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**

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

Семестр: 2

Выполнил студент ИВТ-22-2б:

Мельников Глеб Владимирович

(дата, подпись)

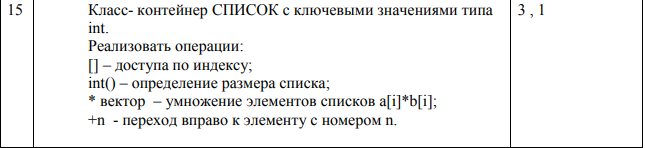
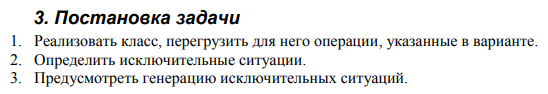
Проверила:

Полякова Ольга Андреевна

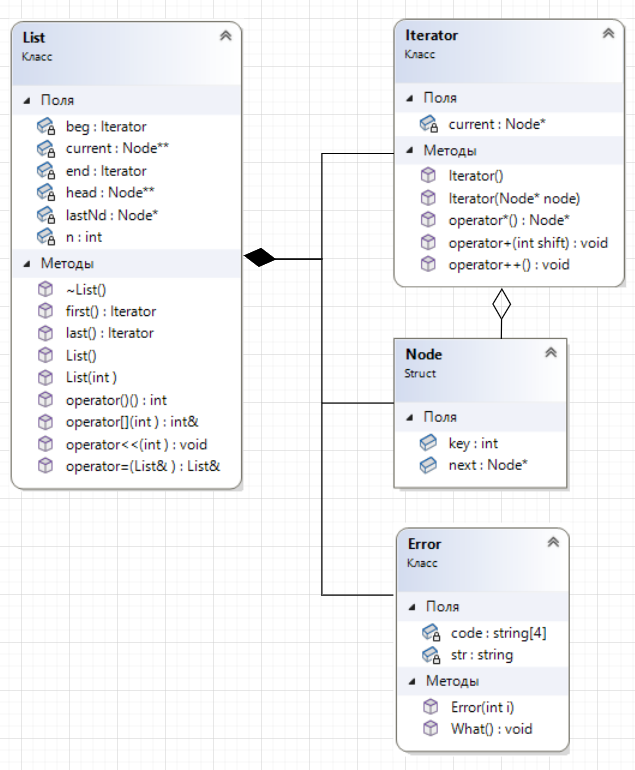
(дата, подпись)

Пермь 2023

**Задание**



**UML-Диаграмма**



**Программное решение**

**Error.h**

#pragma once

#pragma once

#include <string>

#include <iostream>

using namespace std;

class Error {

private:

string code[4] = { "Ошибка: нельзя сместиться, выход за пределы списка!", "Ошибка: невозможно получить первый элемент пустого списка!", "Ошибка: невозможно получить последний элемент пустого списка!", "Ошибка: Индекс за пределами диапазона!" };

string str;

public:

Error(int i) {

str = code[i];

}

void What() {

cout << str << '\n';

}

};

**List.h**

#pragma once

#include <string>

#include <iostream>

using namespace std;

#include "Error.h"

struct Node {

int key;

Node\* next;

};

class Iterator {

private:

Node\* current;

friend class List;

public:

Iterator() {

current = NULL;

};

Iterator(Node\* node) {

current = node;

};

void operator ++ () {

if (current == NULL) {

throw Error(0);

}

current = current->next;

}

Node\* operator \*() const {

return current;

}

void operator + (int shift) {

Node\* tmp = current;

int i = 0;

while (i < shift && current != NULL && current->next != NULL) {

current = current->next;

i++;

}

if (i < shift) {

current = tmp;

throw Error(0);

}

}

};

class List {

public:

List() {};

List(int);

~List();

List& operator = (List&);

int& operator[] (int);

int operator () ();

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

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

Iterator first();

Iterator last();

void operator << (int);

private:

Node\* lastNd, \* current, \* head;

int n = 0;

Iterator beg, end;

};

**List.cpp**

#include "List.h"

List::List(int count) {

n = count;

head = new Node;

head->key = 0;

lastNd = head;

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

current = new Node;

current->key = 0;

lastNd->next = current;

lastNd = current;

}

lastNd->next = NULL;

}

List::~List() {

lastNd = head;

while (lastNd != NULL) {

current = lastNd->next;

delete lastNd;

lastNd = current;

}

n = 0;

