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 №18.7**

Дисциплина: Основы теории алгоритмов и структуры данных.

Тема: “Объектно-ориентированное программирование.

Шаблоны классов”

Вариант.12

Выполнила работу:

студент группы ИВТ-20-2Б

Ананина Арина Юрьевна

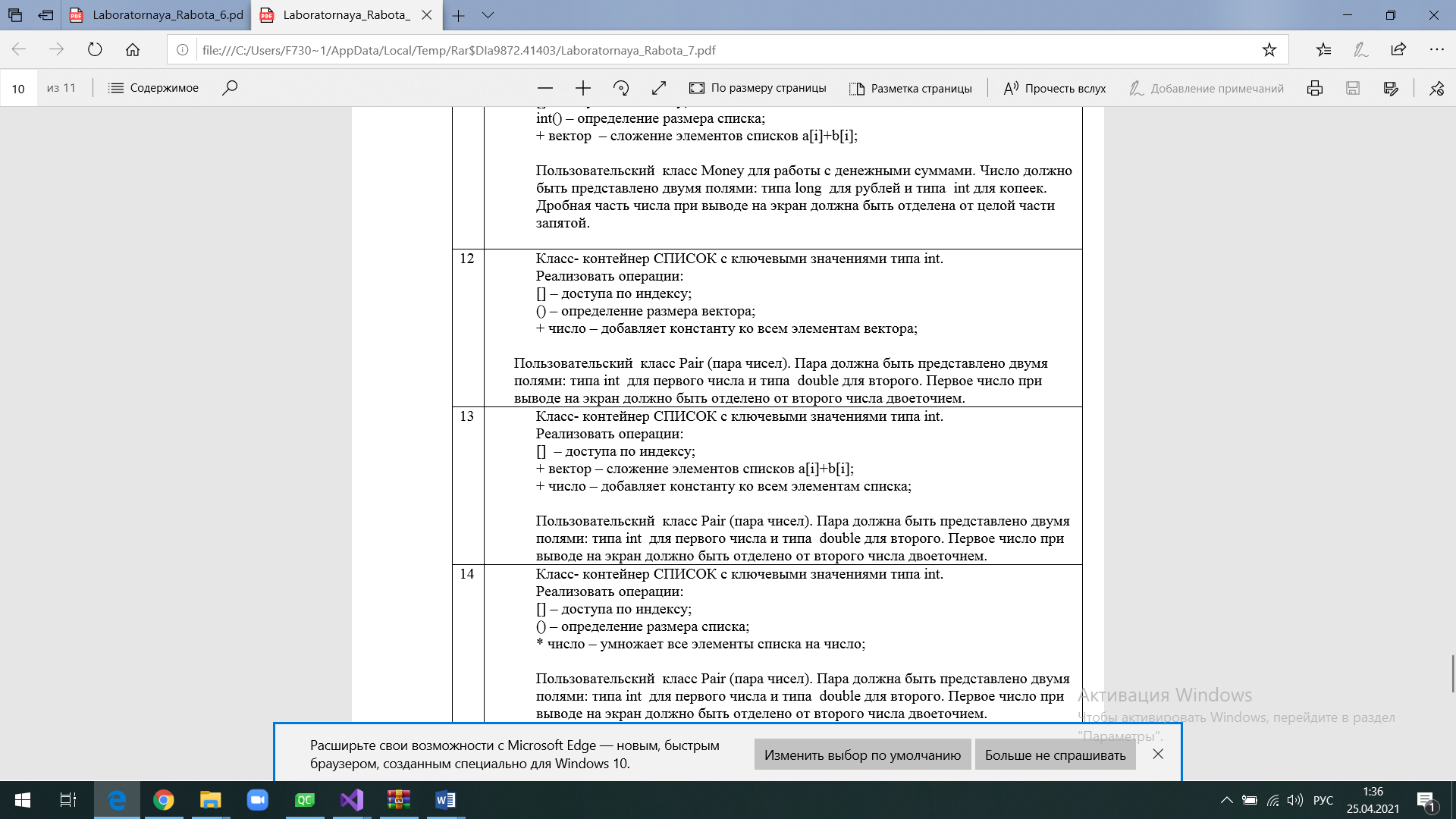
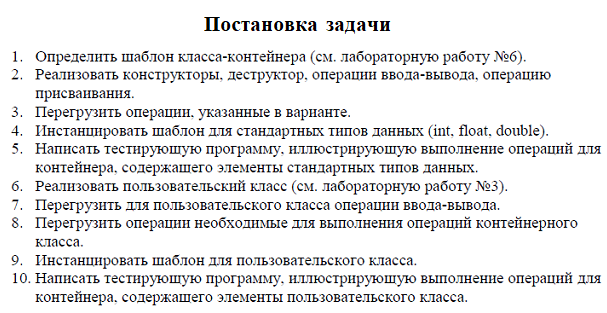
Проверила:

