Министерство образования и науки Российской Федерации  
Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский  
политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе № 15.9**

Дисциплина: «Информатика»

Тема Объектно-ориентированное программирование Обработка исключительных ситуаций

Вариант 12

Выполнила:

Студентка группы РИС-22-1б

Черкасова А.А.

Проверила:

Доцент кафедры ИТАС

Полякова О. А.

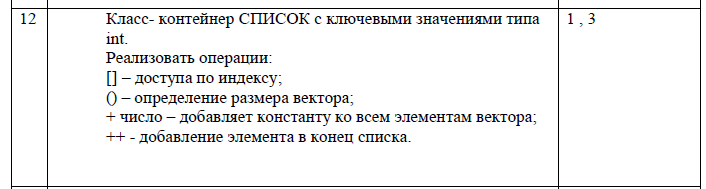
**Пермь 2023**

**Цель работы**

1. Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.
2. Реализация класса-контейнера.

**Постановка задачи**

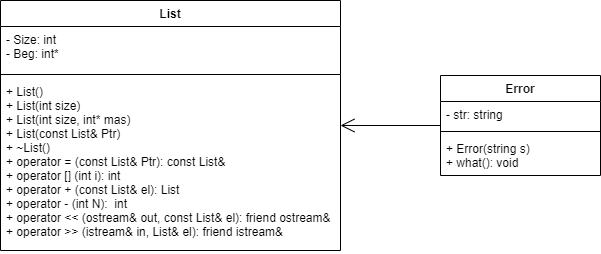
1. Реализовать класс, перегрузить для него операции, указанные в варианте.
2. Определить исключительные ситуации.
3. Предусмотреть генерацию исключительных ситуаций.

