Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**О Т Ч Ё Т**

**по лабораторной работе №18.9**

Дисциплина: Основы теории алгоритмов и структуры данных.

Тема: “ Объектно-ориентированное программирование.

Обработка исключительных ситуаций.”

Вариант.12

Выполнила работу:

студент группы ИВТ-20-2Б

Ананина Арина Юрьевна

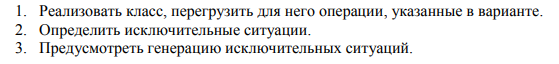
Проверила:

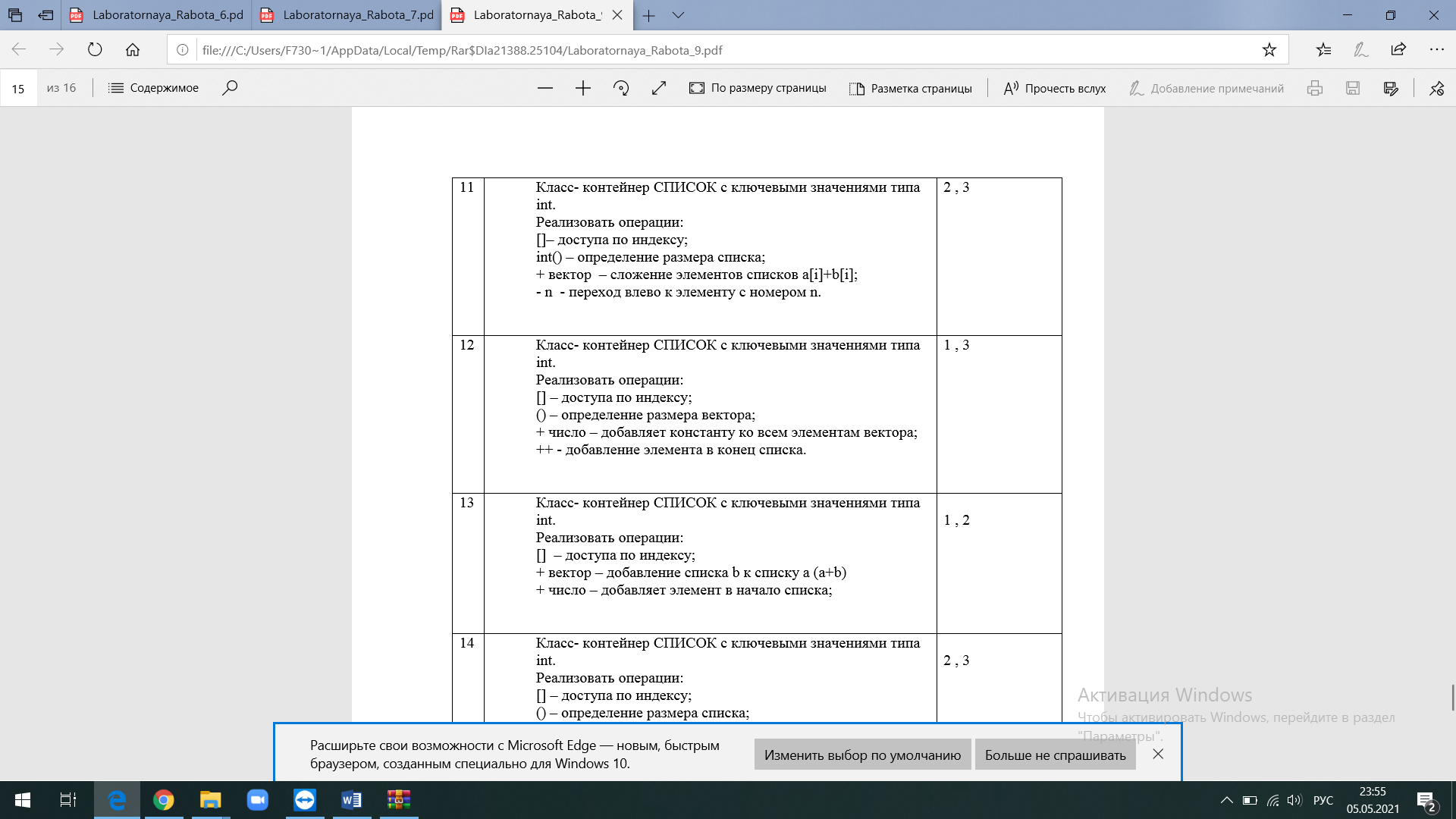
Доцент кафедры ИТАС

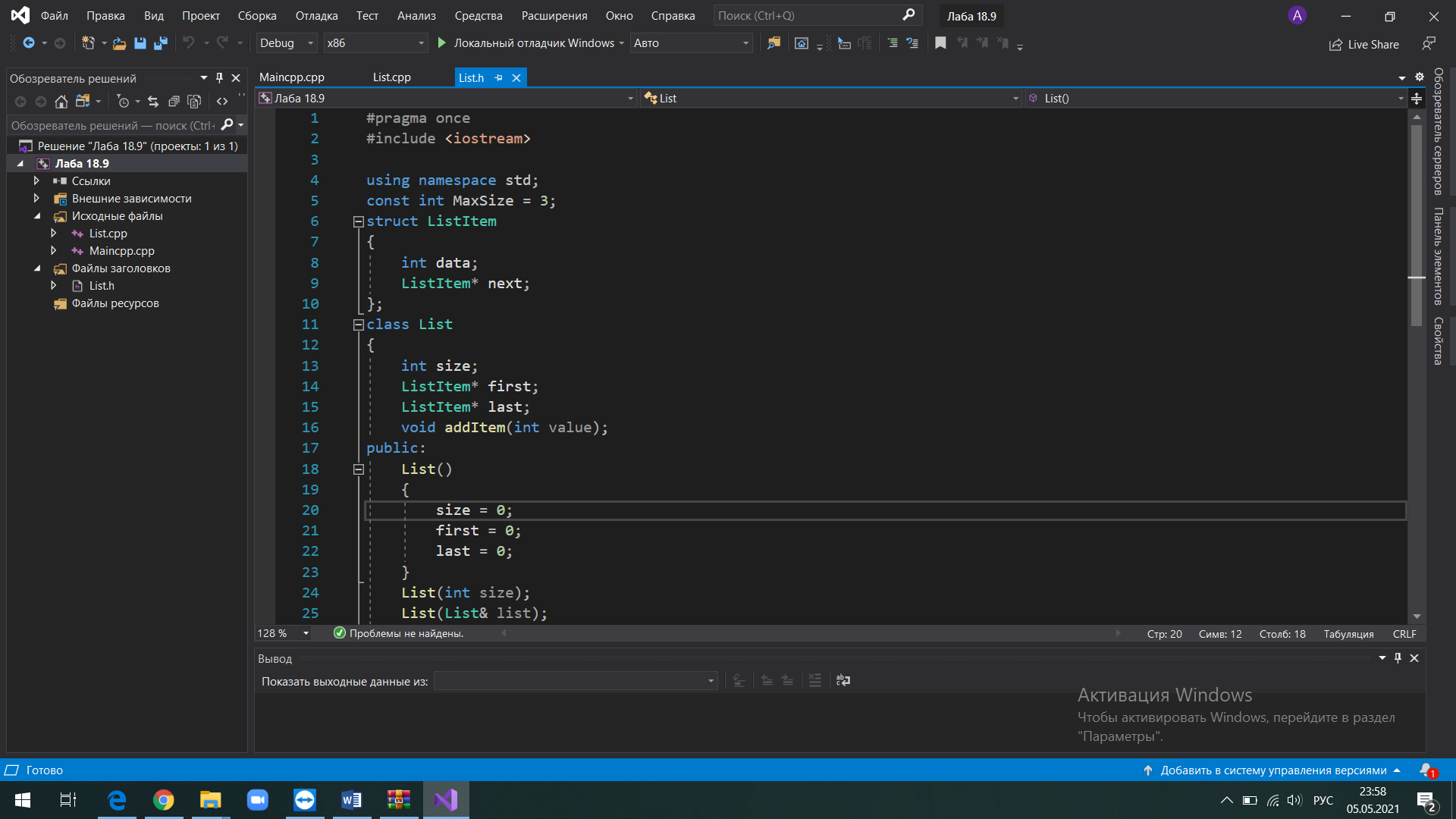
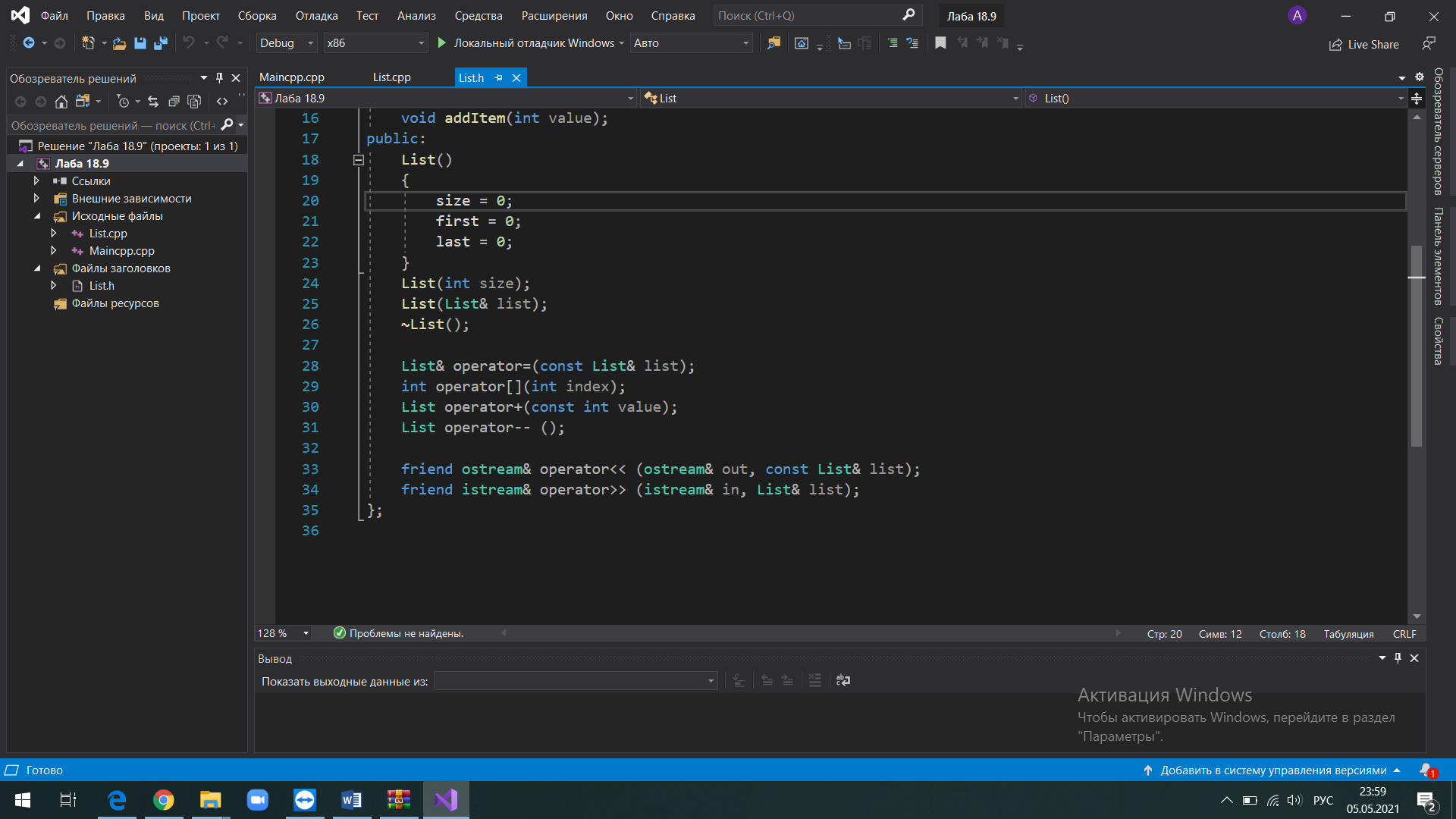
Полякова О.А.