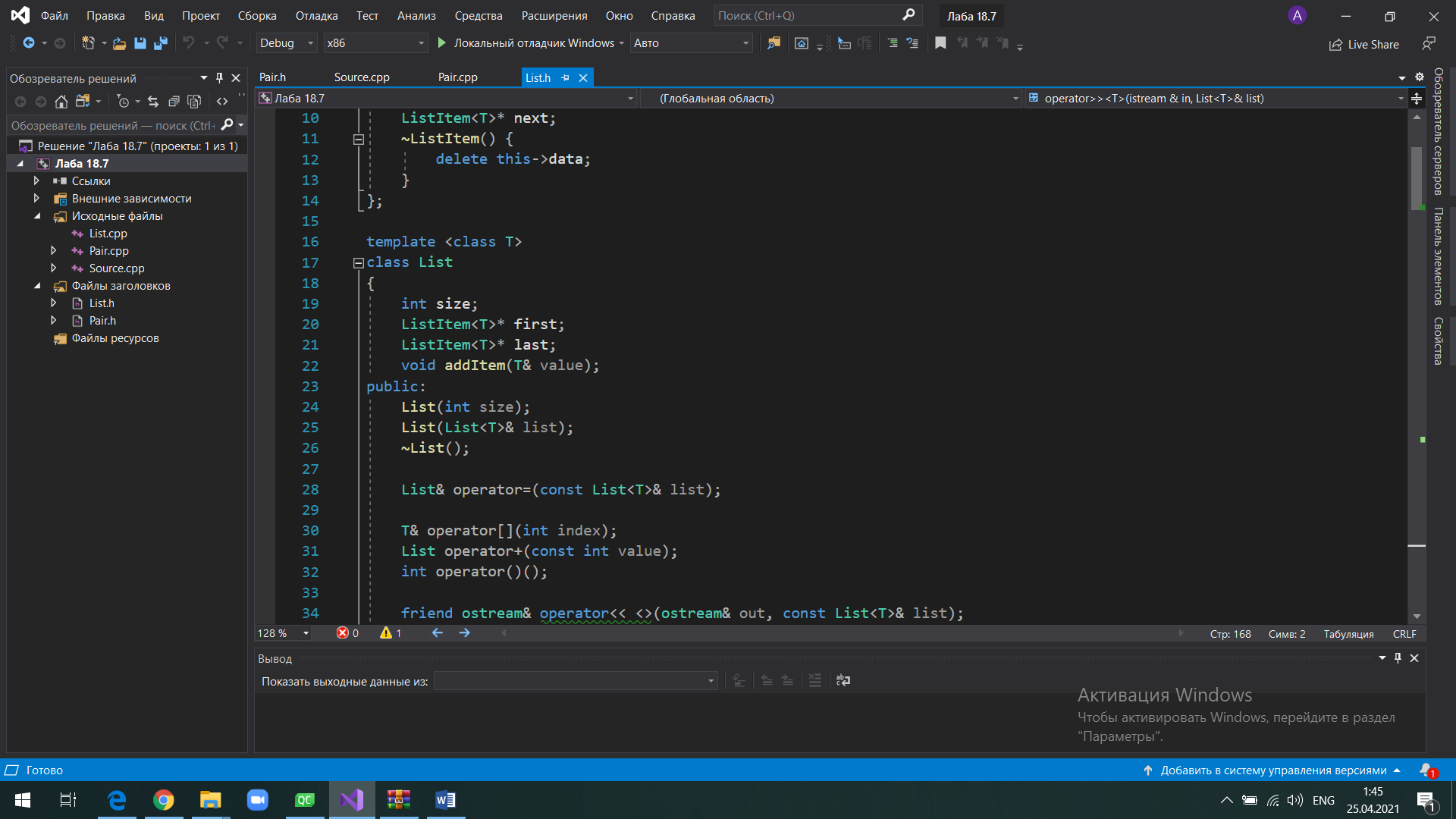