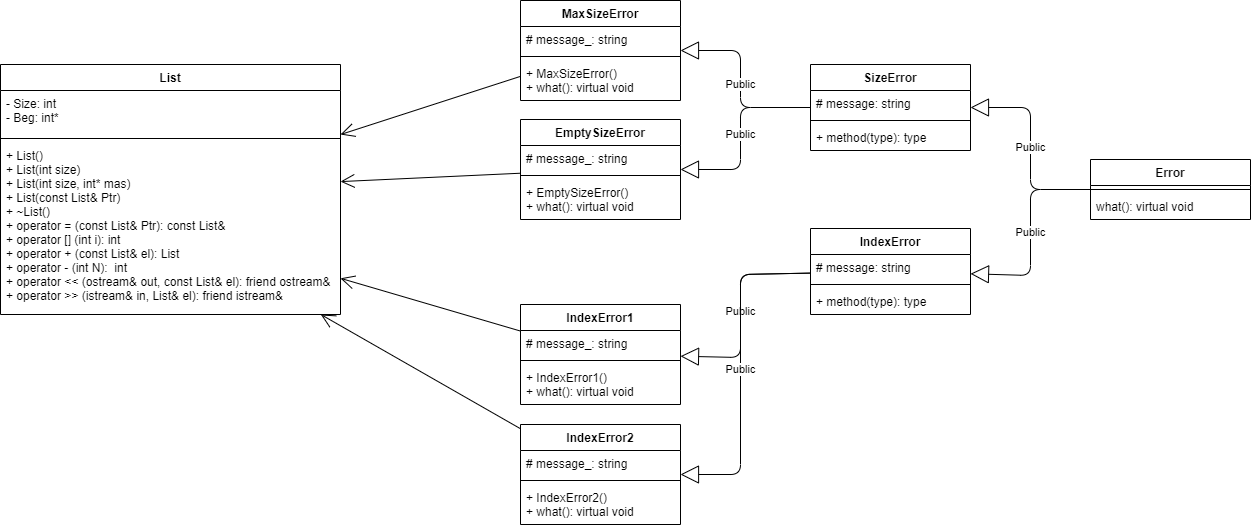


**Анализ задачи**

1. Для решения задачи необходимо:
   1. Реализовать определение классов Error / IndexError и определение необходимых методов данных классов.
   2. Реализовать определение классов IndexError1 / IndexError2 и определение необходимых методов и операторов данных классов.
   3. Реализовать определение производного от классов SizeError / MaxSizeError / EmptySizeError и определение необходимых методов и операторов данных классов.
   4. Реализовать определение класса List и определение необходимых методов и операторов данного класса
   5. Реализовать применение этих функций в главной функции.
2. Для решения задачи данные были представлены в следующем виде:
   1. Для работы с данными используются атрибуты класса.
3. Для операций ввода и вывода использовались следующие операторы и функции:
   1. Ввод данных реализован с помощью оператора cin, используемых при реализации в главной функции и функций перегрузки оператора.
   2. Вывод данных реализован с помощью оператора cout, используемого при реализации в главной функции и функций перегрузки операторов ввода-вывода.
4. Поставленные задачи решены следующими действиями:
   1. В классах Error находится реализация обработки исключительных ситуаций.
   2. В классе List находится реализация списка, определение размера списка, доступ элемента списка по индексу, сложение векторов.

**UML-Диаграмма**





**Реализация задачи на языке С++**

**(1)**

**Заголовочный файл List.h**

#pragma once

#include <iostream>

using namespace std;

#define MAX\_SIZE 10

struct Node

{

int data;

Node\* next;

Node\* prev;

Node()

{

data = 0;

next = nullptr;

prev = nullptr;

}

};

class List

{

private:

int size;

Node\* head;

Node\* tail;

// Iterator beg;

// Iterator end;

public:

List();

List(int n, int k = 0); // Конструктор с параметрами: выделение под s элементов и заполнение их значением k

List(const List&); // Конструктор копирования

~List(); // Деструктор

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

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

List& operator=(const List&);

int& operator[](int index); // Операция доступа по индексу

int& operator()(); // Длина списка

List operator+(int p); // Операция сложения

List& operator++(int elem); //добавление элемена в конец

void pushback(int); // добавление в конец

};

**Файл с описанием методов класса List.cpp**

#include "List.h"

#define SIZE\_ERROR 1

#define MIN\_INDEX\_ERROR 2

#define MAX\_INDEX\_ERROR 3

#define FULL\_SIZE\_ERROR 4

List::List()

{

head = nullptr;

tail = nullptr;

size = 0;

}

List::List(int n, int k)

{

if (n > MAX\_SIZE || n < 1) throw SIZE\_ERROR;

size = n;

head = nullptr; // пока нет элементов, помимо первого, нет и ссылки на следующий элемент

tail = nullptr;

Node\* node = new Node; // выделение памяти под узел

node->data = k; // заполнение первого узла данными

node->next = nullptr;

node->prev = nullptr;

head = node; //присваиваем данные первому узлу

tail = node;

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

{

Node\* NewNode = new Node; // Выделение димнамической памяти под новый элемент

Node\* PrevNode = node; //Указатель на "предыдущий" элемент(изначально тот, на котором стоит указатель на следующий)

node->next = NewNode;

node = node->next; // Переход указателем на следующий элемент

node->data = k; // Запись нового элемента

node->next = nullptr; // Указатель на следующий элемент равен нулю, т. к. был записан последний элемент

node->prev = PrevNode;

}

tail = node;

}

List::List(const List& p)

{

Node\* node = p.head;

while (node != nullptr)

{

pushback(node->data);

node = node->next;

}

}

void List::pushback(int k)

{

Node\* newNode = new Node;

newNode->data = k;

if (head == nullptr)

head = newNode;

else

{

tail->next = newNode;

newNode->prev = tail;

}

tail = newNode;

size++;

}

List::~List()

{

Node\* tmp = head;

while (tmp != nullptr)

{

head = tmp->next;

delete tmp;

tmp = head;

}

}

ostream& operator<<(ostream& out, const List& p)