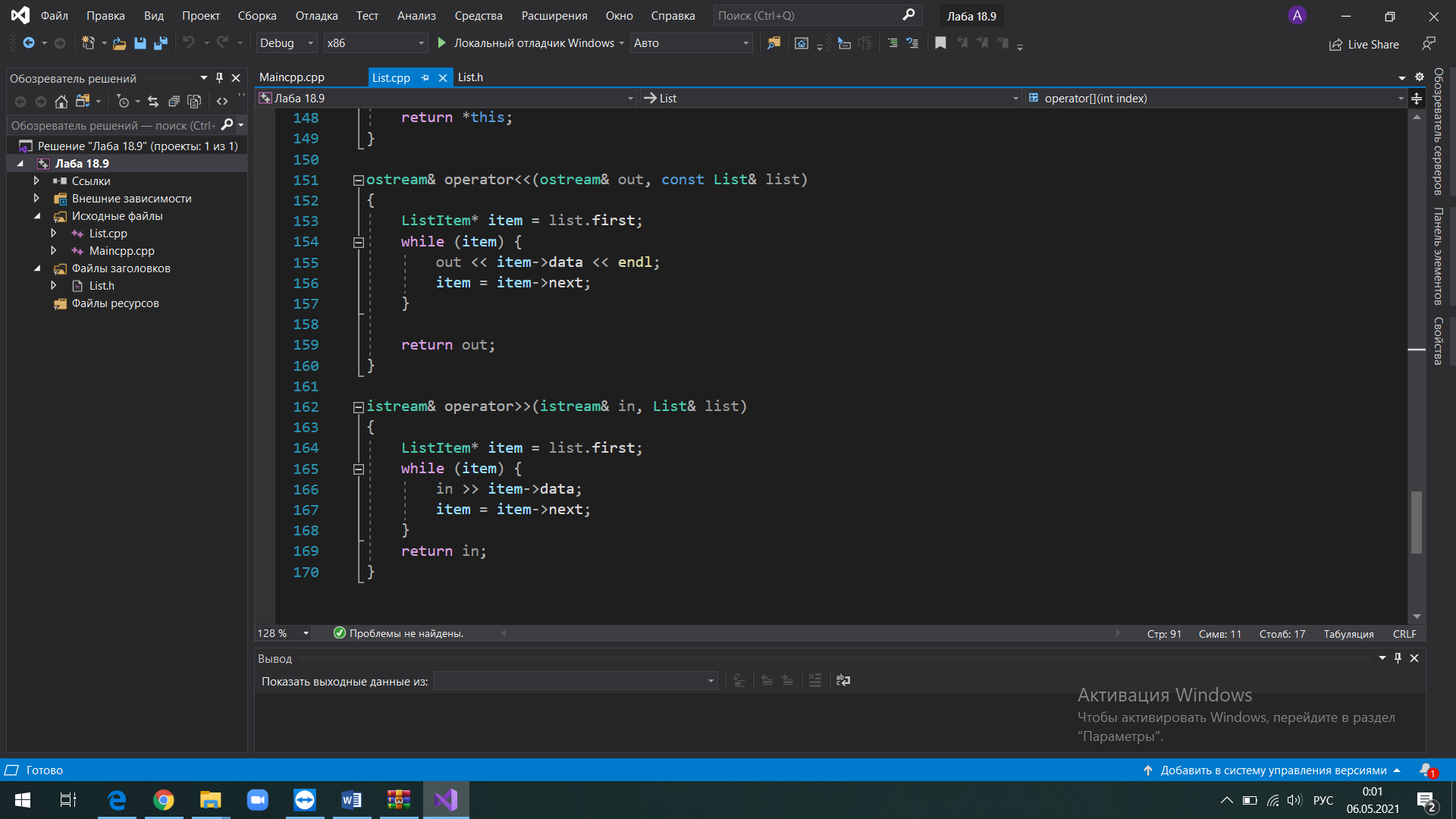
Пермь, 2021

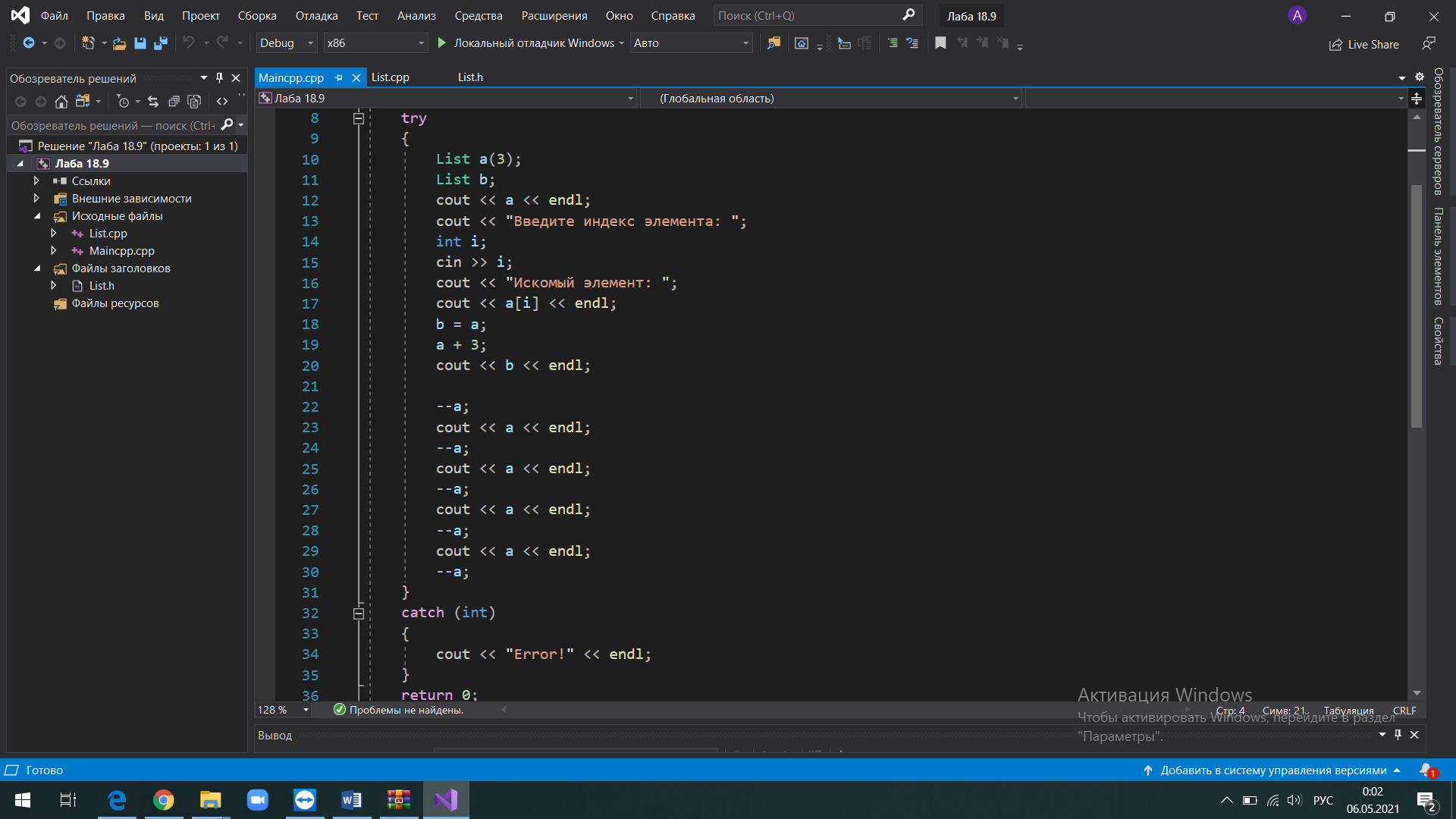
1. **Постановка задачи**





1. **Анализ задачи**
2. Для решения задачи необходимо:
   1. Использовать класс List из ЛР7
   2. Добавить обработку исключительных ситуаций
3. В ходе работы были использованы следующие типы данных:
   1. Класс List, реализацию см. в ЛР7:  
      
   2. Операторы:  
      
4. Для решения задачи данные были представлены в следующем виде:
   1. Данные хранятся в контейнере List.
5. Для операций ввода и вывода использовались следующие операторы и функции:
   1. Перегруженные операторы “>>” и “<<” для класса List, см. ЛР7.



