ПНИПУ

Федеральное государственное автономное образовательное учреждение высшего образования «Пермский национальный исследовательский политехнический университет»

Лабораторная работа №9.

Обработка исключительных ситуаций.

Выполнил студент группы РИС-23-3Б

Буковский Денис Владимирович

Проверила доцент кафедры ИТАС О.А. Полякова

2024

1. Постановка задачи

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

2. Определить исключительные ситуации.

3. Предусмотреть генерацию исключительных ситуаций.

Класс- контейнер СПИСОК с ключевыми значениями типа int.

Реализовать операции:

[] - доступа по индексу;

int() — определение размера списка;

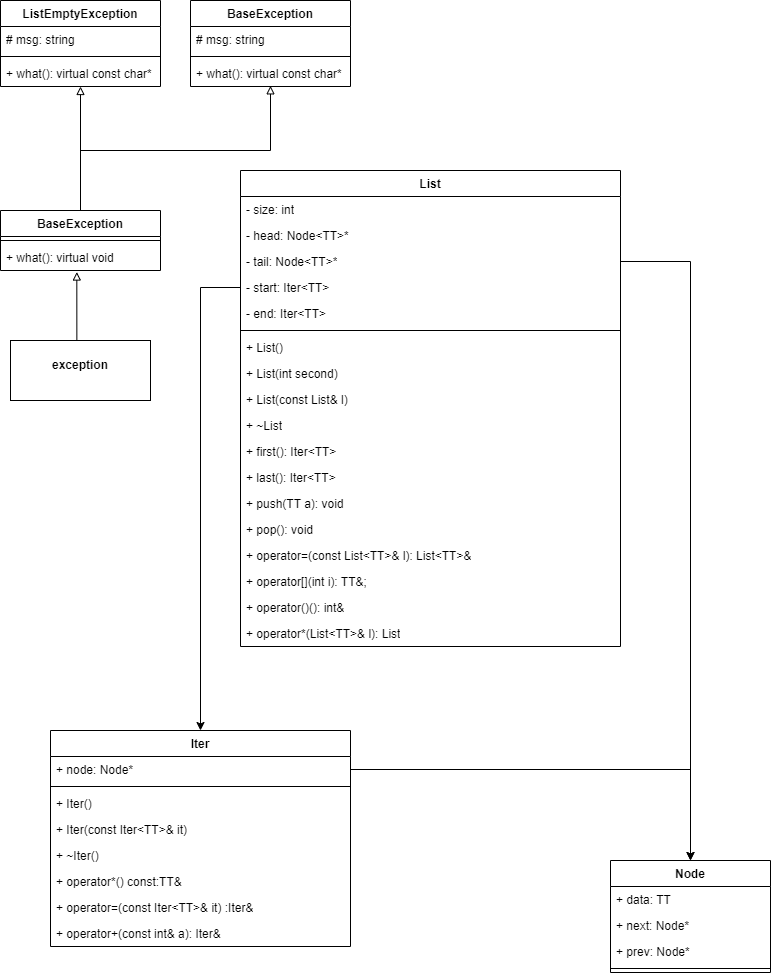
\* вектор — умножение элементов списков a[i]\*b[i];

+n - переход вправо к элементу с номером n.

1. Анализ задачи

Возьмём текст программы List.h из прошлой работы. Напишем классы содержащие исключения. Добавим в определение функций в List.h обработку ошибок. Протестируем в функции main().

1. Блок-схема



1. Код

ListExceptions.h

#pragma once

#include <iostream>

class BaseException : public std::exception {

public:

virtual const char\* what() = 0;

};

class ListEmptyException : public BaseException

{

protected:

std::string msg;

public:

ListEmptyException() { this->msg = "Empty list error! You are probably trying to delete an element from empty list!\n"; }

virtual const char\* what(){

std::cout << msg;

return "runtime\_error";

}

};

class ListIndexException : public BaseException

{

protected:

std::string msg;

public:

ListIndexException() { this->msg = "List index is bigger than its size, or less than 0."; }

virtual const char\* what() {

std::cout << msg;

return "runtime\_error";

}

};

List.h

#pragma once

#include <iostream>

#include "ListExceptions.h"

template <typename T>

struct Node {

T data;

Node\* next = nullptr;

Node\* prev = nullptr;

};

template <typename T>

class Iter

{

public:

Node<T>\* node;

Iter();

Iter(const Iter<T>\*);

~Iter();

Iter<T>& operator=(const Iter<T>&);

T& operator\*() const;

Iter& operator+(const int&);

};

template <typename T>

class List {

private:

int size;

Node<T>\* head;

Node<T>\* tail;

Iter<T> start;

Iter<T> end;

public:

List();

List(int);

List(const List<T>&);

~List();

Iter<T> first();

Iter<T> last();

void push(T);

void pop();

List& operator=(const List<T>&);

T& operator[](int);

int& operator()();

List<T> operator\*(List<T>&);

};

template <typename T>

List<T>::List() {

this->size = 0;

this->head = nullptr;

this->tail = nullptr;

}

template <typename T>

List<T>::List(int count) {

this->size = count;

Node<T>\* new\_node = new Node<T>;

this->head = new\_node;

this->tail = new\_node;

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

new\_node = new Node<T>;

this->tail->next = new\_node;

new\_node->prev = this->tail;

this->tail = new\_node;