{

Node\* tmp = p.head;

while (tmp != nullptr)

{

out << tmp->data << " ";

tmp = tmp->next;

}

out << endl;

return out;

}

istream& operator>>(istream& in, const List& p)

{

Node\* tmp = p.head;

while (tmp != nullptr)

{

in >> tmp->data;

tmp = tmp->next;

}

return in;

}

List List::operator+(int p)

{

List tmpList(this->size);

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

tmpList[i] = (\*this)[i] + p;

return tmpList;

}

List& List::operator++(int elem)

{

if (size >= MAX\_SIZE) throw FULL\_SIZE\_ERROR;

this->pushback(elem);

return \*this;

}

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

{

if (index < size && index >= 0)

{

Node\* tmp = head;

for (int i = 0; i < index; i++) tmp = tmp->next;

return tmp->data;

}

else

{

if (index >= size)

{

throw MAX\_INDEX\_ERROR;

}

if (index < 0)

{

throw MIN\_INDEX\_ERROR;

}

}

}

int& List::operator()()

{

return size;

}

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

{

if (this == &p) return \*this;

Node\* node = head;

while (node != nullptr) {

head = node->next;

delete node;

node = head;

size--;

}

Node\* curr = p.head;

while (curr != nullptr) {

pushback(curr->data);

curr = curr->next;

}

return \*this;

}

**Файл с главной программой main.cpp**

#include <iostream>

#include "List.h"

using namespace std;

int main()

{

List A1(4);

try

{

List A2(-2);

}

catch (int) { cout << "List size error!" << endl; }

try

{

List A3(15);

}

catch (int) { cout << "List size error!" << endl; }

List B1(A1 + 5);

cout << "List B1: element 3 is " << B1[2] << endl;

try

{

cout << "List B1: element 3 is " << B1[-3] << endl;

}

catch (int) { cout << "Min index error!" << endl; }

try

{

cout << "List B1: element 3 is " << B1[6] << endl;

}

catch (int) { cout << "Max index error!" << endl; }

List C1(10);

try

{

C1++;

}

catch (int) { cout << "List is already full! Can't add new element" << endl; }

return 0;

}

**(3)**

**Заголовочный файл Error.h**

#pragma once

#include <string>

#include <iostream>

using namespace std;

class Error // Базовый класс

{

public:

virtual void what() {};

};

class IndexError : public Error // Ошибка в индексе вектора

{

protected:

string message;

public:

IndexError()

{

message = "Index error!\n";

}

virtual void what()

{

cout << message;

}

};

class SizeError : public Error // Ошибка в размере вектора

{

protected:

string message;

public:

SizeError()

{

message = "Size error!\n";

}

virtual void what()

{

cout << message;

}

};

class MaxSizeError : public SizeError // Превышение максимального размера

{

protected:

string message\_;

public:

MaxSizeError()

{

SizeError();

message\_ = "Size > MAX\_SIZE!\n";

}

virtual void what()

{

cout << message << message\_;

}

};

class FullSizeError : public SizeError // Добавление в заполненный вектор

{

protected:

string message\_;

public:

FullSizeError()

{

SizeError();

message\_ = "List is full!\n";

}

virtual void what()

{

cout << message << message\_;

}

};

class MinIndexError : public IndexError // Индекс меньше нуля

{

protected:

string message\_;

public:

MinIndexError()

{

IndexError();

message\_ = "Index < 0!\n";

}

virtual void what()

{

cout << message << message\_;

}

};

class MaxIndexError : public IndexError // Индекс больше текущего размера вектора

{

protected:

string message\_;

public:

MaxIndexError()

{

IndexError();

message\_ = "Index > Size!\n";

}

virtual void what()

{

cout << message << message\_;

}

};

**Заголовочный файл List.h**

#pragma once

#include <iostream>

using namespace std;

#define MAX\_SIZE 10

struct Node

{

int data;

Node\* next;

Node\* prev;

