Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра информационных технологий и автоматизированных систем

**ОТЧЕТ**

**о работе по информатике**

**Тема:** Лабораторная работа № 9 по ООП

Семестр: 2

Выполнил студент ИВТ-22-2б:

Солодов Александр Андреевич

Проверил доцент кафедры ИТАС:

Полякова Ольга Андреевна

Пермь 2023

**Постановка задачи**

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

*Вариант 12*

Класс-контейнер СПИСОК с ключевыми значениями типа int. Реализовать операции:

[] – доступа по индексу;

() – определение размера списка;

+ число – добавляет константу ко всем элементам вектора;

++ - добавление элемента в конец списка.

Варианты реализации: 1,3

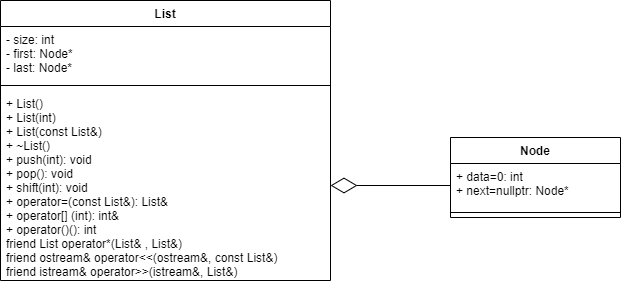
**Исключительные ситуации**

Исключительные ситуации генерируются:

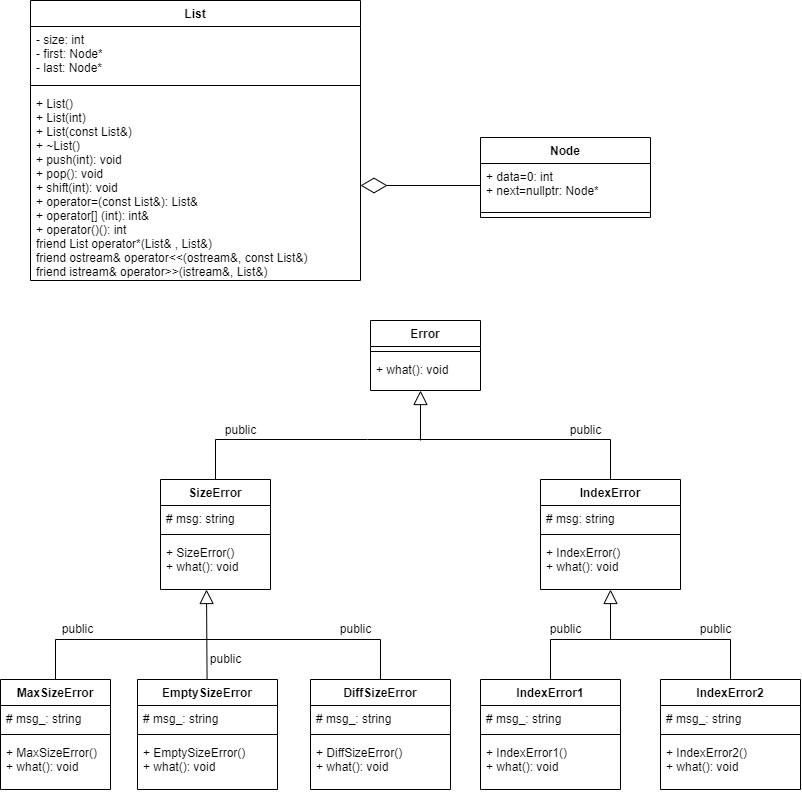
1 – в конструкторе с параметром при попытке создать вектор больше максимального размера;

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

В остальных случаях – exception.

**UML-диаграммы**

Реализация 1.

Реализация 3.

**Программное решение**

*Реализация 1:*

List.h

#pragma once

#include <iostream>

using namespace std;

const int MAX\_SIZE = 5;

class Node

{

public:

int data = 0;

Node\* pNext = nullptr;

Node(int data = 0, Node\* pNext = nullptr) {

this->data = data;

this->pNext = pNext;

}

};

class List

{

private:

int size;

Node\* first;

Node\* last;

public:

List();

List(int);

List(const List&);

~List();

void push\_back(int);

void pop\_front();

List& operator = (const List&);

int& operator [] (int);

int operator () ();

void operator + (const int);

void operator ++ ();

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

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

};

List.cpp

#include "List.h"

//конструктор по умолчанию

List::List() {

size = 0;

first = nullptr;

last = nullptr;

}

//конструктор с параметрами

List::List(int s) {

if (s > MAX\_SIZE) { //обработка превышения максимального размера

throw 1;

