Федеральное государственное автономное образовательное учреждение высшего образования «Пермский национальный исследовательский

политехнический университет»

Лабораторная работа №9

«Обработка исключительных ситуаций»

Выполнил:

студент первого курса

ЭТФ группы РИС-23-3б

Коротаев Александр Дмитриевич

Проверила:

Доцент кафедры ИТАС О. А. Полякова

Пермь, 2024

Обработка исключительных ситуаций

**Цель задания**

1. Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.
2. Разработка программы, обрабатывающей исключительные

ситуации.

**Постановка задачи**

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

варианте.

1. Определить исключительные ситуации.
2. Предусмотреть генерацию исключительных ситуаций.

**Задание**

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.

1. ,

**Анализ задачи 1**

1. Для реализации класса список необходимо создать структуру

Node.

1. В классе список необходимо реализовать конструкторы, деструктор, функции вставки/ удаления первого/последнего элемента, а также операторы перегрузки.
2. Операции последовательного доступа можно реализовать

посредством создания класса итератора. В классе должны быть

конструкторы, деструктор, а также перегрузки операций.

[] – доступа по индексу;

int() – определение размера списка;

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

+n - переход вправо к элементу с номером n ( с помощью класса-итератора).

1. В классе Iterator необходимо указать, что класс List –

дружественный.

1. Исключительные ситуации генерируются:

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

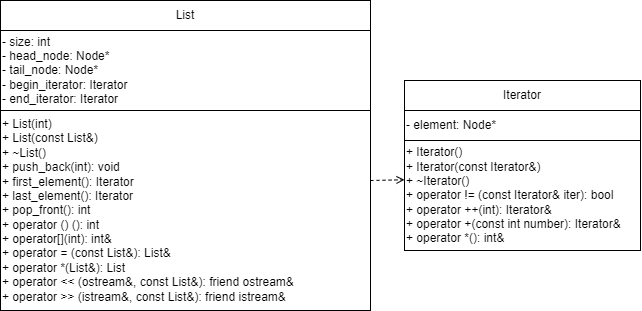
2, 3 – в операции [] – при попытке обратиться к элементу с номером меньше 0 или больше текущего размера вектора;

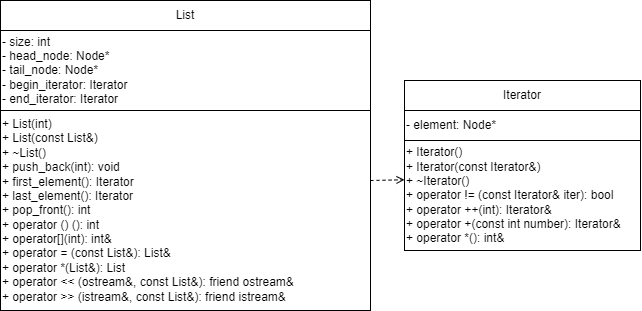
– в операции + – при попытке добавить элемент с номером больше максимального размера;

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

1. Информация об исключительных ситуациях передается с помощью стандартного типа данных.

**UML диаграмма**

****



**Код**

Файл ООП9\_1.cpp

#include <iostream>

using namespace std;

#include "list.h"

#include "iterator.h"

void list\_rand(List& list);

void print\_iterator(List& list);

void msg\_error(int& x)

{

cout << "Ошибка с кодом " << x << "\n";

switch (x)

{

case 1:

cout << "Недопустимый индекс списка! Отрицательный индекс недопустим.\n";

break;

case 2:

cout << "Недопустимый индекс списка! Индекс больше текущего размера списка.\n";

break;

case 3:

cout << "Недопустимый размер списка! Отрицательный рамер недопустим.\n";

break;

case 4:

cout << "Недопустимый размер списка! Превышен максимальный размер.\n";

break;

default:

break;

}

cout << "\n";

}

int main()

{

system("chcp 1251 > Null");

srand(time(0));