Node()

{

data = 0;

next = nullptr;

prev = nullptr;

}

};

class List

{

private:

int size;

Node\* head;

Node\* tail;

// Iterator beg;

// Iterator end;

public:

List();

List(int n, int k = 0); // Конструктор с параметрами: выделение под s элементов и заполнение их значением k

List(const List&); // Конструктор копирования

~List(); // Деструктор

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

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

List& operator=(const List&);

int& operator[](int index); // Операция доступа по индексу

int& operator()(); // Длина списка

List operator+(int p); // Операция сложения

List& operator++(int elem); //добавление элемена в конец

void pushback(int); // добавление в конец

};

**Файл с описанием методов класса List.cpp**

#include "List.h"

#include "Error.h"

List::List()

{

head = nullptr;

tail = nullptr;

size = 0;

}

List::List(int n, int k)

{

if (n > MAX\_SIZE || n < 1) throw FullSizeError();

size = n;

head = nullptr; // пока нет элементов, помимо первого, нет и ссылки на следующий элемент

tail = nullptr;

Node\* node = new Node; // выделение памяти под узел

node->data = k; // заполнение первого узла данными

node->next = nullptr;

node->prev = nullptr;

head = node; //присваиваем данные первому узлу

tail = node;

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

{

Node\* NewNode = new Node; // Выделение димнамической памяти под новый элемент

Node\* PrevNode = node; //Указатель на "предыдущий" элемент(изначально тот, на котором стоит указатель на следующий)

node->next = NewNode;

node = node->next; // Переход указателем на следующий элемент

node->data = k; // Запись нового элемента

node->next = nullptr; // Указатель на следующий элемент равен нулю, т. к. был записан последний элемент

node->prev = PrevNode;

}

tail = node;

}

List::List(const List& p)

{

Node\* node = p.head;

while (node != nullptr)

{

pushback(node->data);

node = node->next;

}

}

void List::pushback(int k)

{

Node\* newNode = new Node;

newNode->data = k;

if (head == nullptr)

head = newNode;

else

{

tail->next = newNode;

newNode->prev = tail;

}

tail = newNode;

size++;

}

List::~List()

{

Node\* tmp = head;

while (tmp != nullptr)

{

head = tmp->next;

delete tmp;

tmp = head;

}

}

ostream& operator<<(ostream& out, const List& p)

{

Node\* tmp = p.head;

while (tmp != nullptr)

{

out << tmp->data << " ";

tmp = tmp->next;

}

out << endl;

return out;

}

istream& operator>>(istream& in, const List& p)

{

Node\* tmp = p.head;

while (tmp != nullptr)

{

in >> tmp->data;

tmp = tmp->next;

}

return in;

}

List List::operator+(int p)

{

List tmpList(this->size);

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

tmpList[i] = (\*this)[i] + p;

return tmpList;

}

List& List::operator++(int elem)

{

if (size >= MAX\_SIZE) throw FullSizeError();

this->pushback(elem);

return \*this;

}

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

{

if (index < size && index >= 0)

{

Node\* tmp = head;

for (int i = 0; i < index; i++) tmp = tmp->next;

return tmp->data;

}

else

{

if (index >= size)

{

throw MaxIndexError();

}

if (index < 0)

{

throw MinIndexError();

}

}

}

int& List::operator()()

{

return size;

}

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

{

if (this == &p) return \*this;

Node\* node = head;

while (node != nullptr) {

head = node->next;

delete node;

node = head;

size--;

}

Node\* curr = p.head;

while (curr != nullptr) {

pushback(curr->data);

curr = curr->next;

}

return \*this;

}

**Файл с главной программой main.cpp**

#include <iostream>

#include "List.h"

#include "Error.h"

using namespace std;

int main()

{

List A1(4);

try

{

List A2(-2);

}

catch (Error& e) { cout << "List size error!" << endl; }

try

{

List A3(15);