}

size = s;

first = nullptr;

last = nullptr;

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

int rnd = rand() % 10;

Node\* addNode = new Node(rnd);

if (last == nullptr) {

first = addNode;

last = addNode;

}

else {

last->pNext = addNode;

last = addNode;

}

}

}

//конструктор копирования

List::List(const List& list) {

size = 0;

first = nullptr;

last = nullptr;

Node\* current = list.first;

while (current != nullptr) {

Node\* addNode = new Node();

addNode->data = current->data;

addNode->pNext = nullptr;

if (last == nullptr) {

first = addNode;

last = addNode;

}

else {

last->pNext = addNode;

last = addNode;

}

current = current->pNext;

size++;

}

}

//деструктор

List::~List() {

while (size > 0) {

pop\_front();

}

}

//функция для добавления элемента в конец списка

void List::push\_back(int data) {

if (size + 1 == MAX\_SIZE) {

throw 1; //обработка превышения максимального размера

}

if (first == nullptr) {

first = new Node(data);

}

else {

Node\* current = this->first;

while (current->pNext != nullptr) {

current = current->pNext;

}

current->pNext = new Node(data);

}

size++;

}

//функция для удаления элемента из начала списка

void List::pop\_front() {

if (first == nullptr) {

throw exception("Список пуст!"); //обработка случая с пустым списком

}

Node\* temp = first;

first = first->pNext;

delete temp;

size--;

}

//перегрузка оператора присваивания

List& List::operator = (const List& other) {

if (this == &other) {

return \*this;

}

while (first != nullptr) {

Node\* temp = first;

first = first->pNext;

delete temp;

}

Node\* currentOther = other.first;

while (currentOther != nullptr) {

push\_back(currentOther->data);

currentOther = currentOther->pNext;

}

return \*this;

}

//перегрузка квадратных скобок

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

if (index < 0)

throw 2; //обработка индекса меньше нуля

if (index >= size)

throw 1; //обработка индекса больше размера

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

int count = 0;

Node\* current = this->first;

while (current != nullptr) {

if (count == index) {

return current->data;

}

current = current->pNext;

count++;

}

}

}

//перегрузка круглых скобок - получение размера списка

int List::operator () () {

return size;

}

//перегрузка оператора сложения

void List::operator + (const int add) {

Node\* current = this->first;

while (current != nullptr) {

current->data = current->data + add;

current = current->pNext;

}

}

//перегрузка инкремента

void List::operator ++ () {

if (size + 1 == MAX\_SIZE) {

throw 3; //обработка превышения максимального размера

}

if (first == nullptr) {

first = new Node();

}

else {

Node\* current = this->first;

while (current->pNext != nullptr) {

current = current->pNext;

}

current->pNext = new Node();

}

size++;

}

//перегрузка операции ввода

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

int count;

cout << "Введите размер списка: ";

in >> count;

cout << "Введите элементы списка через Enter: ";

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

int data;

in >> data;

list.push\_back(data);

}

return in;

}

//перегрузка операции вывода

ostream& operator << (ostream& out, const List& list) {

if (list.first == nullptr) {

cout << "Список пуст!" << endl;

return out;

}

Node\* current = list.first;

while (current != nullptr) {

cout << current->data << " ";

current = current->pNext;

}

cout << endl;

return out;

}

main.cpp

/\* Вариант 12.

Класс-контейнер СПИСОК с ключевыми значениями типа int.

Реализовать операции:

[] – доступа по индексу;

() – определение размера списка;

+ число – добавляет константу ко всем элементам списка;

++ - добавление элемента в конец списка.

Реализация 1 - Информация об исключительных ситуациях

передается с помощью стандартного типа данных. \*/

#include "List.h"

int main() {

setlocale(LC\_ALL, "RU");

try {

int s1, s2, ind;

cout << "Введите размер первого списка: ";

cin >> s1;

List list1(s1);

cout << "Первый список: " << list1;

cout << "Введите индекс элемента для вывода: ";

cin >> ind;

cout << "Выбранный элемент первого списка: " << list1[ind] << endl;

cout << "Введите размер второго списка: ";

cin >> s2;

List list2(s2);

cout << "Второй список: " << list2 << endl;

list2 + 5;

cout << "Второй список после добавления константы: " << list2 << endl;

++list2;

cout << "Второй список с добавлением элемента: " << list2 << endl;

}

catch (int) {

cout << "Error, incorrect data!" << endl;

}

catch (exception& err) {

cout << err.what();

}

return 0;