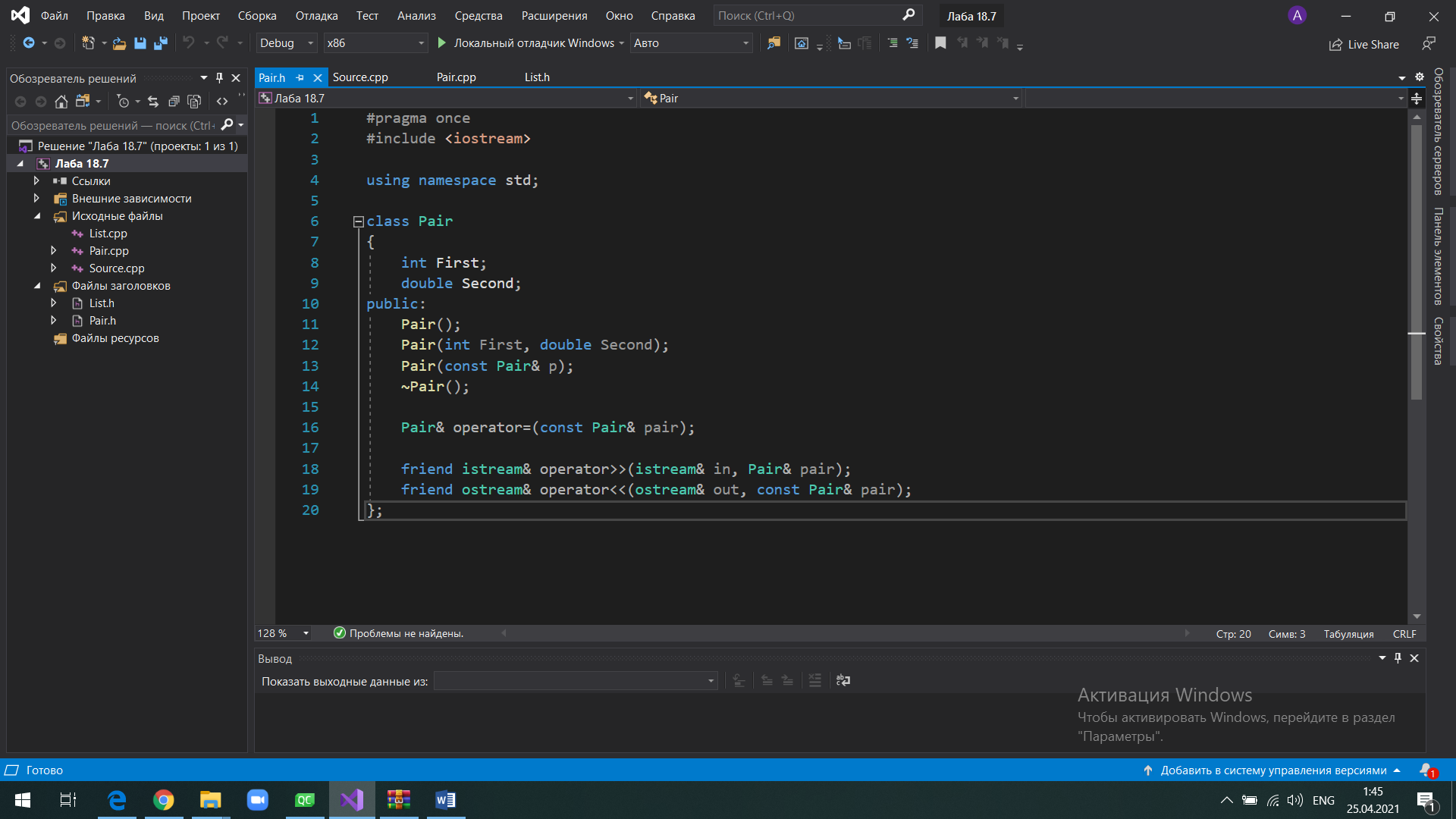
