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
Акбашева Софья Руслановна

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.
2. Определить исключительные ситуации.
3. 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.	3 , 1

Анализ задачи 1

1) Для реализации класса список необходимо создать структуру Node.

2) В классе список необходимо реализовать конструкторы, деструктор, функции вставки/ удаления первого/последнего элемента, а также операторы перегрузки.

3) 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).

4) В классе Iterator необходимо указать, что класс List – дружественный.

5) Исключительные ситуации генерируются:

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

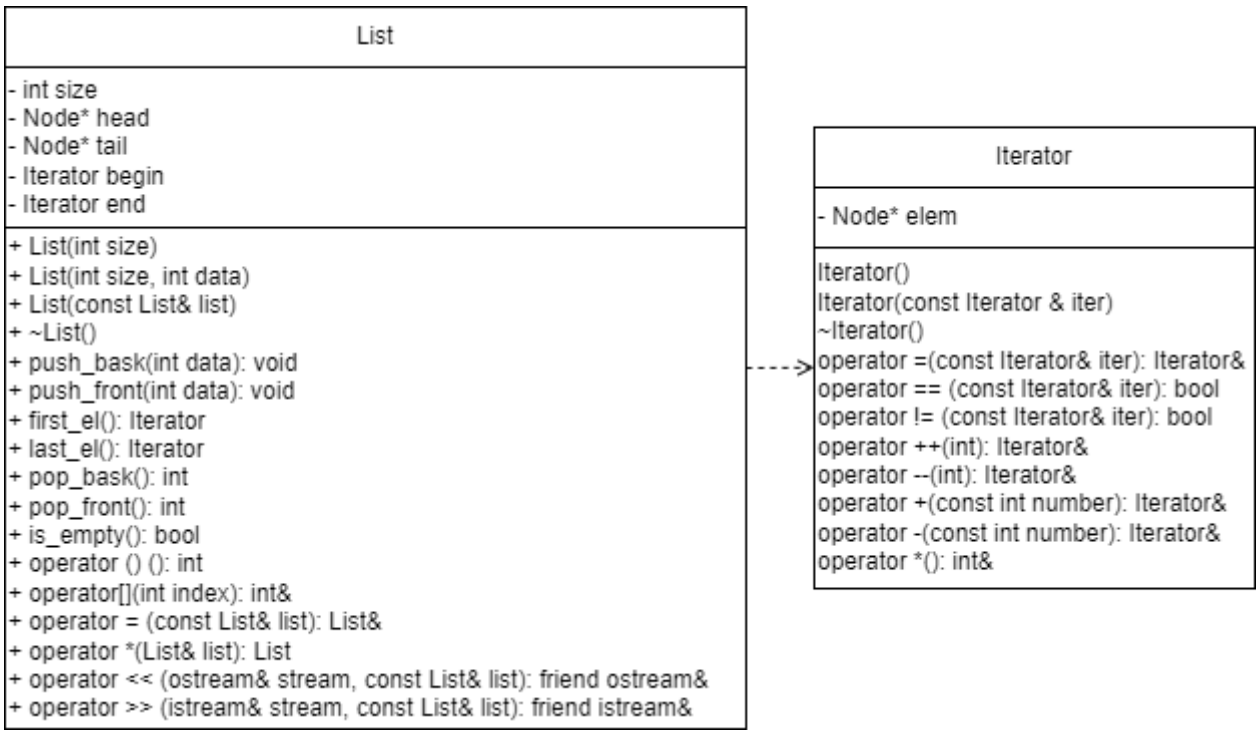
2, 3 – в операции [] – при попытке обратиться к элементу с номером меньше 0 или больше текущего размера вектора;

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

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

6) Информация об исключительных ситуациях передается с помощью стандартного типа данных.

UML диаграмма



Код

Файл List.h

```
#pragma once //предотвращает повторную загрузку заголовочного файла, если он уже был включен
#include <iostream> //стандартные потоки ввода и вывода
#include <string>
#include "Iterator.h"
using namespace std;

struct Node {
	int data; //данные
	Node* ptr_to_prev_node = nullptr; //указатель на предыдущий элемент
	Node* ptr_to_next_node = nullptr; //указатель на следующий элемент
};
```

```
class List {
private:
	int size; //размер
	Node* head; //указатель на голову
	Node* tail; //указатель на хвост
	Iterator begin;//указатель на начальный элемент
	Iterator end;//указатель на последний элемент
public:
	List(int size); //конструктор с длиной
	List(int size, int data);//конструктор с длиной и данными
	List(const List& list);//конструктор копирования
	~List();//деструктор

	void push_bask(int data);//вставка элемента в конец
	void push_front(int data);//вставка элемента в НАЧАЛО

	Iterator first_el();//последний элемент
	Iterator last_el();//первый элемент

	int pop_bask();//удалить последний элемент
	int pop_front();//удалить первый элемент

	bool is_empty();//список пустой иль нет

	int operator () ();//размер списка
	int& operator[](int index);//данные по индексу
	List& operator = (const List& list);//оператор присваивания
	List operator *(List& list);

	friend ostream& operator << (ostream& stream, const List& list);//ВЫВОД
	friend istream& operator >> (istream& stream, const List& list);//ВВОД
};
```

