Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное учреждение высшего образования «Пермский национальный исследовательский политехнический университет»

ПНИПУ

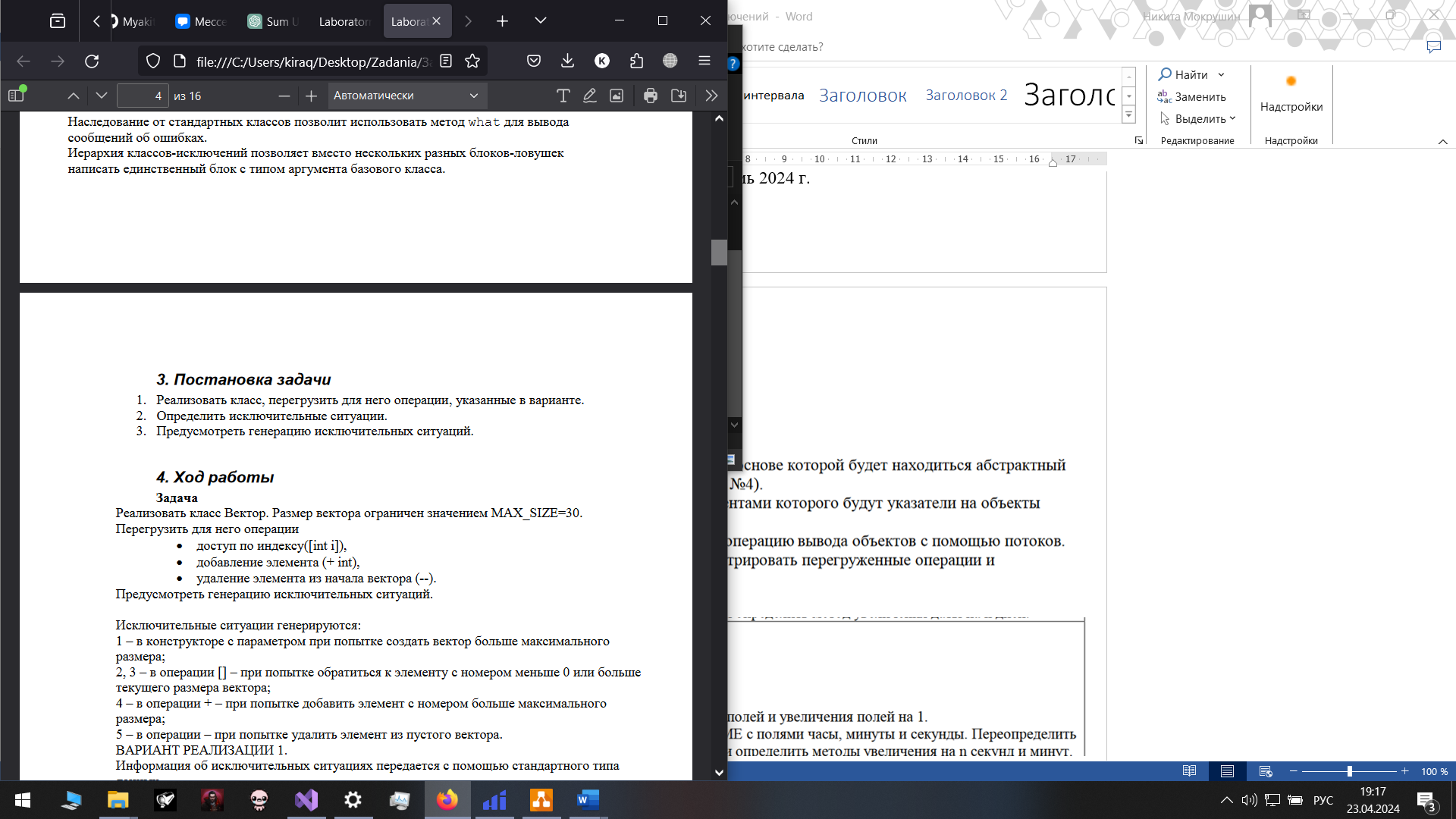
Лабораторная работа  
«Обработка исключений»