}

*Реализация 3:*

List.h

#pragma once

#include <iostream>

using namespace std;

const int MAX\_SIZE = 5;

class Node

{

public:

int data = 0;

Node\* pNext = nullptr;

Node(int data = 0, Node\* pNext = nullptr) {

this->data = data;

this->pNext = pNext;

}

};

class List

{

private:

int size;

Node\* first;

Node\* last;

public:

List();

List(int);

List(const List&);

~List();

void push\_back(int);

void pop\_front();

List& operator = (const List&);

int& operator [] (int);

int operator () ();

void operator + (const int);

void operator ++ ();

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

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

};

List.cpp

#include "List.h"

#include "Error.h"

//конструктор по умолчанию

List::List() {

size = 0;

first = nullptr;

last = nullptr;

}

//конструктор с параметрами

List::List(int s) {

if (s > MAX\_SIZE) {

throw MaxSizeError(); //обработка превышения максимального размера

}

size = s;

first = nullptr;

last = nullptr;

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

int rnd = rand() % 10;

Node\* addNode = new Node(rnd);

if (last == nullptr) {

first = addNode;

last = addNode;

}

else {

last->pNext = addNode;

last = addNode;

}

}

}

//конструктор копирования

List::List(const List& list) {

size = 0;

first = nullptr;

last = nullptr;

Node\* current = list.first;

while (current != nullptr) {

Node\* addNode = new Node();

addNode->data = current->data;

addNode->pNext = nullptr;

if (last == nullptr) {

first = addNode;

last = addNode;

}

else {

last->pNext = addNode;

last = addNode;

}

current = current->pNext;

size++;

}

}

//деструктор

List::~List() {

while (size > 0) {

pop\_front();

}

}

//функция для добавления элемента в конец списка

void List::push\_back(int data) {

if (size >= MAX\_SIZE) {

throw MaxSizeError(); //обработка превышения максимального размера

}

if (first == nullptr) {

first = new Node(data);

}

else {

Node\* current = this->first;

while (current->pNext != nullptr) {

current = current->pNext;

}

current->pNext = new Node(data);

}

size++;

}

//функция для удаления элемента из начала списка

void List::pop\_front() {

if (first == nullptr) {

throw EmptySizeError(); //обработка случая с пустым списком

}

Node\* temp = first;

first = first->pNext;

delete temp;

size--;

}

//перегрузка оператора присваивания

List& List::operator = (const List& other) {

if (this == &other) {

return \*this;

}

while (first != nullptr) {

Node\* temp = first;

first = first->pNext;

delete temp;

}

Node\* currentOther = other.first;

while (currentOther != nullptr) {

push\_back(currentOther->data);

currentOther = currentOther->pNext;

}

return \*this;

}

//перегрузка квадратных скобок

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

if (index < 0)

throw IndexErrorMin(); //обработка индекса меньше нуля

if (index >= size)

throw IndexErrorMax(); //обработка индекса больше размера

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

int count = 0;

Node\* current = this->first;

while (current != nullptr) {

if (count == index) {

return current->data;

}

current = current->pNext;

count++;

}

}

}

//перегрузка круглых скобок - получение размера списка

int List::operator () () {

return size;

}

//перегрузка оператора сложения

void List::operator + (const int add) {

Node\* current = this->first;

while (current != nullptr) {

current->data = current->data + add;

current = current->pNext;

}

}

//перегрузка инкремента

void List::operator ++ () {

if (size >= MAX\_SIZE) {

throw MaxSizeError(); //обработка превышения максимального размера

}

if (first == nullptr) {

first = new Node();

}

else {

Node\* current = this->first;

while (current->pNext != nullptr) {

current = current->pNext;

}

current->pNext = new Node();

}

size++;

}

//перегрузка операции ввода

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

int count;

cout << "Введите размер списка: ";

in >> count;

cout << "Введите элементы списка через Enter: ";

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

int data;

in >> data;

list.push\_back(data);

}

return in;

}

//перегрузка операции вывода

ostream& operator << (ostream& out, const List& list) {

if (list.first == nullptr) {

cout << "Список пуст!" << endl;

return out;

}

Node\* current = list.first;

while (current != nullptr) {

cout << current->data << " ";

current = current->pNext;

}

cout << endl;

return out;

}

Error.h

#pragma once

#include <string>

#include <iostream>

using namespace std;

class Error

{

public:

virtual void what() {};

};

class SizeError : public Error

{

protected:

string msg;

public:

SizeError() {

msg = "Ошибка размера списка! ";

}

virtual void what() {

cout << msg;