Файл List.cpp

```
#include "list.h"
using namespace std;

List::List(int size) {//конструктор с длиной
	if (size <= 0) throw 3;
	this->size = size;
	if (size > 0) {//если длина больше 0
		Node* node = new Node;//создаю узел
		this->head = node; //указатель на голову
		this->tail = node; //указатель на хвост
		for (int i = 1; i < size; i++) {//заполняю со второго номера
			Node* New_Node = new Node;//новый узел
			tail->ptr_to_next_node = New_Node;//хвост указывает на новый узел
			New_Node->ptr_to_prev_node = tail;//новый узел ук-т на хвост как на пред-й
			tail = New_Node;//новый узел становится хвостом
		}
		tail->ptr_to_next_node = nullptr;//до хвоста элементов нет
	}
	else {
		this->head = nullptr; //указатель на голову
		this->tail = nullptr; //указатель на хвост
	}
	this->begin.elem = this->head;
	this->end.elem = this->tail;
}

List::List(int size, int data) {//конструктор с длиной и данными
```

```
	if (size <= 0) throw 3;
	this->size = size;
	if (size > 0) {//если длина больше 0
		Node* node = new Node;//создаю узел
		node->data = data; //данные для узла
		this->head = node; //указатель на голову
		this->tail = node; //указатель на хвост
		for (int i = 1; i < size; i++) {//заполняю со второго номера
			Node* New_Node = new Node;//новый узел
			New_Node->data = data; //данные для нового узла
			tail->ptr_to_next_node = New_Node;//хвост указывает на новый узел
			New_Node->ptr_to_prev_node = tail;//новый узел ук-т на хвост как на пред-й
			tail = New_Node;//новый узел становится хвостом
		}
		tail->ptr_to_next_node = nullptr;//до хвоста элементов нет
	}
	else {
		this->head = nullptr; //указатель на голову
		this->tail = nullptr; //указатель на хвост
	}
	this->begin.elem = this->head;
	this->end.elem = this->tail;
}

List::List(const List& list) {//конструктор копирования
	this->head = nullptr; //указатель на голову
	this->tail = nullptr; //указатель на хвост
	this->size = 0;

	Node* current_node = list.head;//создаю узел
	while (current_node != nullptr) {//пока не пройду весь список
		push_bask(current_node->data);//добавляю в конец эл-т
		current_node = current_node->ptr_to_next_node;//беру следующий элемент
	}
	this->begin.elem = this->head;
	this->end.elem = this->tail;
}

List::~List() {//деструктор
	Node* current_node = head;//голова
	while (current_node != nullptr) {//пока не пройду весь список
		Node* next_node = current_node->ptr_to_next_node;//следующий элемент
		delete current_node;//удаляю текущий элемент
		current_node = next_node;//беру новый элемент
	}
	head = nullptr; //голова пустая
}

void List::push_bask(int data) {//вставка элемента в конец
	Node* New_Node = new Node;//новый узел
	New_Node->data = data; //данные для нового узла
	New_Node->ptr_to_next_node = nullptr;//новый узел пока никуда не указывает
	if (this->head == nullptr) {//если список пустой
		this->head = New_Node;//новый узел - голова
		this->tail = New_Node;//новый узел - хвост
		this->begin.elem = this->head;
		this->end.elem = this->tail;
	}
	else {
		tail->ptr_to_next_node = New_Node;//хвост указывает на узел
		New_Node->ptr_to_prev_node = tail;
		tail = New_Node; ////новый узел - хвост
```

```
                this->end.elem = this->tail;
        }
        this->size++;//увеличиваю длину
}

void List::push_front(int data) {//вставка элемента в НАЧАЛО
        Node* New_Node = new Node;//новый узел
        New_Node->data = data; //данные для нового узла

        if (this->head == nullptr) {//если список пустой
                this->head = New_Node;//новый узел - голова
                this->tail = New_Node;//новый узел - хвост
                this->begin.elem = this->head;
                this->end.elem = this->tail;
        }
        else {
                head->ptr_to_prev_node = New_Node;//голова указывает на узел
                New_Node->ptr_to_next_node = head;
                head = New_Node; //новый узел - голова
                this->begin.elem = this->head;
        }
        this->size++;//увеличиваю длину
}

Iterator List::first_el() {//первый элемент
        return this->begin;
}

Iterator List::last_el() {//первый элемент
        return this->end;
}

int List::pop_bask() {//удаление последнего
        int temp;
        if (this->tail != nullptr) {//если список не пустой
                Node* current_node = this->tail;//узел предыдущий
                tail = current_node->ptr_to_prev_node;//меняю хвост
                temp = current_node->data;
                tail->ptr_to_next_node = nullptr;
                this->size--;//уменьшаю размер
                this->end.elem = tail;
        }
        return temp;//значение из удаленной переменной
}

int List::pop_front() {//удаление первого
        int temp;
        if (this->tail != nullptr) {//если список не пустой
                Node* current_node = this->head;//узел предыдущий
                head = current_node->ptr_to_next_node;//меняю хвост
                temp = current_node->data;
                head->ptr_to_prev_node = nullptr;
                this->size--;//уменьшаю размер
                this->begin.elem = head;
        }
        return temp;//значение из удаленной переменной
}

bool List::is_empty() {
        return this->size == 0;//если пустой, то вернет 1 иначе 0
}
```

```
int List::operator () () {//размер списка
	return this->size;
}

int& List::operator[](int index) {
	if (index < this->size && index >= 0) {//если индекс меньше размера и больше -
		Node* current_node = this->head;
		for (int i = 0; i != index; i++) {
			current_node = current_node->ptr_to_next_node;//беру следующий элемент
		}
		return current_node->data;
	}
	else {
		if (index > this->size) throw 2;
		if (index < 0) throw 1;
		exit(0);
	}
}

List& List::operator = (const List& list) {
	if (this == &list) {//если элементы и так уже равны
		return *this;
	}
	while (head != nullptr) {//сначала очищаю список полностью
		Node* temp = head;
		head = head->ptr_to_next_node;
		delete temp;
	}
	size = 0;

	Node* current_node = list.head;//голова
	while (current_node != nullptr) {//пока не рпойду весь спиок
		push_bask(current_node->data);//добавляю очередной элемент в список
		current_node = current_node->ptr_to_next_node;//беру следующий элемент
	}
	this->begin = list.begin;
	this->end = list.end;
	return *this;
}

List List::operator *(List& list) {
	int temp_size;
	if (this->size > list.size) {//если первый размер списка больше размера 2 списка
		temp_size = list.size;//наименьший размер из двух
	}
	else {
		temp_size = this->size;//наименьший размер из двух
	}

	List temp(temp_size, 0);//список с наименьшей длиной
	for (int i = 0; i < temp_size; i++) {//пока не пройду по списку наименьшей длины
		temp[i] = (*this)[i] * list[i];
	}
	return temp;
}

ostream& operator <<(ostream& stream, const List& list) {//вывод
	cout << "Вывод элементов списка:" << endl;
	Node* current_node = list.head;
	while (current_node != nullptr) {//пока не пройду весь список
		stream << current_node->data << ' ';
		current_node = current_node->ptr_to_next_node;//беру следующий элемент
```

```
        }
        return stream;
}

istream& operator >>(istream& stream, const List& list) {//ввод
        cout << "Введите элементы списка" << endl;
        Node* current_node = list.head;
        while (current_node != nullptr) {//пока не пройду весь список
                stream >> current_node->data;
                current_node = current_node->ptr_to_next_node;//беру следующий элемент
        }
        return stream;
}
```