List list\_1(10);

cout << "list\_1\n";

cout << list\_1 << "\n\n";

list\_rand(list\_1);

cout << "list\_1: " << list\_1 << "\n\n";

cout << "Первый элемент: " << \*(list\_1.first\_element()) << "\n";

cout << "Последний элемент: " << \*(list\_1.last\_element()) << "\n\n";

List list\_2(list\_1);

cout << "list\_2\n";

cout << list\_2 << "\n\n";

try

{

int size;

cout << "Размер list\_3: "; cin >> size;

List list\_3(size);

cout << "list\_3\n";

cin >> list\_3;

cout << list\_3 << "\n\n";

int index;

cout << "Элемент с каким индексом вывести: "; cin >> index;

cout << "Элемент: " << list\_3[index] << "\n\n";

List list\_4(15);

list\_rand(list\_4);

cout << "list\_4\n";

cout << list\_4 << "\n\n";

List list\_5 = list\_3 \* list\_4;

cout << "list\_5 = list\_3 \* list\_4\n";

cout << list\_5 << "\n\n";

}

catch (int& x)

{

msg\_error(x);

}

return 0;

}

void list\_rand(List& list)

{

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

{

list[i] = rand() % (100 + 1) - 50;

}

}

void print\_iterator(List& list)

{

for (Iterator iter = list.first\_element(); iter != list.last\_element() + 1; iter++)

{

cout << \*iter << ' ';

}

}

Файл list.h

#pragma once

#include <iostream>

#include "iterator.h"

struct Node

{

int data;

Node\* ptr\_to\_next\_node = nullptr;

};

class List

{

private:

int size;

Node\* head\_node;

Node\* tail\_node;

Iterator begin\_iterator;

Iterator end\_iterator;

public:

List(int);

List(const List&);

~List();

void push\_back(int);

Iterator first\_element();

Iterator last\_element();

void pop\_front();

int operator () ();

int& operator[](int);

List& operator = (const List&);

List operator \*(List&);

friend std::ostream& operator << (std::ostream&, const List&);

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

};

Файл list.cpp

#pragma once

#include "list.h"

#include "iterator.h"

List::List(int size)

{

const int Max\_size = 30;

if (size < 0) throw 3;

if (size > Max\_size) throw 4;

if (size > 0)

{

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

{

this->push\_back(0);

}

}

else

{

this->head\_node = nullptr;

this->tail\_node = nullptr;

}

this->begin\_iterator.element = this->head\_node;

this->end\_iterator.element = this->tail\_node;

}

List::List(const List& list)

{

this->size = 0;

Node\* current\_node = list.head\_node;

while (current\_node != nullptr)

{

push\_back(current\_node->data);

current\_node = current\_node->ptr\_to\_next\_node;

}

this->begin\_iterator.element = this->head\_node;

this->end\_iterator.element = this->tail\_node;

}

List::~List()

{

while (head\_node != nullptr)

{

this->pop\_front();

}

}

void List::push\_back(int data)

{

Node\* New\_Node = new Node;

New\_Node->data = data;

if (this->head\_node == nullptr)

{

this->head\_node = New\_Node;

this->tail\_node = New\_Node;

this->begin\_iterator.element = this->head\_node;

this->end\_iterator.element = this->tail\_node;

}

else

{

this->tail\_node->ptr\_to\_next\_node = New\_Node;

this->tail\_node = New\_Node;

this->end\_iterator.element = this->tail\_node;

}

this->size++;

}

Iterator List::first\_element()

{

return this->begin\_iterator;

}

Iterator List::last\_element()

{

return this->end\_iterator;

}

void List::pop\_front()

{

int temp;

if (this->tail\_node != nullptr)

{

Node\* current\_node = this->head\_node;

head\_node = current\_node->ptr\_to\_next\_node;

temp = current\_node->data;

this->size--;

this->begin\_iterator.element = head\_node;

}

}

int List::operator () ()

{

return this->size;

}

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

{

if (index < 0) throw 1;

if (index > size - 1) throw 2;

Node\* current\_node = this->head\_node;

for (int i = 0; i != index; i++)

{

current\_node = current\_node->ptr\_to\_next\_node;

}

return current\_node->data;