}

List& List::operator=(List& l) {

if (this != &l) {

if (this != 0) {

lastNd = head;

while (lastNd != NULL) {

current = lastNd->next;

delete lastNd;

lastNd = current;

}

n = 0;

}

lastNd = head = new Node;

l.lastNd = l.head->next;/////

lastNd->key = l.head->key;

while (l.lastNd != NULL) {

lastNd->next = new Node;

lastNd->next->key = l.lastNd->key;

l.lastNd = l.lastNd->next;

lastNd = lastNd->next;

}

lastNd->next = NULL;

n = l.n;

}

return \*this;

}

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

if (index >= n) {

throw Error(3);

}

lastNd = head;

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

lastNd = lastNd->next;

}

return lastNd->key;

}

int List::operator () () {

return n;

}

Iterator List::first() {

if (!n) {

throw Error(1);

}

beg.current = head;

return beg;

}

Iterator List::last() {

if (!n) {

throw Error(2);

}

lastNd = head;

while (lastNd->next != NULL) {

lastNd = lastNd->next;

}

end.current = lastNd;

return end;

}

ostream& operator<<(ostream& out, List& l) {

if (l.n) {

l.lastNd = l.head;

while (l.lastNd != NULL) {

out << l.lastNd->key << ' ';

l.lastNd = l.lastNd->next;

}

}

else {

out << "Список пуст!";

}

return out;

}

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

l.lastNd = l.head;

while (l.lastNd != NULL) {

in >> l.lastNd->key;

l.lastNd = l.lastNd->next;

}

return in;

}

void List::operator << (int number) {

if (head == NULL) {

head = new Node;

head->key = number;

head->next = NULL;

}

else {

lastNd = head;

while (lastNd->next != NULL) {

lastNd = lastNd->next;

}

lastNd->next = new Node;

lastNd->next->key = number;

lastNd = lastNd->next;

lastNd->next = NULL;

}

n += 1;

}

**OOPLab9.cpp**

#include <iostream>

using namespace std;

#include "List.h"

#include "Error.h"

int main() {

system("chcp 1251 > NULL");

try {

List a(2);

List b;

cout << a << '\n';

cout << b << '\n';

cout << b[0];//Ошибка

Iterator i = b.first();//Ошибка

Iterator j = b.last();//Ошибка

Iterator k = a.last();

++k;

++k;//Ошибка

k + 1;//Ошибка

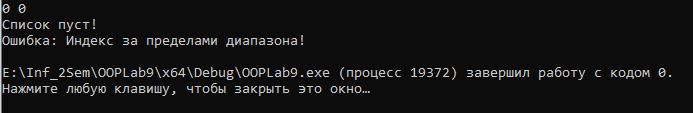
}

catch (Error& m\_Error) {

m\_Error.What();

}

return 0;

}

**Контрольные вопросы**

1. Что представляет собой исключение в С++?

Ответ: исключение — объект, генерируемый системой при возникновении исключительной ситуации.

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

Исключения позволяют разделить вычислительный процесс на 2 части:

* Обнаружение аварийной ситуации
* Обработка аварийной ситуации.

Достоинства:

* Удобство использования в программе из нескольких модулей
* Отсутствие требования возвращать значение в вызывающую функцию

1. Какой оператор используется для генерации исключительной ситуации?

throw

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

Контролируемый блок try нужен для проверки помещённого кода на исключения. В нем описывается оператор throw, который передает исключения в блок обработки исключения catch

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

Блок catch – обработчик исключения. Он нужен для перехватки исключения, которое кидает оператор throw.

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

Спецификация исключения может быть выражена типом и именем, типом и остальными исключениями. Первая и вторая формы перехватывают исключения выбранного типа. Третья форма обрабатывает все исключения, которые не попадают по выделенный тип исключения

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

exception

1. Каким образом можно создать собственную иерархию исключений?

Описанием собственных классов

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

int и double

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

Не может

1. В какой части программы может генерироваться исключение?

Внутри оператора try

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

Функцию реализовать в 4 вариантах:

* без спецификации исключений;

double Area (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 Area(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 Area(double a, double b, double c) {

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

throw 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 Error : public exception {

public:

virtual const char\* what() const throw() {

return "Invalid triangle sides";

}

};

double Area(double a, double b, double c) {

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

throw Error;

}

double p = (a + b + c) / 2;

return sqrt(p \* (p - a) \* (p - b) \* (p - c));

}