Файл Iterator.h

```
#pragma once //предотвращает повторную загрузку заголовочного файла, если он уже был включен
#include "List.h"
#include <iostream> //стандартные потоки ввода и вывода
using namespace std;

class Iterator {
private:
        friend struct Node;
        friend class List;
        Node* elem;
public:
        Iterator(); //конструктор по умолчанию
        Iterator(const Iterator& iter);//конструктор копирования
        ~Iterator() {}
        Iterator& operator =(const Iterator& iter);//текущий итератор
        bool operator == (const Iterator& iter);//элементы равны?
        bool operator != (const Iterator& iter);//элементы НЕ равны?
        Iterator& operator ++(int);//следующий элемент
        Iterator& operator --(int);//предыдущий элемент
        Iterator& operator +(const int number);//элемент по индексу
        Iterator& operator -(const int number);//элемент по индексу
        int& operator *();//доступ к данным элемента по итератору
};
```

Файл Iterator.cpp

```
#include "list.h"
#include "iterator.h"
#include <iostream>
#include <string>
using namespace std;

Iterator::Iterator() {//конструктор по умолчанию
        this->elem = nullptr;
}

Iterator::Iterator(const Iterator& iter) {//конструктор копирования
        this->elem = iter.elem;
}

Iterator& Iterator:: operator = (const Iterator& iter) {//текущий итератор
        this->elem = iter.elem;
        return *this;
}

bool Iterator::operator == (const Iterator& iter) {//элементы равны?
```

```
	return this->elem == iter.elem;
}

bool Iterator::operator != (const Iterator& iter) {//элементы НЕ равны?
	return this->elem != iter.elem;
}

Iterator& Iterator::operator ++(int) {//следующий элемент
	this->elem = this->elem->ptr_to_next_node;
	return *this;
}

Iterator& Iterator::operator --(int) {
	this->elem = this->elem->ptr_to_prev_node;//предыдущий элемент
	return *this;
}

Iterator& Iterator::operator +(const int number) {//элемент по индексу
	Iterator temp(*this);
	int i = 0;
	for (i; i < number; i++) {
		if (temp.elem == nullptr) throw 2;
		temp.elem = temp.elem->ptr_to_next_node;
	}
	return temp;
}

Iterator& Iterator::operator -(const int number) {//элемент по индексу
	Iterator temp(*this);
	for (int i = 0; i < number; i++) {
		if (temp.elem == nullptr) throw 1;
		temp.elem = temp.elem->ptr_to_prev_node;
	}
	return temp;
}

int& Iterator::operator *() {//доступ к данным элемента по итератору
	return this->elem->data;
}
```

Файл Class_9-1.cpp

```
#include "List.h"
#include "Iterator.h"
#include <iostream>
#include <string>

using namespace std;

void creature_list_rand(List& list);
void print_list_iterator_start(List& list);
void print_list_iterator_end(List& list);
void msg_error(int& x) {
  switch (x){
  case 1:
    cout << "Недопустимый индекс списка! Отрицательный индекс недопустим." << endl;
    break;
  case 2:
    cout << "Недопустимый индекс списка! Индекс больше текущего размера списка." << endl;
    break;
  case 3:
    cout << "Недопустимый размер списка! Отрицательный рамер недопустим." << endl;
```

```
        break;
    case 4:
        cout << "Недопустимый размер списка! Превышен максимальный размер." << endl;
        break;
    default:
        break;
    }
    cout << endl;
}

int main() {
    system("chcp 1251 > Null");
    srand(time(0));

    List list_1(10, 0);
    cout << "list_1" << endl;
    cout << list_1 << endl << endl;

    creature_list_rand(list_1);
    cout << "list_1" << endl;
    cout << list_1 << endl;
    cout << "Первый элемент: " << *(list_1.first_el()) << endl;
    cout << "Последний элемент: " << *(list_1.last_el()) << endl << endl;

    List list_2(list_1);
    cout << "list_2" << endl;
    cout << list_2 << endl << endl;

    list_2.push_front(100);
    list_2.pop_bask();
    cout << list_2 << endl << endl;

    try {//недопустимый размер множества, отрицательный размер
        List list_3(-1);
        cout << "list_3" << endl;
        cin >> list_3;
        cout << list_3 << endl << endl;

    }
    catch (int& x) {
        cout << "Ошибка с кодом " << x << endl;
        msg_error(x);
    }

    List list_4(11);
    creature_list_rand(list_4);
    cout << "list_4" << endl;
    cout << list_4 << endl << endl;

    List list_5 = list_2 * list_4;
    cout << "list_5" << endl;
    cout << list_5 << endl << endl;

    Iterator iter = list_1.first_el();
    try {
        cout << "Вывод элемента с индексом 100 list_1 с помощью Iterator с НАЧАЛА" << endl;
        cout << *(iter + 100);
        cout << endl;
    }
    catch (int& x) {
        cout << "Ошибка с кодом " << x << endl;
        msg_error(x);
```

```
    }

    iter = list_1.last_el();
    try {
        cout << "Вывод элемента с индексом -100 list_1 с помощью Iterator с КОНЦА" << endl;
        cout << *(iter - 100);
        cout << endl;
    }
    catch (int& x) {
        cout << "Ошибка с кодом " << x << endl;
        msg_error(x);
    }

    try {
        cout << "Вывод элемента с индексом 20 list_5 с помощью перегрузки операции" << endl;
        cout << list_5[20];
        cout << endl;
    }
    catch (int& x) {
        cout << "Ошибка с кодом " << x << endl;
        msg_error(x);
    }

    try {
        cout << "Вывод элемента с индексом -1 list_5 с помощью перегрузки операции" << endl;
        cout << list_5[-1];
        cout << endl;
    }
    catch (int& x) {
        cout << "Ошибка с кодом " << x << endl;
        msg_error(x);
    }

    cout << endl;

    return 0;
}

void creature_list_rand(List& list) {
    for (int i = 0; i < list(); i++) {
        list[i] = rand() % (100 + 1) - 50;
    }
}

void print_list_iterator_start(List& list) {
    for (Iterator iter = list.first_el(); iter != list.last_el() + 1; iter++) {
        cout << *iter << ' ';
    }
}

void print_list_iterator_end(List& list) {
    for (Iterator iter = list.last_el(); iter != list.first_el() - 1; iter--) {
        cout << *iter << ' ';
    }
}
```