}

List& List::operator = (const List& list)

{

if (this != &list)

{

while (head\_node != nullptr)

{

this->pop\_front();

}

size = 0;

Node\* current\_node = list.head\_node;

while (current\_node != nullptr)

{

push\_back(current\_node->data);

current\_node = current\_node->ptr\_to\_next\_node;

}

}

return \*this;

}

List List::operator \*(List& list)

{

int temp\_size;

if (this->size > list.size)

{

temp\_size = list.size;

}

else

{

temp\_size = this->size;

}

List temp\_list(temp\_size);

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

{

temp\_list[i] = (\*this)[i] \* list[i];

}

return temp\_list;

}

std::ostream& operator <<(std::ostream& stream, const List& list)

{

std::cout << "Вывод элементов списка: \n";

Node\* current\_node = list.head\_node;

while (current\_node != nullptr)

{

stream << current\_node->data << ' ';

current\_node = current\_node->ptr\_to\_next\_node;

}

return stream;

}

std::istream& operator >>(std::istream& stream, List& list)

{

std::cout << "Введите элементы списка: ";

Node\* current\_node = list.head\_node;

while (current\_node != nullptr)

{

stream >> current\_node->data;

current\_node = current\_node->ptr\_to\_next\_node;

}

return stream;

}

Файл iterator.h

#pragma once

#include "list.h"

class Iterator

{

private:

friend struct Node;

friend class List;

Node\* element;

public:

Iterator();

Iterator(const Iterator&);

~Iterator() {}

bool operator != (const Iterator&);

Iterator& operator ++(int);

Iterator& operator +(const int);

int& operator \*();

};

Файл iterator.cpp

#pragma once

#include "list.h"

#include "iterator.h"

Iterator::Iterator()

{

this->element = nullptr;

}

Iterator::Iterator(const Iterator& iterator)

{

this->element = iterator.element;

}

bool Iterator::operator!=(const Iterator& iterator)

{

return this->element != iterator.element;

}

Iterator& Iterator::operator++(int)

{

this->element = this->element->ptr\_to\_next\_node;

return \*this;

}

Iterator& Iterator::operator+(const int number)

{

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

{

this->element = this->element->ptr\_to\_next\_node;

}

return \*this;

}

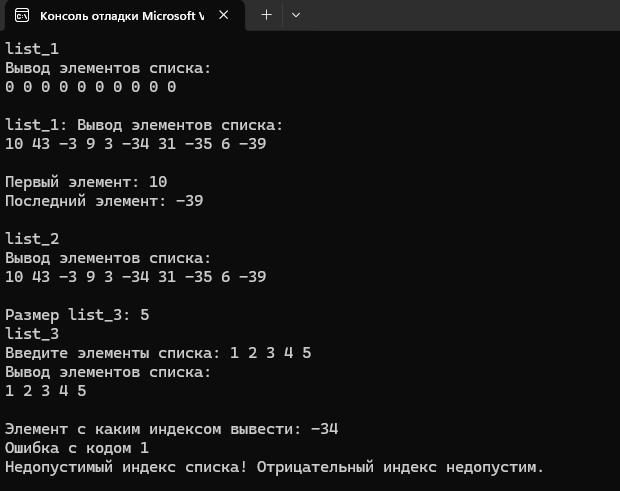
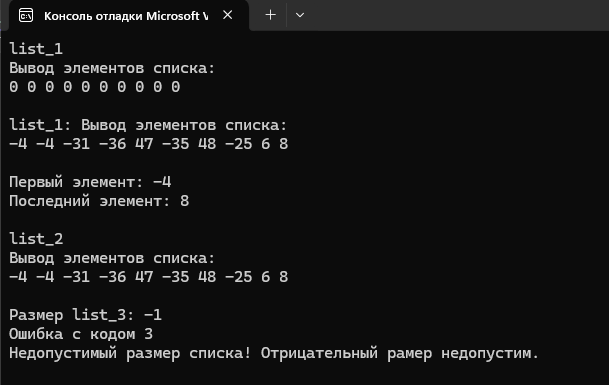
int& Iterator::operator\*()

{

return this->element->data;

}

**Результаты работы**



**Анализ задачи 2**

1. Для реализации класса список необходимо создать структуру

Node.

