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: «Основы алгоритмизации и программирования»

Тема: «Обработка исключительных ситуаций»

Семестр 2

Выполнил работу

Студент группы ИВТ-22-2б

Устюгова Полина Дмитриевна

Проверил

Доцент кафедры ИТАС

Полякова Ольга Андреевна

Г. Пермь-2023

**Постановка задачи**

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

**Описание класса**

Класс- контейнер СПИСОК с ключевыми значениями типа int. Реализовать операции:

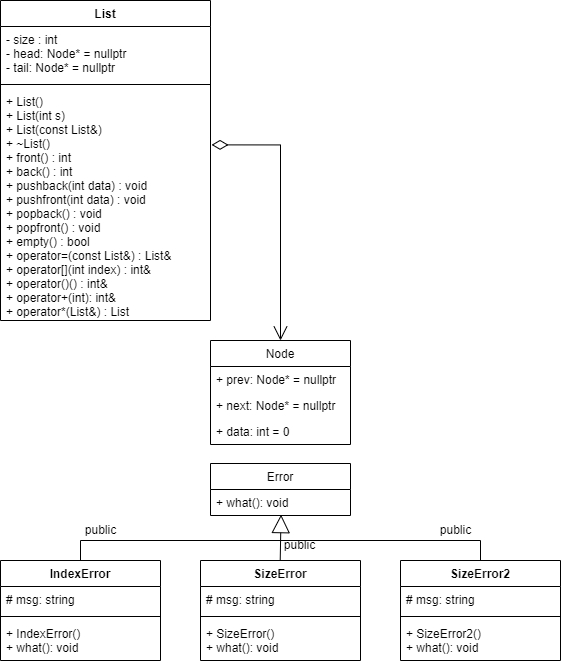
[] – доступа по индексу;

int() – определение размера списка;

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

Пользовательский класс Pair (пара чисел). Пара должна быть представлено двумя полями: типа int для первого числа и типа double для второго. Первое число при выводе на экран должно быть отделено от второго числа двоеточием.

**UML - Диаграмма**



**Рис. 1 – UML-диаграмма классов List, Node, Error, IndexError, SizeError, SizeError2**

**Контрольные вопросы**

1. Что представляет собой исключение в С++?

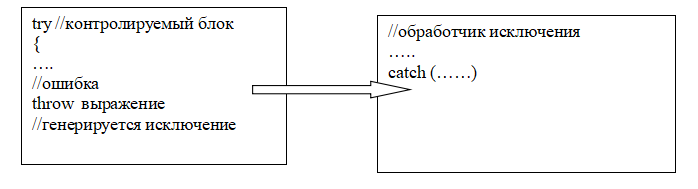
Исключение – это непредвиденное или аварийное событие.

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;

}