Результаты работы

```
list_1
Вывод элементов списка:
0 0 0 0 0 0 0 0 0 0

list_1
Вывод элементов списка:
-39 12 7 -19 49 49 -8 -24 21 3
Первый элемент: -39
Последний элемент: 3

list_2
Вывод элементов списка:
-39 12 7 -19 49 49 -8 -24 21 3

Вывод элементов списка:
100 -39 12 7 -19 49 49 -8 -24 21

Ошибка с кодом 3
Недопустимый размер списка! Отрицательный рамер недопустим.

list_4
Вывод элементов списка:
41 -22 34 -46 -38 38 -21 -3 -17 48 -22

list_5
Вывод элементов списка:
4100 858 408 -322 722 1862 -1029 24 408 1008

Вывод элемента с индексом 100 list_1 с помощью Iterator с НАЧАЛА
Ошибка с кодом 2
Недопустимый индекс списка! Индекс больше текущего размера списка.

Вывод элемента с индексом -100 list_1 с помощью Iterator с КОНЦА
Ошибка с кодом 1
Недопустимый индекс списка! Отрицательный индекс недопустим.

Вывод элемента с индексом 20 list_5 с помощью перегрузки операции
Ошибка с кодом 2
Недопустимый индекс списка! Индекс больше текущего размера списка.

Вывод элемента с индексом -1 list_5 с помощью перегрузки операции
Ошибка с кодом 1
Недопустимый индекс списка! Отрицательный индекс недопустим.
```

Анализ задачи 2

1) Для реализации класса список необходимо создать структуру Node.

2) В классе список необходимо реализовать конструкторы, деструктор, функции вставки/ удаления первого/последнего элемента, а также операторы перегрузки.

3) 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).

4) В классе Iterator необходимо указать, что класс List – дружественный.

5) Исключительные ситуации генерируются:

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

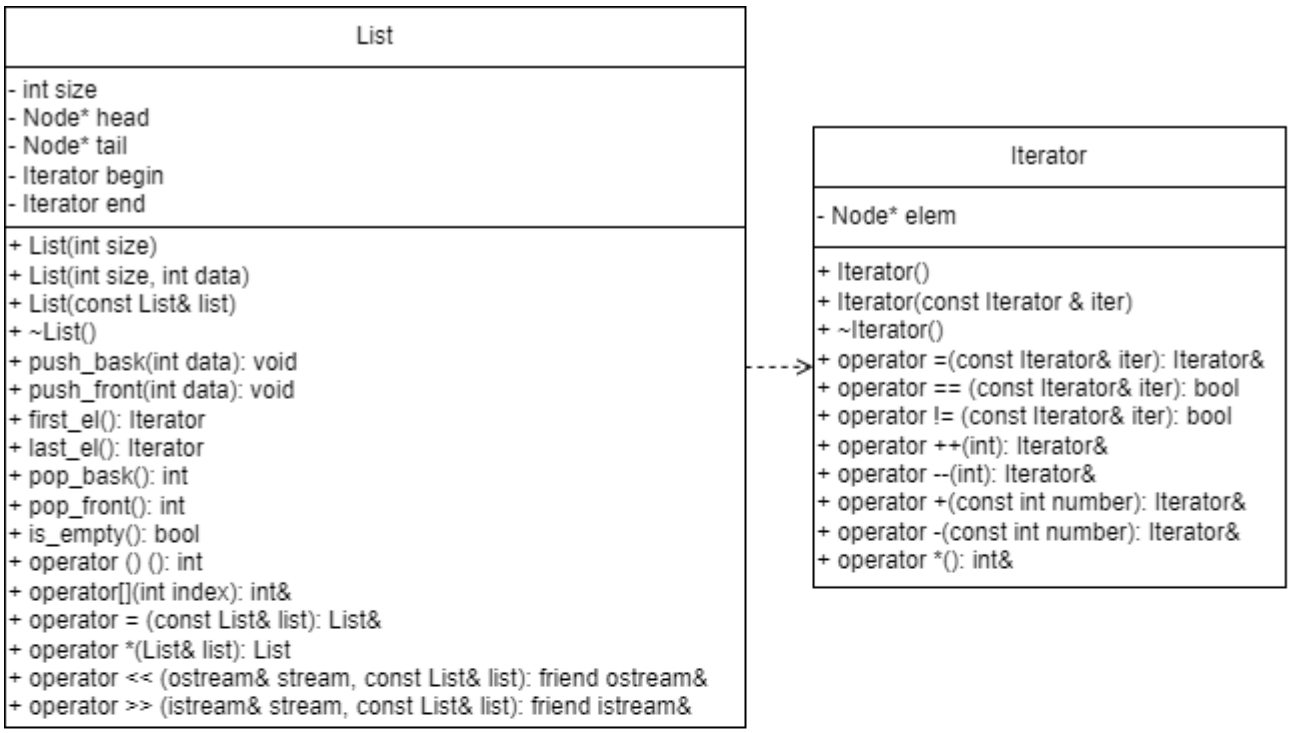
2, 3 – в операции [] – при попытке обратиться к элементу с номером меньше 0 или больше текущего размера вектора;