1. В классе список необходимо реализовать конструкторы,

деструктор, функции вставки/ удаления первого/последнего элемента, а также операторы перегрузки.

1. Операции последовательного доступа можно реализовать посредством создания класса итератора. В классе должны быть конструкторы, деструктор, а также перегрузки операций.

[] – доступа по индексу;

int() – определение размера списка;

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

+n - переход вправо к элементу с номером n ( с помощью класса-итератора).

1. В классе Iterator необходимо указать, что класс List –

дружественный.

1. Исключительные ситуации генерируются:

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

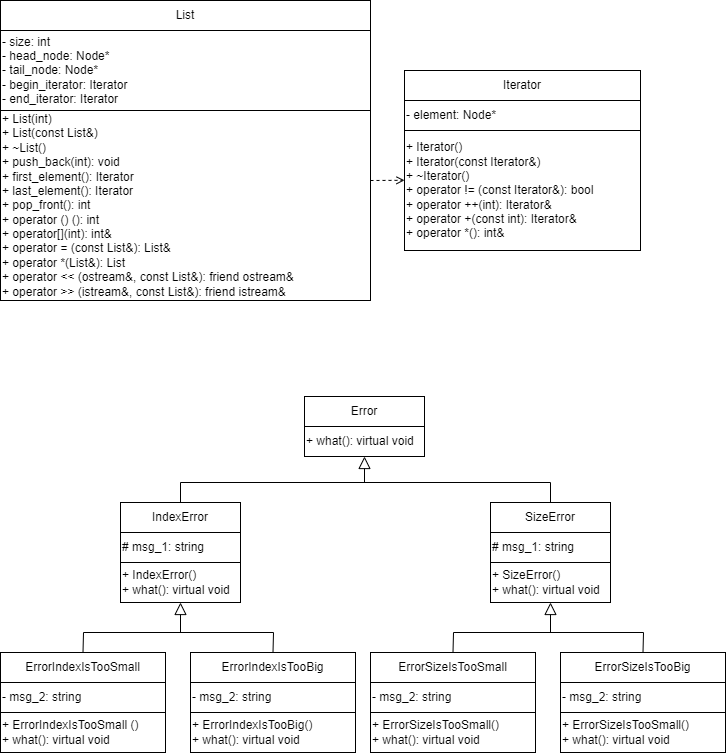
2, 3 – в операции [] – при попытке обратиться к элементу с номером меньше 0 или больше текущего размера вектора;

– в операции + – при попытке добавить элемент с номером больше максимального размера;

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

1. Информация об исключительных ситуациях передается с помощью иерархии пользовательских классов.

**UML диаграмма**



**Код**

Файл ООП9\_3.cpp

#include <iostream>

#include <string>

using namespace std;

#include "list.h"

#include "iterator.h"

#include "error.h"

void list\_rand(List& list);

void print\_iterator(List& list);

int main()

{

system("chcp 1251 > Null");

srand(time(0));

List list\_1(10);

cout << "list\_1\n";

cout << list\_1 << "\n\n";

list\_rand(list\_1);

cout << "list\_1: " << list\_1 << "\n\n";

cout << "Первый элемент: " << \*(list\_1.first\_element()) << "\n";

cout << "Последний элемент: " << \*(list\_1.last\_element()) << "\n\n";

List list\_2(list\_1);

cout << "list\_2\n";

cout << list\_2 << "\n\n";

try

{

int size;

cout << "Размер list\_3: "; cin >> size;

List list\_3(size);

cout << "list\_3\n";

cin >> list\_3;

cout << list\_3 << "\n\n";

int index;

cout << "Элемент с каким индексом вывести: "; cin >> index;

cout << "Элемент: " << list\_3[index] << "\n\n";

List list\_4(15);

list\_rand(list\_4);

cout << "list\_4\n";

cout << list\_4 << "\n\n";