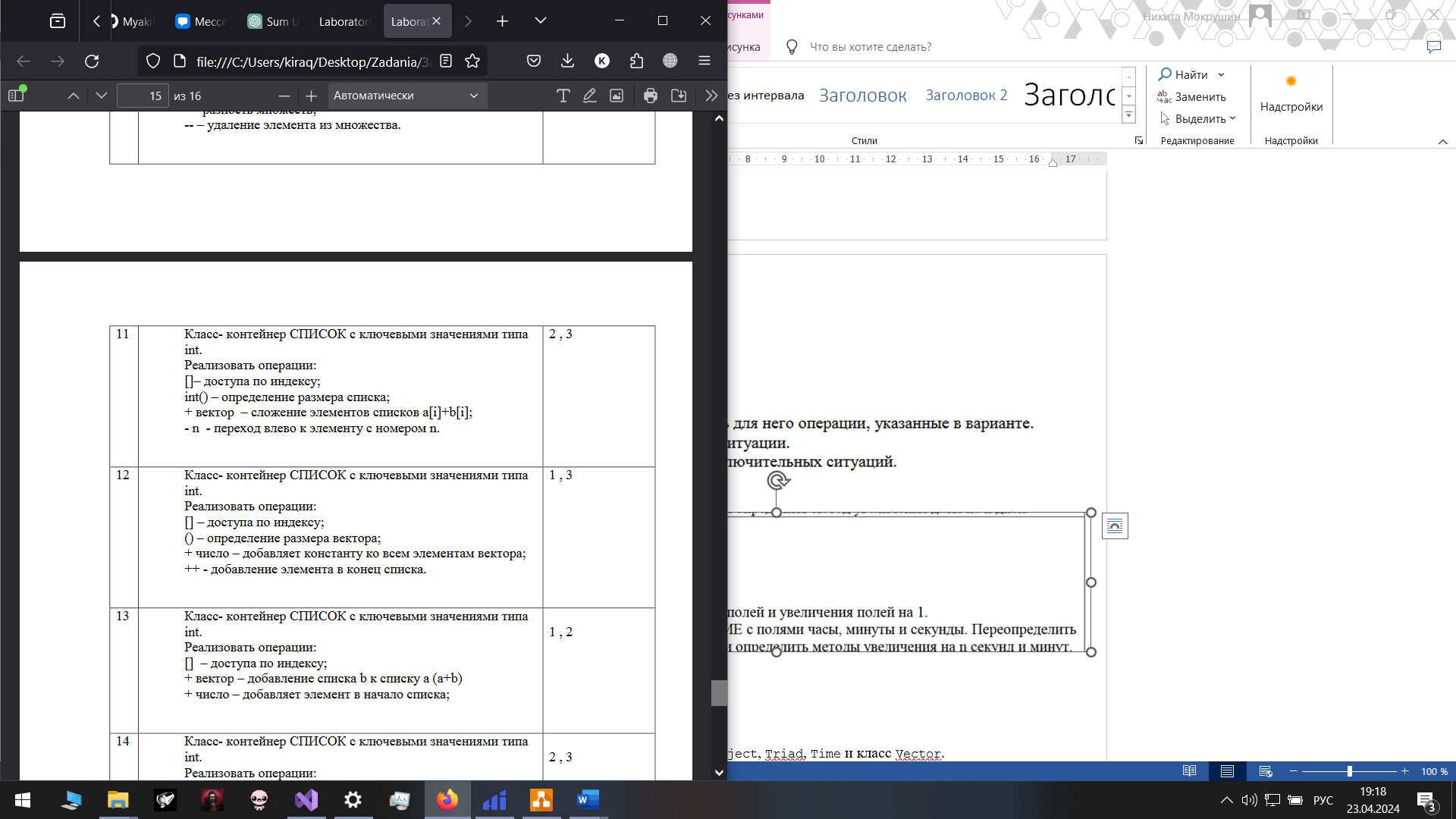
Выполнил:   
студент группы РИС-23-1б   
Мокрушин Никита Дмитриевич

Проверила:   
доцент кафедры ИТАС   
О.А. Полякова

Пермь 2024 г.

Постановка задачи:





Анализ задачи:

 **труктура Node**: Определяет узел списка, содержащий указатели на предыдущий и следующий элементы, а также данные элемента.

 **Класс Error и его наследники Error\_ind\_min, Error\_ind\_max**: Определяют классы исключений для обработки ошибок с индексами списка.

 **Класс Iterator**: Представляет итератор для обхода списка. Позволяет перемещаться по элементам списка.

 **Класс List**:

* Содержит указатели на начало и конец списка, а также на итераторы.
* Реализует операции добавления и удаления элементов, доступа к первому и последнему элементам, проверки на пустоту, доступа к элементам по индексу, перегрузку операторов для работы с элементами списка.
* Имеет конструкторы с параметрами и копирования, деструктор, операторы присваивания и сложения.

 **Функции main**:

* Создает список, заполняет его элементами и выводит информацию о списке.