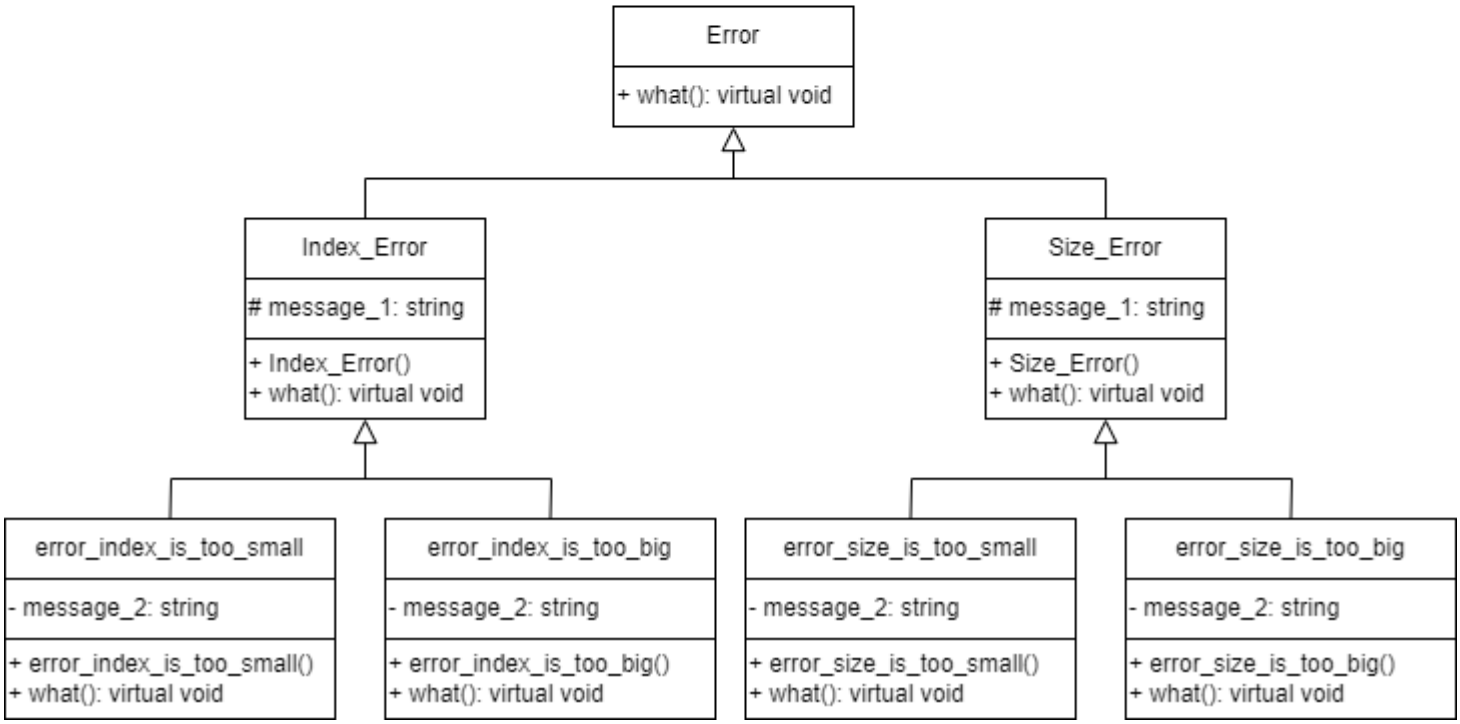
– в операции + – при попытке добавить элемент с номером больше максимального размера;

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

6) Информация об исключительных ситуациях передается с помощью иерархии пользовательских классов.

UML диаграмма





Код

Файл List.h

```
#pragma once //предотвращает повторную загрузку заголовочного файла, если он уже был включен
#include <iostream> //стандартные потоки ввода и вывода
#include <string>
#include "Iterator.h"
#include "Error.h"
using namespace std;

struct Node {
	int data; //данные
```

```
        Node* ptr_to_prev_node = nullptr; //указатель на предыдущий элемент
        Node* ptr_to_next_node = nullptr; //указатель на следующий элемент
};

class List {
private:
        int size; //размер
        Node* head; //указатель на голову
        Node* tail; //указатель на хвост
        Iterator begin;//указатель на начальный элемент
        Iterator end;//указатель на последний элемент
public:
        List(int size); //конструктор с длиной
        List(int size, int data);//конструктор с длиной и данными
        List(const List& list);//конструктор копирования
        ~List();//деструктор

        void push_bask(int data);//вставка элемента в конец
        void push_front(int data);//вставка элемента в НАЧАЛО

        Iterator first_el();//последний элемент
        Iterator last_el();//первый элемент

        int pop_bask();//удалить последний элемент
        int pop_front();//удалить первый элемент

        bool is_empty();//список пустой иль нет

        int operator () ();//размер списка
        int& operator[](int index);//данные по индексу
        List& operator = (const List& list);//оператор присваивания
        List operator *(List& list);

        friend ostream& operator << (ostream& stream, const List& list);//ВЫВОД
        friend istream& operator >> (istream& stream, const List& list);//ВВОД
};
```