List list\_5 = list\_3 \* list\_4;

cout << "list\_5 = list\_3 \* list\_4\n";

cout << list\_5 << "\n\n";

}

catch (Error& error)

{

error.what();

}

return 0;

}

void list\_rand(List& list)

{

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

{

list[i] = rand() % (100 + 1) - 50;

}

}

void print\_iterator(List& list)

{

for (Iterator iter = list.first\_element(); iter != list.last\_element() + 1; iter++)

{

cout << \*iter << ' ';

}

}

Файл list.h

#pragma once

#include <iostream>

#include "iterator.h"

#include "error.h"

struct Node

{

int data;

Node\* ptr\_to\_next\_node = nullptr;

};

class List

{

private:

int size;

Node\* head\_node;

Node\* tail\_node;

Iterator begin\_iterator;

Iterator end\_iterator;

public:

List(int);

List(const List&);

~List();

void push\_back(int);

Iterator first\_element();

Iterator last\_element();

void pop\_front();

int operator () ();

int& operator[](int);

List& operator = (const List&);

List operator \*(List&);

friend std::ostream& operator << (std::ostream&, const List&);

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

};

Файл list.cpp

#pragma once

#include "list.h"

#include "iterator.h"

List::List(int size)

{

const int Max\_size = 30;

if (size < 0) throw ErrorSizeIsTooSmall();

if (size > Max\_size) throw ErrorSizeIsTooBig();

if (size > 0)

{

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

{

this->push\_back(0);

}

}

else

{

this->head\_node = nullptr;

this->tail\_node = nullptr;

}

this->begin\_iterator.element = this->head\_node;

this->end\_iterator.element = this->tail\_node;

}

List::List(const List& list)

{

this->size = 0;

Node\* current\_node = list.head\_node;

while (current\_node != nullptr)

{

push\_back(current\_node->data);

current\_node = current\_node->ptr\_to\_next\_node;

}

this->begin\_iterator.element = this->head\_node;

this->end\_iterator.element = this->tail\_node;

}

List::~List()

{

while (head\_node != nullptr)

{

this->pop\_front();

}

}

void List::push\_back(int data)

{

Node\* New\_Node = new Node;

New\_Node->data = data;

if (this->head\_node == nullptr)

{

this->head\_node = New\_Node;

this->tail\_node = New\_Node;

this->begin\_iterator.element = this->head\_node;

this->end\_iterator.element = this->tail\_node;

}

else

{

this->tail\_node->ptr\_to\_next\_node = New\_Node;

this->tail\_node = New\_Node;

this->end\_iterator.element = this->tail\_node;

}

this->size++;

}

Iterator List::first\_element()

{

return this->begin\_iterator;

}

Iterator List::last\_element()

{

return this->end\_iterator;

}

void List::pop\_front()

{

int temp;

if (this->tail\_node != nullptr)

{

Node\* current\_node = this->head\_node;

head\_node = current\_node->ptr\_to\_next\_node;

temp = current\_node->data;

this->size--;

this->begin\_iterator.element = head\_node;

}

}

int List::operator () ()

{

return this->size;

}

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

{

if (index < 0) throw ErrorIndexIsTooSmall();

if (index > size - 1) throw ErrorIndexIsTooBig();

Node\* current\_node = this->head\_node;

for (int i = 0; i != index; i++)

{

current\_node = current\_node->ptr\_to\_next\_node;

}

return current\_node->data;

}

List& List::operator = (const List& list)

{

if (this != &list)

{

while (head\_node != nullptr)

{

this->pop\_front();

}

size = 0;

Node\* current\_node = list.head\_node;

while (current\_node != nullptr)

{

push\_back(current\_node->data);

current\_node = current\_node->ptr\_to\_next\_node;

}

}

return \*this;

}

List List::operator \*(List& list)

{

int temp\_size;

if (this->size > list.size)

{

temp\_size = list.size;

}

else

{

temp\_size = this->size;

}

List temp\_list(temp\_size);

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

{

temp\_list[i] = (\*this)[i] \* list[i];

}

return temp\_list;