}

};

class MaxSizeError : public SizeError

{

protected:

string msg\_;

public:

MaxSizeError() {

SizeError();

msg\_ = "Размер списка больше максимального!\n";

}

virtual void what() {

cout << msg << msg\_;

}

};

class EmptySizeError : public SizeError

{

protected:

string msg\_;

public:

EmptySizeError() {

SizeError();

msg\_ = "Список пустой!\n";

}

virtual void what() {

cout << msg << msg\_;

}

};

class IndexError : public Error

{

protected:

string msg;

public:

IndexError() {

msg = "Ошибка идекса! ";

}

virtual void what() {

cout << msg;

}

};

class IndexErrorMin : public IndexError

{

protected:

string msg\_;

public:

IndexErrorMin() {

IndexError();

msg\_ = "Индекс меньше нуля!\n";

}

virtual void what() {

cout << msg << msg\_;

}

};

class IndexErrorMax : public IndexError

{

protected:

string msg\_;

public:

IndexErrorMax() {

IndexError();

msg\_ = "Индекс за пределами списка!\n";

}

virtual void what() {

cout << msg << msg\_;

}

};

main.cpp

/\* Вариант 12.

Класс-контейнер СПИСОК с ключевыми значениями типа int.

Реализовать операции:

[] – доступа по индексу;

() – определение размера списка;

+ число – добавляет константу ко всем элементам списка;

++ - добавление элемента в конец списка.

Реализация 3 - Информация об исключительных ситуациях передается

с помощью иерархии пользовательских классов. \*/

#include "List.h"

#include "Error.h"

int main() {

setlocale(LC\_ALL, "RU");

try {

int s1, s2, ind;

cout << "Введите размер первого списка: ";

cin >> s1;

List list1(s1);

cout << "Первый список: " << list1;

cout << "Введите индекс элемента для вывода: ";

cin >> ind;

cout << "Выбранный элемент первого списка: " << list1[ind] << endl;

cout << "Введите размер второго списка: ";

cin >> s2;

List list2(s2);

cout << "Второй список: " << list2 << endl;

list2 + 5;

cout << "Второй список после добавления константы: " << list2 << endl;

++list2;

cout << "Второй список с добавлением элемента: " << list2 << endl;

}

catch (Error& err) {

err.what();

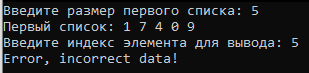
}

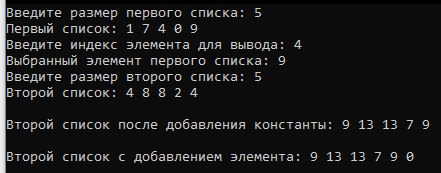
return 0;

}

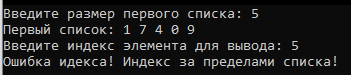
**Результат работы программы**

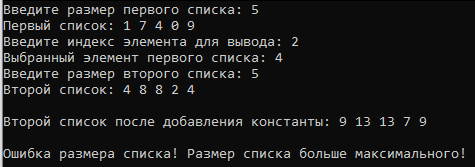
*Реализация 1:*

**

**

*Реализация 3:*





**Контрольные вопросы**

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

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

1. *На какие части исключения позволяют разделить вычислительный процесс? Достоинства такого подхода?*
2. Обнаружение аварийной ситуации (неизвестно, как обрабатывать).
3. Обработка аварийной ситуации (неизвестно, где возникла).

*Достоинства:*

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

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

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

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

try { операторы }

*Свойства блока try*:

- Связан с одним или несколькими операторами catch.

- Все переменные объявленные внутри try являются локальными.

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

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

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

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

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

1. (тип имя)
2. (тип)
3. (…)

Формы 1 и 2 обрабатывают конкретные исключения, а форма 3 перехватывает все исключения, такую ловушку надо помещать последней, тогда она будет обрабатывать все исключения, которые еще не были обработаны.

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

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

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

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

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

class Child\_Error: public Error{};

class Parents\_Error: public Error{};

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

Только исключения типа int и double.

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

Исключения абсолютно любого типа.

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

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

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

*Без спецификации исключений:*

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

int S, P;

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

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

return S;

}

*Со спецификатором throw:*

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

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

double P;

double S;

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

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

return S;

}

*C собственным реализованным исключением:*

void F() {

exit(1);

}

void set\_terminate() {

F();

}

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

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

double P;

double S;

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

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

return S;

}

*С конкретной спецификацией с подходящим стандартным исключением:*

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

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

double P;

double S;

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

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

return S;

}