Файл List.cpp

```
#include "List.h"
using namespace std;

List::List(int size) {//конструктор с длиной
        if (size <= 0) throw error_size_is_too_small();
        this->size = size;
        if (size > 0) {//если длина больше 0
                Node* node = new Node;//создаю узел
                this->head = node; //указатель на голову
                this->tail = node; //указатель на хвост
                for (int i = 1; i < size; i++) {//заполняю со второго номера
                        Node* New_Node = new Node;//новый узел
                        tail->ptr_to_next_node = New_Node;//хвост указывает на новый узел
                        New_Node->ptr_to_prev_node = tail;//новый узел ук-т на хвост как на пред-й
                        tail = New_Node;//новый узел становится хвостом
                }
                tail->ptr_to_next_node = nullptr;//до хвоста элементов нет
        }
        else {
                this->head = nullptr; //указатель на голову
                this->tail = nullptr; //указатель на хвост
        }
        this->begin.elem = this->head;
        this->end.elem = this->tail;
```

```
}

List::List(int size, int data) {//конструктор с длиной и данными
	if (size <= 0) throw error_size_is_too_small();
	this->size = size;
	if (size > 0) {//если длина больше 0
		Node* node = new Node;//создаю узел
		node->data = data; //данные для узла
		this->head = node; //указатель на голову
		this->tail = node; //указатель на хвост
		for (int i = 1; i < size; i++) {//заполняю со второго номера
			Node* New_Node = new Node;//новый узел
			New_Node->data = data; //данные для нового узла
			tail->ptr_to_next_node = New_Node;//хвост указывает на новый узел
			New_Node->ptr_to_prev_node = tail;//новый узел ук-т на хвост как на пред-й
			tail = New_Node;//новый узел становится хвостом
		}
		tail->ptr_to_next_node = nullptr;//до хвоста элементов нет
	}
	else {
		this->head = nullptr; //указатель на голову
		this->tail = nullptr; //указатель на хвост
	}
	this->begin.elem = this->head;
	this->end.elem = this->tail;
}

List::List(const List& list) {//конструктор копирования
	this->head = nullptr; //указатель на голову
	this->tail = nullptr; //указатель на хвост
	this->size = 0;

	Node* current_node = list.head;//создаю узел
	while (current_node != nullptr) {//пока не пройду весь список
		push_bask(current_node->data);//добавляю в конец эл-т
		current_node = current_node->ptr_to_next_node;//беру следующий элемент
	}
	this->begin.elem = this->head;
	this->end.elem = this->tail;
}

List::~List() {//деструктор
	Node* current_node = head;//голова
	while (current_node != nullptr) {//пока не пройду весь список
		Node* next_node = current_node->ptr_to_next_node;//следующий элемент
		delete current_node;//удаляю текущий элемент
		current_node = next_node;//беру новый элемент
	}
	head = nullptr; //голова пустая
}

void List::push_bask(int data) {//вставка элемента в конец
	Node* New_Node = new Node;//новый узел
	New_Node->data = data; //данные для нового узла
	New_Node->ptr_to_next_node = nullptr;//новый узел пока никуда не указывает
	if (this->head == nullptr) {//если список пустой
		this->head = New_Node;//новый узел - голова
		this->tail = New_Node;//новый узел - хвост
		this->begin.elem = this->head;
		this->end.elem = this->tail;
	}
	else {
```

```
                tail->ptr_to_next_node = New_Node;//хвост указывает на узел
                New_Node->ptr_to_prev_node = tail;
                tail = New_Node; ////новый узел - хвост
                this->end.elem = this->tail;
        }
        this->size++;//увеличиваю длину
}

void List::push_front(int data) {//вставка элемента в НАЧАЛО
        Node* New_Node = new Node;//новый узел
        New_Node->data = data; //данные для нового узла

        if (this->head == nullptr) {//если список пустой
                this->head = New_Node;//новый узел - голова
                this->tail = New_Node;//новый узел - хвост
                this->begin.elem = this->head;
                this->end.elem = this->tail;
        }
        else {
                head->ptr_to_prev_node = New_Node;//голова указывает на узел
                New_Node->ptr_to_next_node = head;
                head = New_Node; //новый узел - голова
                this->begin.elem = this->head;
        }
        this->size++;//увеличиваю длину
}

Iterator List::first_el() {//первый элемент
        return this->begin;
}

Iterator List::last_el() {//первый элемент
        return this->end;
}

int List::pop_bask() {//удаление последнего
        int temp;
        if (this->tail != nullptr) {//если список не пустой
                Node* current_node = this->tail;//узел предыдущий
                tail = current_node->ptr_to_prev_node;//меняю хвост
                temp = current_node->data;
                tail->ptr_to_next_node = nullptr;
                this->size--;//уменьшаю размер
                this->end.elem = tail;
        }
        return temp;//значение из удаленной переменной
}

int List::pop_front() {//удаление первого
        int temp;
        if (this->tail != nullptr) {//если список не пустой
                Node* current_node = this->head;//узел предыдущий
                head = current_node->ptr_to_next_node;//меняю хвост
                temp = current_node->data;
                head->ptr_to_prev_node = nullptr;
                this->size--;//уменьшаю размер
                this->begin.elem = head;
        }
        return temp;//значение из удаленной переменной
}

bool List::is_empty() {
```

```
	return this->size == 0;//если пустой, то вернет 1 иначе 0
}

int List::operator () () {//размер списка
	return this->size;
}

int& List::operator[](int index) {
	if (index < this->size && index >= 0) {//если индекс меньше размера и больше -
		Node* current_node = this->head;
		for (int i = 0; i != index; i++) {
			current_node = current_node->ptr_to_next_node;//беру следующий элемент
		}
		return current_node->data;
	}
	else {
		if (index > this->size) throw error_index_is_too_big();
		if (index < 0) throw error_index_is_too_small();
		exit(0);
	}
}

List& List::operator = (const List& list) {
	if (this == &list) {//если элементы и так уже равны
		return *this;
	}
	while (head != nullptr) {//сначала очищаю список полностью
		Node* temp = head;
		head = head->ptr_to_next_node;
		delete temp;
	}
	size = 0;

	Node* current_node = list.head;//голова
	while (current_node != nullptr) {//пока не рпойду весь спиок
		push_bask(current_node->data);//добавляю очередной элемент в список
		current_node = current_node->ptr_to_next_node;//беру следующий элемент
	}
	this->begin = list.begin;
	this->end = list.end;
	return *this;
}

List List::operator *(List& list) {
	int temp_size;
	if (this->size > list.size) {//если первый размер списка больше размера 2 списка
		temp_size = list.size;//наименьший размер из двух
	}
	else {
		temp_size = this->size;//наименьший размер из двух
	}

	List temp(temp_size, 0);//список с наименьшей длиной
	for (int i = 0; i < temp_size; i++) {//пока не пройду по списку наименьшей длины
		temp[i] = (*this)[i] * list[i];
	}
	return temp;
}

ostream& operator <<(ostream& stream, const List& list) {//вывод
	cout << "Вывод элементов списка:" << endl;
	Node* current_node = list.head;
```

```
	while (current_node != nullptr) {//пока не пройду весь список
		stream << current_node->data << ' ';
		current_node = current_node->ptr_to_next_node;//беру следующий элемент
	}
	return stream;
}

istream& operator >>(istream& stream, const List& list) {//ввод
	cout << "Введите элементы списка" << endl;
	Node* current_node = list.head;
	while (current_node != nullptr) {//пока не пройду весь список
		stream >> current_node->data;
		current_node = current_node->ptr_to_next_node;//беру следующий элемент
	}
	return stream;
}
```