}

std::ostream& operator <<(std::ostream& stream, const List& list)

{

std::cout << "Вывод элементов списка: \n";

Node\* current\_node = list.head\_node;

while (current\_node != nullptr)

{

stream << current\_node->data << ' ';

current\_node = current\_node->ptr\_to\_next\_node;

}

return stream;

}

std::istream& operator >>(std::istream& stream, List& list)

{

std::cout << "Введите элементы списка: ";

Node\* current\_node = list.head\_node;

while (current\_node != nullptr)

{

stream >> current\_node->data;

current\_node = current\_node->ptr\_to\_next\_node;

}

return stream;

}

Файл iterator.h

#pragma once

#include "list.h"

class Iterator

{

private:

friend struct Node;

friend class List;

Node\* element;

public:

Iterator();

Iterator(const Iterator&);

~Iterator() {}

bool operator != (const Iterator&);

Iterator& operator ++(int);

Iterator& operator +(const int);

int& operator \*();

};

Файл iterator.cpp

#pragma once

#include "list.h"

#include "iterator.h"

Iterator::Iterator()

{

this->element = nullptr;

}

Iterator::Iterator(const Iterator& iterator)

{

this->element = iterator.element;

}

bool Iterator::operator!=(const Iterator& iterator)

{

return this->element != iterator.element;

}

Iterator& Iterator::operator++(int)

{

this->element = this->element->ptr\_to\_next\_node;

return \*this;

}

Iterator& Iterator::operator+(const int number)

{

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

{

if (this->element->ptr\_to\_next\_node == nullptr) throw ErrorIndexIsTooBig();

this->element = this->element->ptr\_to\_next\_node;

}

return \*this;

}

int& Iterator::operator\*()

{

return this->element->data;

}

Файл error.h

#pragma once

#include <iostream>

using namespace std;

class Error

{

public:

virtual void what() {};

};

class IndexError : public Error

{

protected:

string msg\_1;

public:

IndexError()

{

msg\_1 = "Недопустимый индекс списка.";

}

virtual void what()

{

cout << msg\_1 << "\n";

}

};

class ErrorIndexIsTooSmall : public IndexError

{

protected:

string msg\_2;

public:

ErrorIndexIsTooSmall()

{

IndexError();

msg\_2 = "Отрицательный индекс недопустим.";

}

virtual void what()

{

cout << msg\_1 << " " << msg\_2 << "\n\n";

}

};

class ErrorIndexIsTooBig : public IndexError

{

protected:

string msg\_2;

public:

ErrorIndexIsTooBig()

{

IndexError();

msg\_2 = "Индекс больше текущего размера списка.";

}

virtual void what() {

cout << msg\_1 << " " << msg\_2 << "\n\n";

}

};

class SizeError : public Error

{

protected:

string msg\_1;

public:

SizeError()

{

msg\_1 = "Недопустимый размер списка!";

}

virtual void what()

{

cout << msg\_1 << endl;

}

};

class ErrorSizeIsTooSmall : public SizeError

{

protected:

string msg\_2;

public:

ErrorSizeIsTooSmall()

{

IndexError();

msg\_2 = "Отрицательный рамер недопустим.";

}

virtual void what()

{

cout << msg\_1 << " " << msg\_2 << "\n\n";

}

};

class ErrorSizeIsTooBig : public SizeError

{

protected:

string msg\_2;

public:

ErrorSizeIsTooBig()

{

IndexError();

msg\_2 = "Превышен максимальный размер.";

}

virtual void what()

