Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра информационных технологий и автоматизированных систем

**ОТЧЕТ**

**о работе по информатике**

Семестр: 2

На тему: «ООП. Обработка исключительных ситуаций»

Выполнил студент ИВТ-22-2б:

Казанцев Антон Васильевич

(дата, подпись)

Проверила:

Полякова Ольга Андреевна

(дата, подпись)

Пермь 2023

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

*Вариант 15:*

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];

+n - переход вправо к элементу с номером n.

Вариант реализации: 1,3

**Исключительные ситуации**

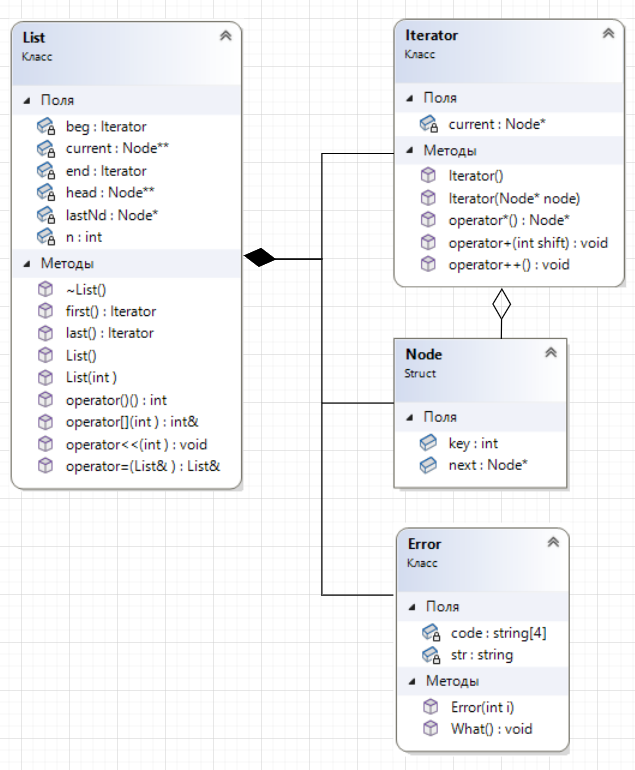
Исключительные ситуации генерируются:

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;

В остальных случаях – exception.

**UML-Диаграмма**



**Программное решение**

***Error.h***

#pragma once

#pragma once

#include <string>

#include <iostream>

using namespace std;

class Error {

private:

string code[4] = { "Ошибка: нельзя сместиться, выход за пределы списка!", "Ошибка: невозможно получить первый элемент пустого списка!", "Ошибка: невозможно получить последний элемент пустого списка!", "Ошибка: Индекс за пределами диапазона!" };

string str;

public:

Error(int i) {

str = code[i];

}

void What() {

cout << str << '\n';

}

};

***List.h***

#pragma once

#include <string>

#include <iostream>

using namespace std;

#include "Error.h"

struct Node {

int key;

Node\* next;

};

class Iterator {

private:

Node\* current;

friend class List;

public:

Iterator() {

current = NULL;

};

Iterator(Node\* node) {

current = node;

};

void operator ++ () {

if (current == NULL) {

throw Error(0);

}

current = current->next;

}

Node\* operator \*() const {

return current;

}

void operator + (int shift) {

Node\* tmp = current;

int i = 0;

while (i < shift && current != NULL && current->next != NULL) {

current = current->next;

i++;

}

if (i < shift) {

current = tmp;

throw Error(0);

}

}

};

class List {

public:

List() {};

List(int);

~List();

List& operator = (List&);

int& operator[] (int);

int operator () ();

friend ostream& operator << (ostream&, List&);

friend istream& operator >> (istream&, List&);

Iterator first();

Iterator last();

void operator << (int);

private:

Node\* lastNd, \* current, \* head;

int n = 0;

Iterator beg, end;

};

***List.cpp***

#include "List.h"

List::List(int count) {

n = count;

head = new Node;

head->key = 0;

lastNd = head;

for (int i = 1; i < n; i++) {

current = new Node;

current->key = 0;

lastNd->next = current;

lastNd = current;

}

lastNd->next = NULL;

}

List::~List() {

lastNd = head;

while (lastNd != NULL) {

current = lastNd->next;

delete lastNd;

lastNd = current;

}

n = 0;

}

List& List::operator=(List& l) {

if (this != &l) {

if (this != 0) {

lastNd = head;

while (lastNd != NULL) {

current = lastNd->next;

delete lastNd;

lastNd = current;

}

n = 0;

}

lastNd = head = new Node;

l.lastNd = l.head->next;/////

lastNd->key = l.head->key;

while (l.lastNd != NULL) {

lastNd->next = new Node;

lastNd->next->key = l.lastNd->key;

l.lastNd = l.lastNd->next;

lastNd = lastNd->next;

}

lastNd->next = NULL;

n = l.n;

}

return \*this;

}

int& List::operator[](int index) {

if (index >= n) {

throw Error(3);

}

lastNd = head;

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

lastNd = lastNd->next;

}

return lastNd->key;

}

int List::operator () () {

return n;

}

Iterator List::first() {

if (!n) {

throw Error(1);