Файл Iterator.h

```
#pragma once //предотвращает повторную загрузку заголовочного файла, если он уже был включен
#include "List.h"
#include <iostream> //стандартные потоки ввода и вывода
using namespace std;

class Iterator {
private:
	friend struct Node;
	friend class List;
	Node* elem;
public:
	Iterator(); //конструктор по умолчанию
	Iterator(const Iterator& iter);//конструктор копирования
	~Iterator() {}
	Iterator& operator =(const Iterator& iter);//текущий итератор
	bool operator == (const Iterator& iter);//элементы равны?
	bool operator != (const Iterator& iter);//элементы НЕ равны?
	Iterator& operator ++(int);//следующий элемент
	Iterator& operator --(int);//предыдущий элемент
	Iterator& operator +(const int number);//элемент по индексу
	Iterator& operator -(const int number);//элемент по индексу
	int& operator *();//доступ к данным элемента по итератору
};
```

Файл Iterator.cpp

```
#include "list.h"
#include "iterator.h"
#include <iostream>
#include <string>
using namespace std;

Iterator::Iterator() {//конструктор по умолчанию
	this->elem = nullptr;
}

Iterator::Iterator(const Iterator& iter) {//конструктор копирования
	this->elem = iter.elem;
}

Iterator& Iterator:: operator = (const Iterator& iter) {//текущий итератор
	this->elem = iter.elem;
	return *this;
```

```
}

bool Iterator::operator == (const Iterator& iter) {//элементы равны?
	return this->elem == iter.elem;
}

bool Iterator::operator != (const Iterator& iter) {//элементы НЕ равны?
	return this->elem != iter.elem;
}

Iterator& Iterator::operator ++(int) {//следующий элемент
	this->elem = this->elem->ptr_to_next_node;
	return *this;
}

Iterator& Iterator::operator --(int) {
	this->elem = this->elem->ptr_to_prev_node;//предыдущий элемент
	return *this;
}

Iterator& Iterator::operator +(const int number) {//элемент по индексу
	Iterator temp(*this);
	int i = 0;
	for (i; i < number; i++) {
		if(temp.elem == nullptr) throw error_index_is_too_big();
		temp.elem = temp.elem->ptr_to_next_node;
	}
	return temp;
}

Iterator& Iterator::operator -(const int number) {//элемент по индексу
	Iterator temp(*this);
	for (int i = 0; i < number; i++) {
		if (temp.elem == nullptr) throw error_index_is_too_small();
		temp.elem = temp.elem->ptr_to_prev_node;
	}
	return temp;
}

int& Iterator::operator *() {//доступ к данным элемента по итератору
	return this->elem->data;
}
```

Файл Error.h

```
#pragma once
#include <string>
#include <iostream>
using namespace std;

class Error{
public:
	virtual void what() {};
};
//---------------------------------
class Index_Error: public Error {
protected:
	string message_1;
public:
	Index_Error() {
		message_1 = "Недопустимый индекс списка!";
	}
```

```
	virtual void what() {
		cout << message_1 << endl;
	}
};

class error_index_is_too_small : public Index_Error {
	string message_2;
public:
	error_index_is_too_small() {
		Index_Error();
		message_2 = "Отрицательный индекс недопустим.";
	}
	virtual void what() {
		cout << message_1 << " " << message_2 << endl << endl;
	}
};

class error_index_is_too_big : public Index_Error {
	string message_2;
public:
	error_index_is_too_big() {
		Index_Error();
		message_2 = "Индекс больше текущего размера списка.";
	}
	virtual void what() {
		cout << message_1 << " " << message_2 << endl << endl;
	}
};

//--------------------

class Size_Error : public Error {
protected:
	string message_1;
public:
	Size_Error() {
		message_1 = "Недопустимый размер списка!";
	}
	virtual void what() {
		cout << message_1 << endl;
	}
};

class error_size_is_too_small : public Size_Error {
	string message_2;
public:
	error_size_is_too_small() {
		Index_Error();
		message_2 = "Отрицательный рамер недопустим.";
	}
	virtual void what() {
		cout << message_1 << " " << message_2 << endl << endl;
	}
};

class error_size_is_too_big : public Size_Error {//как будто бы не нужно
	string message_2;
public:
	error_size_is_too_big() {
		Index_Error();
		message_2 = "Превышен максимальный размер.";
	}
```

```
        virtual void what() {
                cout << message_1 << " " << message_2 << endl << endl;
        }
};
```