Код:

#include <iostream>

using namespace std;

struct Node

{

Node\* prev = nullptr;

Node\* next = nullptr;

int data;

};

class Error

{

public:

void what() { cout << message; }

protected:

string message;

};

class Error\_ind\_min : public Error

{

public:

Error\_ind\_min() { message = "индекс меньше нуля"; }

};

class Error\_ind\_max : public Error

{

public:

Error\_ind\_max() { message = "индекс больше максимума"; }

};

class Iterator

{

friend class List;

public:

Iterator() { elem = nullptr; };

Iterator(const Iterator& i) { elem = i.elem; };

Iterator operator=(const Iterator& a) { elem = a.elem; return \*this; }

bool operator==(const Iterator& i) { return elem == i.elem; }

bool operator!=(const Iterator& i) { return elem != i.elem; }

Iterator& operator++() { elem = elem->next; return \*this; }

Iterator& operator--() { elem = elem->prev; return \*this; }

Iterator& operator+(const int& a)

{

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

{

elem = elem->next;

}

}

Iterator& operator-(const int& a)

{

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

{

elem = elem->prev;

}

}

int& operator\*() { return(elem->data); }

private:

Node\* elem;

};

class List

{

public:

List(int s, int k = 0);

List(const List& a);

~List();

int front();

int back();

void pushback(int data);

void pushfront(int data);

void popback();

void popfront();

bool empty();

List& operator=(const List& a);

int& operator[](int index);

int& operator()();

List operator+(List& a);

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

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

Iterator first() { return beg; }

Iterator last() { return end; }

private:

int size;

Node\* head = nullptr;

Node\* tail = nullptr;

Iterator beg;

Iterator end;

};

List::List(int s, int k)

{

size = s;

Node\* node = new Node;

node->data = k;

head = node;

tail = node;

for (int i = 0; i < size - 1; i++)

{

Node\* node = new Node;

node->data = 0;

tail->next = node;

node->prev = tail;

tail = node;

}

tail->next = nullptr;

beg.elem = head;

end.elem = tail->next;

}

List::List(const List& a)

{

Node\* node = a.head;

while (node != nullptr)

{

pushback(node->data);

node = node->next;

}

beg = a.beg;

end = a.end;

}

List::~List()

{

Node\* curnode = head;

while (curnode != nullptr)

{

head = curnode->next;

delete curnode;

curnode = head;

}

}

void List::pushback(int data)

{

Node\* newnode = new Node;

newnode->data = data;

if (head == nullptr)

{

head = newnode;

tail = newnode;

++size;

end.elem = tail->next;

}

else

{

tail->next = newnode;

newnode->prev = tail;

tail = newnode;

++size;

end.elem = tail->next;

}

}

void List::pushfront(int data)

{

Node\* newnode = new Node;

newnode->data = data;

if (head == nullptr)

{

head = newnode;

tail = newnode;

++size;

beg.elem = head;

}

else

{

head->prev = newnode;

newnode->next = head;

head = newnode;

++size;

beg.elem = head;

}

}

int List::front()

{

return head->data;

}

int List::back()

{

return tail->data;

}

void List::popback()

{

if (head != nullptr)

{

Node\* curnode = tail;

tail = curnode->prev;

delete curnode;

tail->next = nullptr;

--size;

end.elem = tail->next;

}

}

void List::popfront()

{

if (head != nullptr)

{

Node\* curnode = head;

head = curnode->next;

delete curnode;

head->prev = nullptr;

--size;

beg.elem = head;

}

else

{

cout << "Список пуст";

}

}

bool List::empty()

{

return size == 0;

}

int& List::operator()()

{

return size;

}

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

{

if (index < 0)throw Error\_ind\_min();

if (index >= size)throw Error\_ind\_max();

Node\* cur = head;

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

{

cur = cur->next;

}

return cur->data;

}

List List::operator+(List& a)

{

int sizetmp = ((size > a.size) ? a.size : size);

List temp(sizetmp);

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

{

temp[i] = (\*this)[i] + a[i];

}

return temp;

}

ostream& operator<<(ostream& os, const List& a)

{

os << endl << "--------Вывод элементов списка-------" << endl;

Node\* cur = a.head;

while (cur != nullptr)

{

cout << cur->data << ' ';

cur = cur->next;

}

os << endl << "-----Вывод элементов списка окончен-----" << endl;

return os;

}

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

{

cout << endl << "--------Ввод элементов списка-------" << endl;

Node\* cur = a.head;

while (cur != nullptr)

{

in >> cur->data;

cur = cur->next;

}

cout << endl << "-----Ввод элементов списка окончен-----" << endl;

return in;

}

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

{

cout << "Оператор приравнивания" << endl;

if (this == &a)

{

return \*this;