1. Поставленные задачи будут решены следующими действиями:  
     
   1. В блоке try продемонстрированы все исключительные ситуации, они улавливаются блоком catch, выводится сообщение об ошибке, и программа завершается со значением -1, в противном же случае пользователю выводится сообщение о том, что ошибок с использованием класса List и Iterator не возникло.  
      
2. **Код программы**

**Файл Main.cpp:**  
#include "List.h"

#include <iostream>

#include <exception>

using namespace std;

int main()

{

setlocale(LC\_ALL, "rus");

try

{

List a(3);

List b;

cout << a << endl;

cout << "Введите индекс элемента: ";

int i;

cin >> i;

cout << "Искомый элемент: ";

cout << a[i] << endl;

b = a;

a + 3;

cout << b << endl;

--a;

cout << a << endl;

--a;

cout << a << endl;

--a;

cout << a << endl;

--a;

cout << a << endl;

--a;

}

catch (int)

{

cout << "Error!" << endl;

}

return 0;

}

**Файл List.h:**  
#pragma once

#include <iostream>

using namespace std;

const int MaxSize = 3;

struct ListItem

{

int data;

ListItem\* next;

};

class List

{

int size;

ListItem\* first;

ListItem\* last;

void addItem(int value);

public:

List()

{

size = 0;

first = 0;

last = 0;

}

List(int size);

List(List& list);

~List();

List& operator=(const List& list);

int operator[](int index);

List operator+(const int value);

List operator-- ();

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

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

};

**Файл List.cpp:**  
#include <iostream>

#include <exception>

#include "List.h"

using namespace std;

List::List(int size)

{

if (size > MaxSize)

{

throw 1;

}

this->last = 0;

this->first = 0;

this->size = 0;

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

int\* data = new int;

cout << "Введите "<< i<< " элемент: ";

cin >> (\*data);

this->addItem(\*data);

}

}

List::List(List& list)

{

this->last = 0;

this->first = 0;

this->size = 0;

\*this = list;

}

List::~List()

{

ListItem\* current = this->first;

while (current) {

ListItem\* rem = current;

current = current->next;

delete rem;

}

}

void List::addItem(int value)

{

ListItem\* item = new ListItem;

item->data = value;

item->next = 0;

if (!this->first) {

this->first = item;

}

if (this->last) {

this->last->next = item;

this->last = item;

}

else {

this->last = item;

}

this->size++;

}

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

{

if (this == &list)

{

return \*this;

}

// Очистка

if (this->size) {

this->~List();

this->size = 0;

this->last = 0;

this->first = 0;

}

// Наполнение

ListItem\* item = list.first;

while (item) {

this->addItem(item->data);

item = item->next;

}

return \*this;

}

int List::operator[](int index)

{

if (index < 0) {

throw 2;

}

if (index >= size) {

throw 3;

return this->first->data;

}

int i = 0;

ListItem\* item = this->first;

while (i < index ) {

item = item->next;

i++;

}

return item->data;

}

List List::operator+(const int value)

{

if (size + 1 == MaxSize)

{

throw 4;

}

ListItem\* item = new ListItem;

item->data = value;

item->next = 0;

if (!this->first) {

this->first = item;

}

if (this->last) {

this->last->next = item;

this->last = item;

}

else {

this->last = item;

}

this->size++;

return \*this;

}

List List::operator-- ()

{

if (size == 0)

{

throw 5;

}

if (size == 1)

{

ListItem\* current = this->first;

delete current;

this->first = 0;

this->last = 0;

this->size = 0;

return \*this;

}

ListItem\* current = this->first;

this->first = this->first->next;

this->size--;

delete current;

return \*this;

}

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

{

ListItem\* item = list.first;

while (item) {

out << item->data << endl;

item = item->next;

}

return out;

}

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

{

ListItem\* item = list.first;

while (item) {

in >> item->data;