Файл Class_9-3.cpp

```
#include "List.h"
#include "Iterator.h"
#include "Error.h"
#include <iostream>
#include <string>

using namespace std;

void creature_list_rand(List& list);
void print_list_iterator_start(List& list);
void print_list_iterator_end(List& list);

int main() {
   system("chcp 1251 > Null");
   srand(time(0));

   List list_1(10, 0);
   cout << "list_1" << endl;
   cout << list_1 << endl << endl;

   creature_list_rand(list_1);
   cout << "list_1" << endl;
   cout << list_1 << endl;
   cout << "Первый элемент: " << *(list_1.first_el()) << endl;
   cout << "Последний элемент: " << *(list_1.last_el()) << endl << endl;

   List list_2(list_1);
   cout << "list_2" << endl;
   cout << list_2 << endl << endl;

   list_2.push_front(100);
   list_2.pop_bask();
   cout << list_2 << endl << endl;

   try {//недопустимый размер множества, отрицательный размер
      List list_3(-1);
      cout << "list_3" << endl;
      cin >> list_3;
      cout << list_3 << endl << endl;

   }
   catch (Error& er) {
      er.what();
   }

   List list_4(11);
   creature_list_rand(list_4);
   cout << "list_4" << endl;
   cout << list_4 << endl << endl;

   List list_5 = list_2 * list_4;
   cout << "list_5" << endl;
   cout << list_5 << endl << endl;

   Iterator iter = list_1.first_el();
```

```
    try {
        cout << "Вывод элемента с индексом 100 list_1 с помощью Iterator с НАЧАЛА" << endl;
        cout << *(iter + 100);
        cout << endl;
    }
    catch (Error& er) {
        er.what();
    }

    iter = list_1.last_el();
    try {
        cout << "Вывод элемента с индексом -100 list_1 с помощью Iterator с КОНЦА" << endl;
        cout << *(iter - 100);
        cout << endl;
    }
    catch (Error& er) {
        er.what();
    }

    try {
        cout << "Вывод элемента с индексом 20 list_5 с помощью перегрузки операции" << endl;
        cout << list_5[20];
        cout << endl;
    }
    catch (Error& er) {
        er.what();
    }

    try {
        cout << "Вывод элемента с индексом -1 list_5 с помощью перегрузки операции" << endl;
        cout << list_5[-1];
        cout << endl;
    }
    catch (Error& er) {
        er.what();
    }

    cout << endl;

    return 0;
}

void creature_list_rand(List& list) {
    for (int i = 0; i < list(); i++) {
        list[i] = rand() % (100 + 1) - 50;
    }
}

void print_list_iterator_start(List& list) {
    for (Iterator iter = list.first_el(); iter != list.last_el() + 1; iter++) {
        cout << *iter << ' ';
    }
}

void print_list_iterator_end(List& list) {
    for (Iterator iter = list.last_el(); iter != list.first_el() - 1; iter--) {
        cout << *iter << ' ';
    }
}
```

Результаты работы

```
list_1
Вывод элементов списка:
0 0 0 0 0 0 0 0 0 0

list_1
Вывод элементов списка:
16 36 -49 -35 15 -20 -46 48 -13 -35
Первый элемент: 16
Последний элемент: -35

list_2
Вывод элементов списка:
16 36 -49 -35 15 -20 -46 48 -13 -35

Вывод элементов списка:
100 16 36 -49 -35 15 -20 -46 48 -13

Недопустимый размер списка! Отрицательный рамер недопустим.

list_4
Вывод элементов списка:
-31 7 7 -13 -15 -1 21 48 14 -21 -43

list_5
Вывод элементов списка:
-3100 112 252 637 525 -15 -420 -2208 672 273

Вывод элемента с индексом 100 list_1 с помощью Iterator с НАЧАЛА
Недопустимый индекс списка! Индекс больше текущего размера списка.

Вывод элемента с индексом -100 list_1 с помощью Iterator с КОНЦА
Недопустимый индекс списка! Отрицательный индекс недопустим.

Вывод элемента с индексом 20 list_5 с помощью перегрузки операции
Недопустимый индекс списка! Индекс больше текущего размера списка.

Вывод элемента с индексом -1 list_5 с помощью перегрузки операции
Недопустимый индекс списка! Отрицательный индекс недопустим.
```

Вывод

В ходе работы я применила знания об обработке исключительных ситуаций. Исключение – это непредвиденное или аварийное событие. Исключения позволяют разделить вычислительный процесс на 2 части: 1) обнаружение аварийной ситуации (неизвестно как обрабатывать); 2) обработка аварийной ситуации (неизвестно, где она возникла). 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;

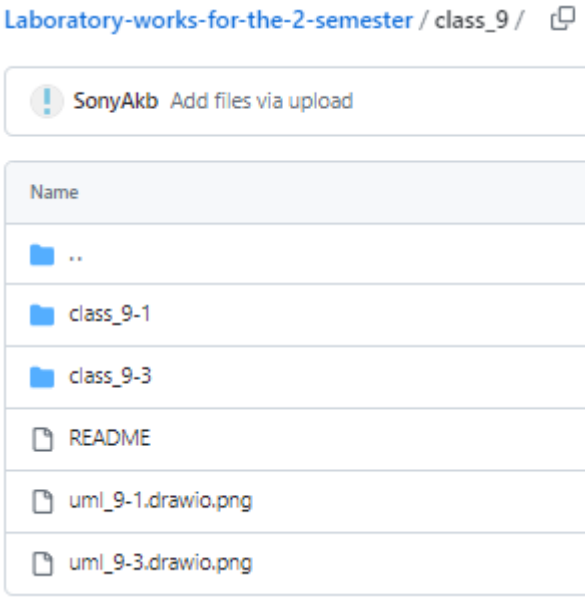
4 – в операции + – при попытке добавить элемент с номером больше максимального размера;

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

В основной функции были продемонстрированы обработки исключительных ситуаций.

GitHub

Ссылка: https://github.com/SonyAkb/Laboratory-works-for-the-2-semester/tree/main/class_9



Контрольные вопросы

1. Что представляет собой исключение в C++?

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

4. 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;

}
```

5. 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);
         }
```

6. 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.

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

class Error

8. 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

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

Функция f1() 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));
}
```