}

Node\* node = head;

while (node != nullptr)

{

head = node->next;

delete node;

node = head;

--size;

}

Node\* curnode = a.head;

while (curnode != nullptr)

{

pushback(curnode->data);

curnode = curnode->next;

}

beg = a.beg;

end = a.end;

return \*this;

}

int main()

{

int count;

int x;

int index;

try

{

setlocale(0, "");

cout << "Введите кол-во элементов" << endl;

cin >> count;

if (count < 1) throw "Кол-ов должно быть больше одного";

List a(count);

cout << "Введите элементы списка а" << endl;

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

{

cin >> x;

a[i] = x;

}

cout << endl << a << endl;

cout << "Первый элемент списка a: " << \*(a.first()) << " / Последний элемент списка a: " << a.back() << endl;

cout << "Введите индекс для обращения к элементу списка а" << endl;

cin >> index;

cout << a[index] << endl;

}

catch (Error& a)

{

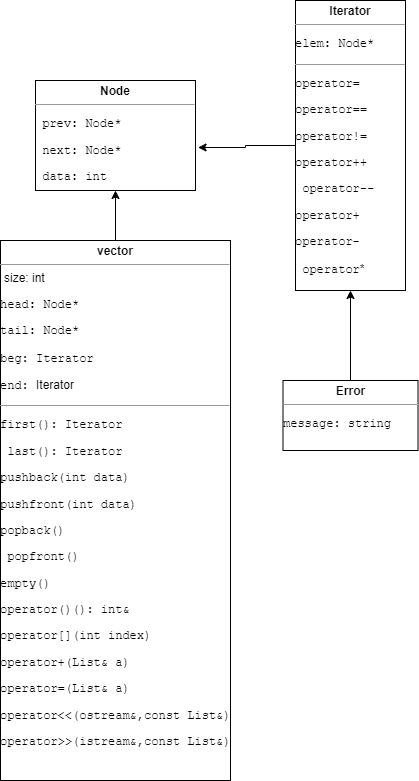
a.what();

}

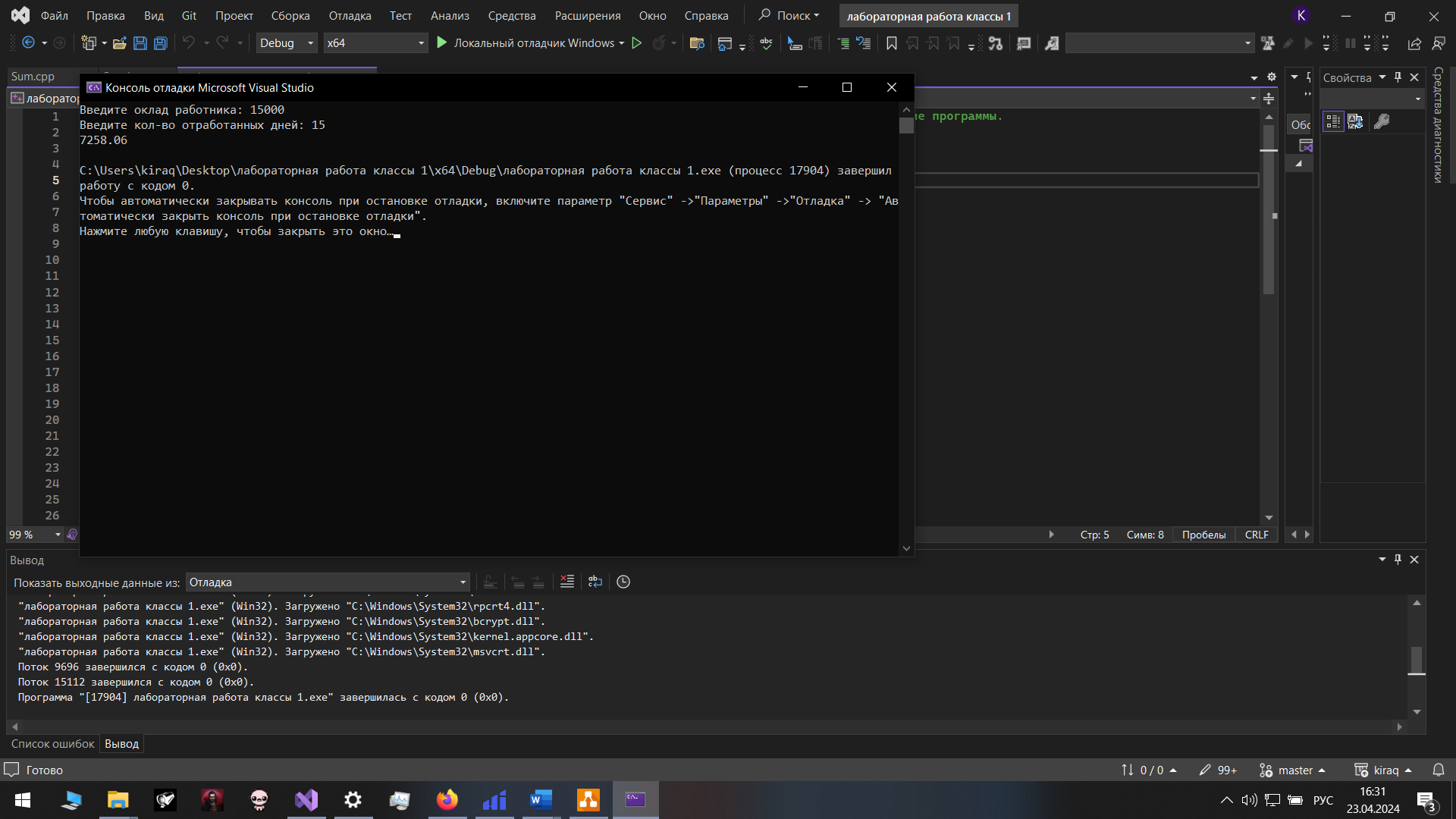
return 0;

