > n) {

size = size - n;

return \*this;

}

}

ostream& operator<<(ostream& out, const vector2& v)

{

if (v.size == 0) out << "Пусто." << endl;

else

{

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

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

out << endl;

}

return out;

}

istream& operator>>(istream& in, vector2& v)

{

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

{

in >> v.beg[i];

}

return in;

}

**Функция main().**

void main()

{

setlocale(LC\_ALL, "Russian");

//1 вариант реализации

try

{

vector x(5);//вектор из двух элементов

cout<<x;//печать вектора х

cout<<"Nomer? ";

int i; cin >> i;

//вывод элемента с номером i, если номер больше размера вектора или меньше 0, то error

cout << x[i] << endl;

x + 4;

cout << x[i] << endl;

cout << x;

//если вектор пустой или размер вектора меньше чем число удаляемых из него элементов, то error

x-3;

cout << x;

}

//обработчик исключения

catch (int a)

{

if (a == 2) cout << "ERROR. [i] , i<0 ";

if (a == 3) cout << "ERROR. [i] , i>=size ";

if (a == 4) cout << "ERROR. -n , size==0 ";

if (a == 5) cout << "ERROR. -n , size< n ";

}

system("pause");

system("cls");

//2 вариант реализации

try

{

vector2 y(5);//вектор из двух элементов

cout << y;//печать вектора х

cout << "Nomer?";

int j; cin >> j;

//вывод элемента с номером i, если номер больше размера вектора или меньше 0, то error

cout << y[j] << endl;

y + 4;

cout << y[j] << endl;

cout << y;

//если вектор пустой или размер вектора меньше чем число удаляемых из него элементов, то error

y - 3;

cout << y;

}

//обработчик исключения

catch (error e)

{

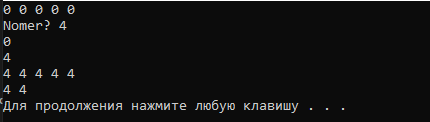
e.what();

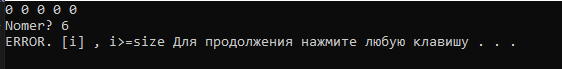
}

}

**Объяснение результатов работы программы.**

**Первая реализация**

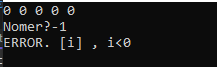






**Вторая реализация**





**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.

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

Исключения позволяют разделить вычислительный процесс на 2 части:

● Обнаружение аварийной ситуации (неизвестно как обрабатывать);

● Обработка аварийной ситуации (неизвестно, где она возникла).

Достоинства такого подхода:

● Удобно использовать в программе, которая состоит из нескольких модулей;

● Не требуется возвращать значение в вызывающую функцию.

Общая схема:

Try

{

Throw

}

Catch(…)

3. 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{};

Генерация исключения будет выглядеть:

throw ZeroDevide();//вызывается конструктор без параметров

или

throw new ZeroDevide();

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

Проверка возникновения исключения делается с помощью оператора try, с которым неразрывно связаны одна или несколько блоков обработки исключений — catch. Оператор try объявляет в любом месте программы контролируемый блок, который имеет следующий вид:

try { /\* контролируемый блок \*/ }

Контролируемый блок, помимо функции контроля, обладает функциями обычного блока: все переменные, объявленные внутри него, являются локальными в этом блоке и не видны вне его.

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

После блока try обязательно прописывается один или несколько блоков catch. Блок кода после catch предложения является обработчиком исключений. Это обработчик, который перехватывает исключение, которое возникает, если типы в throw выражениях и catch совместимы. Форма записи секции-ловушки следующая:

catch (спецификация\_параметра\_исключения) { /\* блок обработки \*/}

6. 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.

Форма 1 означает, что объект передается в блок обработки, чтобы его каким-то образом там использовать, например, для вывода информации в сообщении об ошибке. Примеры:

● catch( exception e) // по значению

● catch( exception &e) // по ссылке

● catch( const exception &e) // по константной ссылке

● catch( exception \*e) //по указателю

Лучше всего передавать объект по ссылке, т. к. при этом не создается временный объект-исключение.

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

В составе стандартной библиотеки С++ реализован ряд стандартных исключений, которые организованы в иерархию классов.

Эта иерархия может служить основой для создания собственных классов исключений и иерархии исключений. Можно определять собственные исключения, унаследовав их от класса exception.

8. 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.

9. Если спецификация исключений имеет вид:

void f1()throw(int,double);

то какие исключения может прождать функция f1()?

Для каждой функции, метода, конструктора или деструктора можно в заголовке указать спецификацию исключений. Если в заголовке спецификация исключений не указана, считается, что функция может порождать любое исключение, если указана, то считается, что функция генерирует те исключения, которые явно указаны в этом списке.

Следовательно, функция f1() может генерировать исключения типа int и double.

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

Если спецификация имеет вид такой вид, то считается, что функция исключений не генерирует.

11. В какой части программы может генерироваться исключение?

Исключение могут генерируется внутри оператора try { }.

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));

}