}

beg.current = head;

return beg;

}

Iterator List::last() {

if (!n) {

throw Error(2);

}

lastNd = head;

while (lastNd->next != NULL) {

lastNd = lastNd->next;

}

end.current = lastNd;

return end;

}

ostream& operator<<(ostream& out, List& l) {

if (l.n) {

l.lastNd = l.head;

while (l.lastNd != NULL) {

out << l.lastNd->key << ' ';

l.lastNd = l.lastNd->next;

}

}

else {

out << "Список пуст!";

}

return out;

}

istream& operator>>(istream& in, List& l) {

l.lastNd = l.head;

while (l.lastNd != NULL) {

in >> l.lastNd->key;

l.lastNd = l.lastNd->next;

}

return in;

}

void List::operator << (int number) {

if (head == NULL) {

head = new Node;

head->key = number;

head->next = NULL;

}

else {

lastNd = head;

while (lastNd->next != NULL) {

lastNd = lastNd->next;

}

lastNd->next = new Node;

lastNd->next->key = number;

lastNd = lastNd->next;

lastNd->next = NULL;

}

n += 1;

}

***LabaOOP9.cpp***

#include <iostream>

using namespace std;

#include "List.h"

#include "Error.h"

int main() {

system("chcp 1251 > NULL");

try {

List a(2);

List b;

cout << a << '\n';

cout << b << '\n';

cout << b[0];//Ошибка

Iterator i = b.first();//Ошибка

Iterator j = b.last();//Ошибка

Iterator k = a.last();

++k;

++k;//Ошибка

k + 1;//Ошибка

}

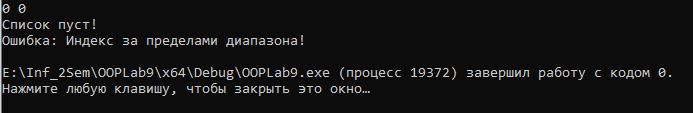
catch (Error& m\_Error) {

m\_Error.What();

}

return 0;

}



**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?*
2. Обнаружение аварийной ситуации (неизвестно, как обрабатывать).
3. Обработка аварийной ситуации (неизвестно, где возникла).

*Достоинства:*

1. Удобно использовать в программе, которая состоит из нескольких модулей.
2. Не требуется возвращать значение в вызывающую функцию.
3. *Какой оператор используется для генерации исключительной ситуации?*

throw <выражение>, где <выражение> либо константа, либо переменная некоторого типа, либо выражение некоторого типа.

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

Служебное слово try позволяет выделить в любом месте исполняемого текста программы так называемый контролируемый блок. Блок try используется для проверки возникновения исключения, соответственно в блок try помещается та часть кода, в которой возможно возникновение исключения.

try { операторы }

*Свойства блока try*:

- Связан с одним или несколькими операторами catch.

- Все переменные объявленные внутри try являются локальными.

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

После блока try обязательно прописывается один или несколько блоков catch, которые обычно называют обработчиками исключений, или секциями-ловушками. Форма записи секции-ловушки следующая:

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

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

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

1. (тип имя)
2. (тип)
3. (…)

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?*

Базовым классом для всех исключений является класс Exception, соответственно для создания своих типов можно унаследовать данный класс.

1. *Каким образом можно создать собственную иерархию исключений?*

Для создания собственной иерархии исключений надо объявить свой базовый класс исключение, например: class Error{};

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

class Child\_Error: public Error{};

class Parents\_Error: public Error{};

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

Только исключения типа int и double.

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

Исключения абсолютно любого типа.

1. *В какой части программы может генерироваться исключение?*

Исключение генерируется в той части кода, где используется контролируемый блок try(). Однако исключение не локализуется в блоке, где использован оператор его генерации. Исключение как объект возникает в точке генерации, распознается в контролируемом блоке и передается в обработчик исключений. Только после обработки оно может исчезнуть.

1. Написать функцию, которая вычисляет площадь треугольника по трем сторонам (формула Герона).

*Без спецификации исключений:*

int geron(int a, int b, int c) {

int S, P;

P = (a + b + c) / 2;

S = sqrt(P\*(P-a)\* P \* (P - b)\* P \* (P - c));

return S;

}

*Со спецификатором throw:*

double geron(int a, int b, int c) {

if (a \* b \* c < 0) throw 1;

double P;

double S;

P = ((a + b + c) / 2);

S = sqrt(P\*(P-a)\* P \* (P - b)\* P \* (P - c));

return S;

}

*C собственным реализованным исключением:*

void F() {

exit(1);

}

void set\_terminate() {

F();

}

double geron(int a, int b, int c) {

if (a \* b \* c < 0) set\_terminate();

double P;

double S;

P = ((a + b + c) / 2);

S = sqrt(P\*(P-a)\* P \* (P - b)\* P \* (P - c));

return S;

}

*С конкретной спецификацией с подходящим стандартным исключением:*

double geron(int a, int b, int c) {

if (a \* b \* c < 0) terminate();

double P;

double S;

P = ((a + b + c) / 2);

S = sqrt(P\*(P-a)\* P \* (P - b)\* P \* (P - c));

return S;

}