}

catch (Error& e) { cout << "List size error!" << endl; }

List B1(A1 + 5);

cout << "List B1: element 3 is " << B1[2] << endl;

try

{

cout << "List B1: element 3 is " << B1[-3] << endl;

}

catch (Error& e) { cout << "Min index error!" << endl; }

try

{

cout << "List B1: element 3 is " << B1[6] << endl;

}

catch (Error& e) { cout << "Max index error!" << endl; }

List C1(10);

try

{

C1++;

}

catch (Error& e) { cout << "List is already full! Can't add new element" << endl; }

return 0;

}

**Ответы на вопросы**

1. Что представляет собой исключение в С++?

В С++ исключение – это объект, который система должна генерировать при возникновении исключительной ситуации. Генерация такого объекта и создает исключительную ситуацию.

2. На какие части исключения позволяют разделить вычислительный процесс? Достоинства такого подхода?

Исключения позволяют разделить вычислительный процесс на 2 части:

● Обнаружение аварийной ситуации (неизвестно как обрабатывать);

● Обработка аварийной ситуации (неизвестно, где она возникла).

Достоинства такого подхода:

● Удобно использовать в программе, которая состоит из нескольких модулей;

● Не требуется возвращать значение в вызывающую функцию.

Общая схема:

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. Для представления исключений часто используют пустой класс:

class ZeroDevide{};

Генерация исключения будет выглядеть:

throw ZeroDevide();//вызывается конструктор без параметров

или

throw new ZeroDevide();

4. Что представляет собой контролируемый блок? Для чего он нужен?

Проверка возникновения исключения делается с помощью оператора try, с которым неразрывно связаны одна или несколько блоков обработки исключений — catch. Оператор try объявляет в любом месте программы контролируемый блок, который имеет следующий вид:

try { /\* контролируемый блок \*/ }

Контролируемый блок, помимо функции контроля, обладает функциями обычного блока: все переменные, объявленные внутри него, являются локальными в этом блоке и не видны вне его.

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

После блока try обязательно прописывается один или несколько блоков catch. Блок кода после catch предложения является обработчиком исключений. Это обработчик, который перехватывает исключение, которое возникает, если типы в throw выражениях и catch совместимы. Форма записи секции-ловушки следующая:

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

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

Спецификация исключения может иметь три формы:

1) (тип имя)

2) (тип)

3) (…)

Тип – это встроенный тип или тип, определенный 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 еще не были обработаны.

Форма 1 означает, что объект передается в блок обработки, чтобы его каким-то образом там использовать, например, для вывода информации в сообщении об ошибке. Примеры:

● catch( exception e) // по значению

● catch( exception &e) // по ссылке

● catch( const exception &e) // по константной ссылке

● catch( exception \*e) //по указателю

Лучше всего передавать объект по ссылке, т. к. при этом не создается временный объект-исключение.

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

В составе стандартной библиотеки С++ реализован ряд стандартных исключений, которые организованы в иерархию классов.

Эта иерархия может служить основой для создания собственных классов исключений и иерархии исключений. Можно определять собственные исключения, унаследовав их от класса exception.

8. Каким образом можно создать собственную иерархию исключений?

Для создания собственной иерархии исключений надо объявить свой базовый класс-исключение, например:

class BaseException{};

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

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.

9. Если спецификация исключений имеет вид:

void f1()throw(int,double);

то какие исключения может прождать функция f1()?

Для каждой функции, метода, конструктора или деструктора можно в заголовке указать спецификацию исключений. Если в заголовке спецификация исключений не указана, считается, что функция может порождать любое исключение, если указана, то считается, что функция генерирует те исключения, которые явно указаны в этом списке.

Следовательно, функция f1() может генерировать исключения типа int и double.

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

Если спецификация имеет вид такой вид, то считается, что функция исключений не генерирует.

11. В какой части программы может генерироваться исключение?

Исключение могут генерируется внутри оператора try { }.

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));

}