}

UML Диаграммы:



Вывод:



программа работает корректно и выдаёт требуемый результат.  
Ответы на вопросы:

 **Исключение в C++** представляет собой объект, который передается во время выполнения программы, чтобы указать на ошибку или необычное состояние, которое может повлиять на нормальное продолжение работы программы.

 **Части исключения**: Разделяются на генерацию и обработку. Генерация исключения происходит в том месте кода, где возникает проблема или ошибка. Обработка исключения происходит в блоке кода, который реагирует на возникшее исключение. Преимущество такого подхода заключается в том, что он позволяет отделить код, который может вызвать ошибку, от кода, который обрабатывает эту ошибку, повышая читаемость и модульность программы.

 Для генерации исключительной ситуации используется оператор throw.

 **Контролируемый блок** - это участок кода, в котором может произойти исключение. Он нужен для того, чтобы определить область, в которой должна быть обработка исключения.

 **Секция-ловушка** - это блок кода, который обрабатывает исключение. Она нужна для того, чтобы предотвратить прекращение выполнения программы из-за исключения и выполнить необходимые действия по его обработке.

 Формы спецификации исключения в секции ловушке могут быть следующими:

* catch (...) - обрабатывает любое исключение, не указанное явно;
* catch (int ex) - обрабатывает исключение типа int;
* catch (double ex) - обрабатывает исключение типа double. Эти формы используются в ситуациях, когда необходимо обработать конкретные типы исключений или все исключения вообще.

 Стандартный класс std::exception можно использовать для создания собственной иерархии исключений.

 Для создания собственной иерархии исключений необходимо унаследоваться от класса std::exception или его производного класса и переопределить метод what().

 Функция f1() может принимать исключения типа int и double.

 Функция f1() не может принимать никаких исключений.

 Исключение может генерироваться в любой части программы, где возникает ошибка или необычное состояние, которое не может быть обработано локально.

 cpp

#include <iostream>

#include <cmath>

#include <stdexcept>

// Без спецификации исключений

double areaNoException(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 areaWithThrow(double a, double b, double c) throw() {

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

if (p \* (p - a) \* (p - b) \* (p - c) < 0)

throw std::domain\_error("Недопустимые значения сторон");

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

}

// С конкретной спецификацией с подходящим стандартным исключением

double areaWithStandardException(double a, double b, double c) throw(std::domain\_error) {

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

if (p \* (p - a) \* (p - b) \* (p - c) < 0)

throw std::domain\_error("Недопустимые значения сторон");

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

}

// Спецификация с собственным реализованным исключением

class InvalidTriangleException : public std::exception {

public:

const char\* what() const noexcept override {

return "Невозможно вычислить площадь: недопустимые значения сторон";

}

};

double areaWithCustomException(double a, double b, double c) throw(InvalidTriangleException) {

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

if (p \* (p - a) \* (p - b) \* (p - c) < 0)

throw InvalidTriangleException();

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

}

int main() {

double a, b, c;

std::cout << "Введите стороны треугольника: ";

std::cin >> a >> b >> c;

try {

double area = areaNoException(a, b, c);

std::cout << "Площадь треугольника: " << area << std::endl;

} catch (const std::exception& e) {

std::cout << "Ошибка: " << e.what() << std::endl;

}

return 0;

}