}

this->start.node = this->head;

this->end.node = this->tail;

}

template <typename T>

List<T>::List(const List<T>& lst) {

Node<T>\* node = lst.head;

while (node != nullptr) {

push(node->data);

node = node->next;

}

this->start.node = this->head;

this->end.node = this->tail;

this->size = lst.size;

}

template <typename T>

List<T>::~List() {

Node<T>\* node = this->head;

while (node != nullptr) {

this->head = node->next;

delete node;

node = this->head;

}

}

template <typename T>

List<T>& List<T>::operator=(const List& lst) {

Node<T>\* node = lst.head;

while (node != nullptr) {

push(node->data);

node = node->next;

}

this->start.node = this->head;

this->end.node = this->tail;

this->size = lst.size;

}

template <typename T>

int& List<T>::operator()() {

return this->size;

}

template <typename T>

T& List<T>::operator[](int index) {

if ((index >= this->size) || (index < 0)) throw ListIndexException();

Node<T>\* node = this->head;

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

node = node->next;

}

return node->data;

}

template <typename T>

List<T> List<T>::operator\*(List<T>& lst) {

if (this->size == 0 || lst.size == 0) throw ListEmptyException();

List<T>\* base = \*this;

int new\_size = this->size;

List<T> tmp;

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

tmp.push(base[i] \* lst[i]);

}

return tmp;

}

template <typename T>

void List<T>::push(T value) {

Node<T>\* new\_node = new Node<T>;

new\_node->data = value;

if (this->size == 0) {

new\_node->next = nullptr;

new\_node->prev = nullptr;

this->head = new\_node;

this->tail = new\_node;

this->size++;

this->start.node = head;

this->end.node = tail;

}

else {

this->tail->next = new\_node;

new\_node->prev = this->tail;

this->tail = new\_node;

this->size++;

this->end.node = tail;

}

}

template <typename T>

void List<T>::pop() {

if (this->size > 0) {

Node<T>\* node = this->tail;

this->tail = node->prev;

delete node;

this->tail->next = nullptr;

this->size--;

this->end.node = this->tail;

}

else throw ListEmptyException();

}

template <typename T>

Iter<T> List<T>::first() {

if (this->size <= 0) throw ListEmptyException();

return this->start;

}

template <typename T>

Iter<T> List<T>::last() {

if (this->size <= 0) throw ListEmptyException();

return this.end;

}

template <typename T>

Iter<T>::Iter() {

this->node = nullptr;

}

template <typename T>

Iter<T>::Iter(const Iter<T>\* item) {

this->node = item->node;

}

template <typename T>

Iter<T>::~Iter() {

this->node = nullptr;

}

template <typename T>

Iter<T>& Iter<T>::operator=(const Iter<T>& item) {

this->node = item.node;

return \*this;

}

template <typename T>

T& Iter<T>::operator\*() const {

return this->node->data;

}

template <typename T>

Iter<T>& Iter<T>::operator+(const int& index) {

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

this->node = this->node->next;

}

return \*this;

}

Main.cpp

#include <iostream>

#include "List.h"

#include "ListExceptions.h"

using namespace std;

int main()

{

try {

int count, in\_value, index;

List<int> list1;

Iter<int> iter;

cout << "Enter quantity of elements: ";

cin >> count;

cout << endl;

cout << "Enter elements:" << endl;

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

{

cin >> in\_value;

list1.push(in\_value);

}

cout << endl;

iter = list1.first();

cout << "Enter element's index: ";

cin >> index;

cout << list1[index] << endl;

cout << endl;

cout << "Enter number to move for from first element: ";

cin >> index;

iter + index;

cout << \*iter << endl;

cout << endl;

}

catch (BaseException &e) {

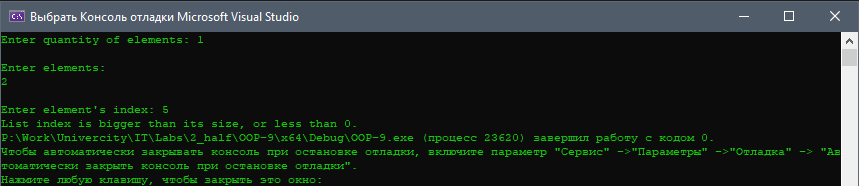
e.what();

}

return 0;

}

1. Вывод программы



1. Ответы на контрольные вопросы

Механизм обработки ошибок

Обнаружение исключения и его обработка

throw

Блок, в котором происходит проверка на наличие исключения

Секция-ловушка передаёт код ошибки обработчику

(Тип, имя); (Тип) ; (...)

<expection>

Создать свой базовый класс исключений

logic\_error, runtime\_error, bad\_cast

logic\_error, runtime\_error

Исключение может генерироваться в той части программы, которая

обнаружила аномальную ситуацию

#include <iostream>

using namespace std;

class BasicError {

public:

BasicError() {}

virtual ~BasicError() {}

void PrintError() {

cout << "Error!" << endl;

}

};

int main()

{

while (true) {

try

{

double a, b, c, P, S;

cin >> a >> b >> c;

if (a < 0 || b < 0 || c < 0 || a + b < c || b + c < a || a + c < b)

{

throw BasicError();

}

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

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

cout << S;

}

catch (BasicError e) {

e.PrintError();

continue;

}

}

}