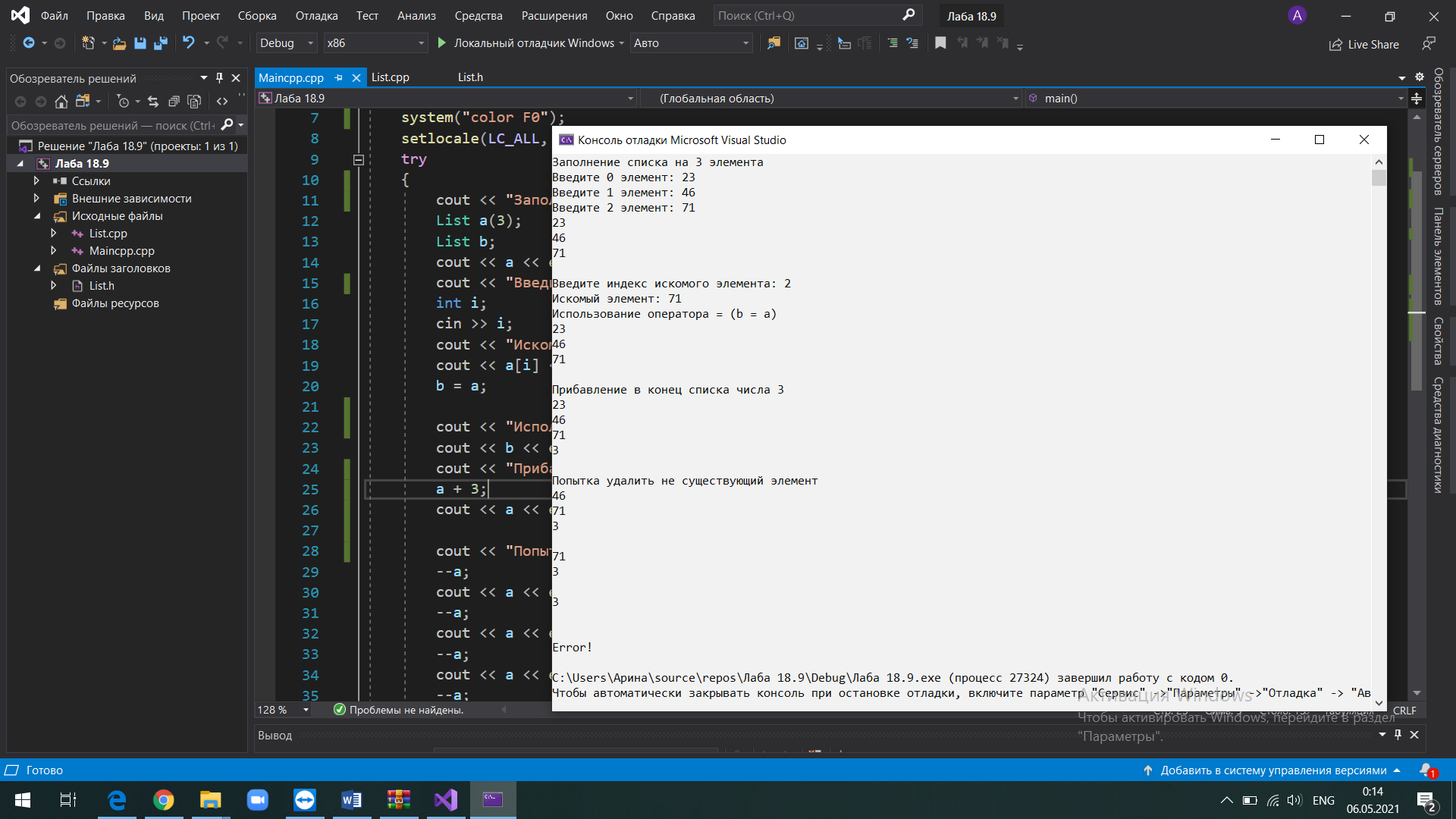
item = item->next;

}

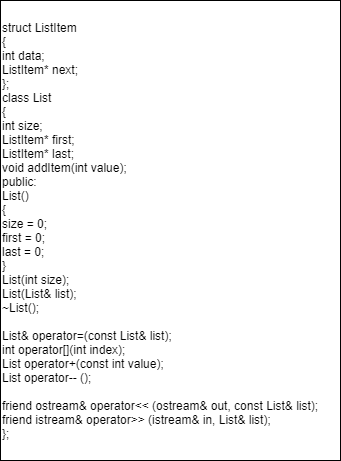
return in;

}

1. **Выполнение программы**



1. **Блок-схема**

****

1. **Ответы на вопросы**