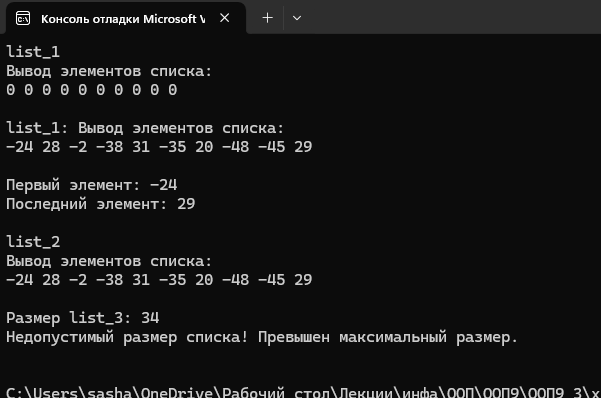
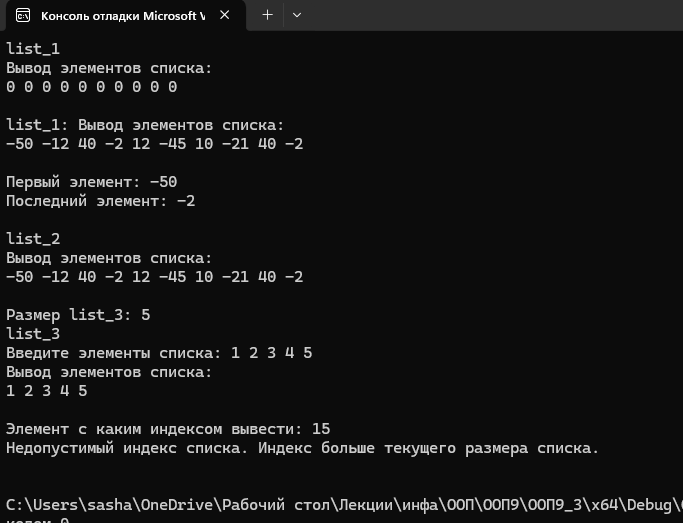
{

cout << msg\_1 << " " << msg\_2 << "\n\n";

}

};

**Результаты работы**

**GitHub**

https://github.com/Korovay4ik/Laboratory-works

**Контрольные вопросы**

1. **Что представляет собой исключение в С++?**

Исключение – это непредвиденное или аварийное событие.

**2.** **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)
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

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

**нужен?**

Контролируемый блок — это блок кода, в котором создаются локальные переменные, инициализирующие объекты, и при выходе из блока автоматически вызываются деструкторы объектов.

Контролируемый блок используется для гарантированного выполнения действий при выходе из блока, независимо от того, какой путь был выбран при выходе.

try {//недопустимый размер множества, отрицательный размер List list\_3(-1);

cout << "list\_3" << endl;

cin >> list\_3;

cout << list\_3 << endl << endl;

}

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

**нужна?**

Секция-ловушка (catch-блок) - это блок кода, который используется для

пере-хвата и обработки исключений, которые могут возникнуть в блоке try-

блоке.

catch (int& x) {

cout << "Ошибка с кодом " << x << endl;

msg\_error(x);

}

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

* catch без аргумента типа: catch (...). Эта форма используется для перехвата любого исключения, которое не было перехвачено ранее. Она может быть по-лезна, например, для записи сообщения в журнал перед завершением про-граммы.
  + catch с аргументом типа std::exception: catch (std::exception& e). Эта форма используется для перехвата исключений, производных от класса

std::exception. Она позволяет получить доступ к информации об исключении, например, к его сообщению, которое можно использовать для вывода пользователю.

* catch с аргументом типа T, где T - это класс исключения: catch (T& e).

Эта форма используется для перехвата конкретного класса исключения T. Она поз-воляет получить доступ к информации об исключении, которую можно ис-пользовать для его обработки.

* catch с несколькими аргументами типа: catch (T1& e1, T2& e2, ...). Эта форма используется для перехвата нескольких исключений разных типов. Она позво-ляет обработать каждое исключение по-разному в зависимости от его типа.

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

class Error

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

Для создания собственной иерархии исключений необходимо определить клас-сы исключений, которые будут наследоваться от стандартного класса Error или его производных классов.

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

Функция fl() может порождать исключения типа int и double

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

Функция fl() не может порождать никаких исключений. Это означает, что если в процессе выполнения функции возникнет исключение, то оно не будет обработано внутри этой функции и будет передано на уровень выше в стеке вызовов.

**11.** **В** **какой** **части** **программы** **может** **генерироваться**

**исключение?**

В любой.

**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));

}