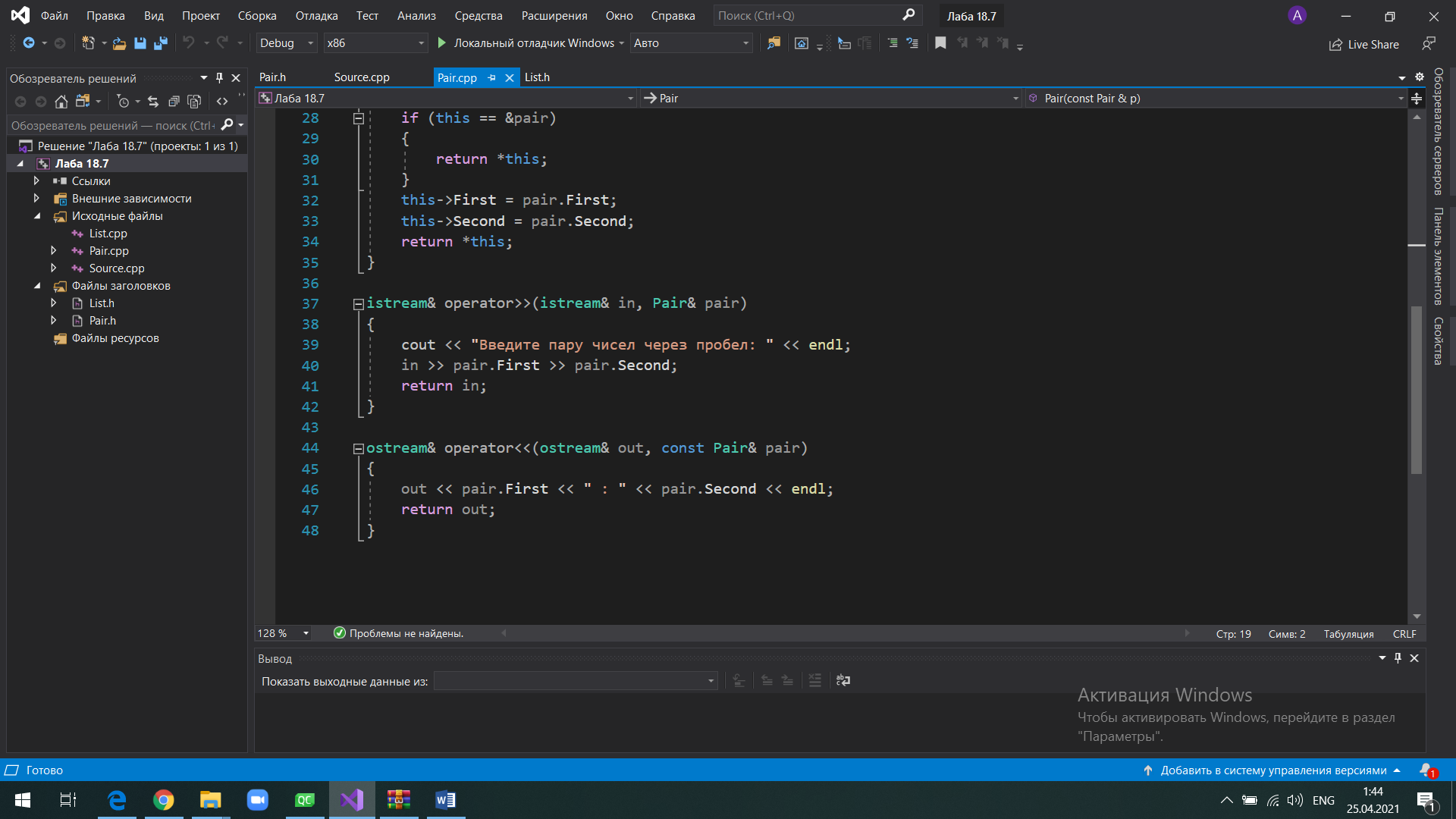
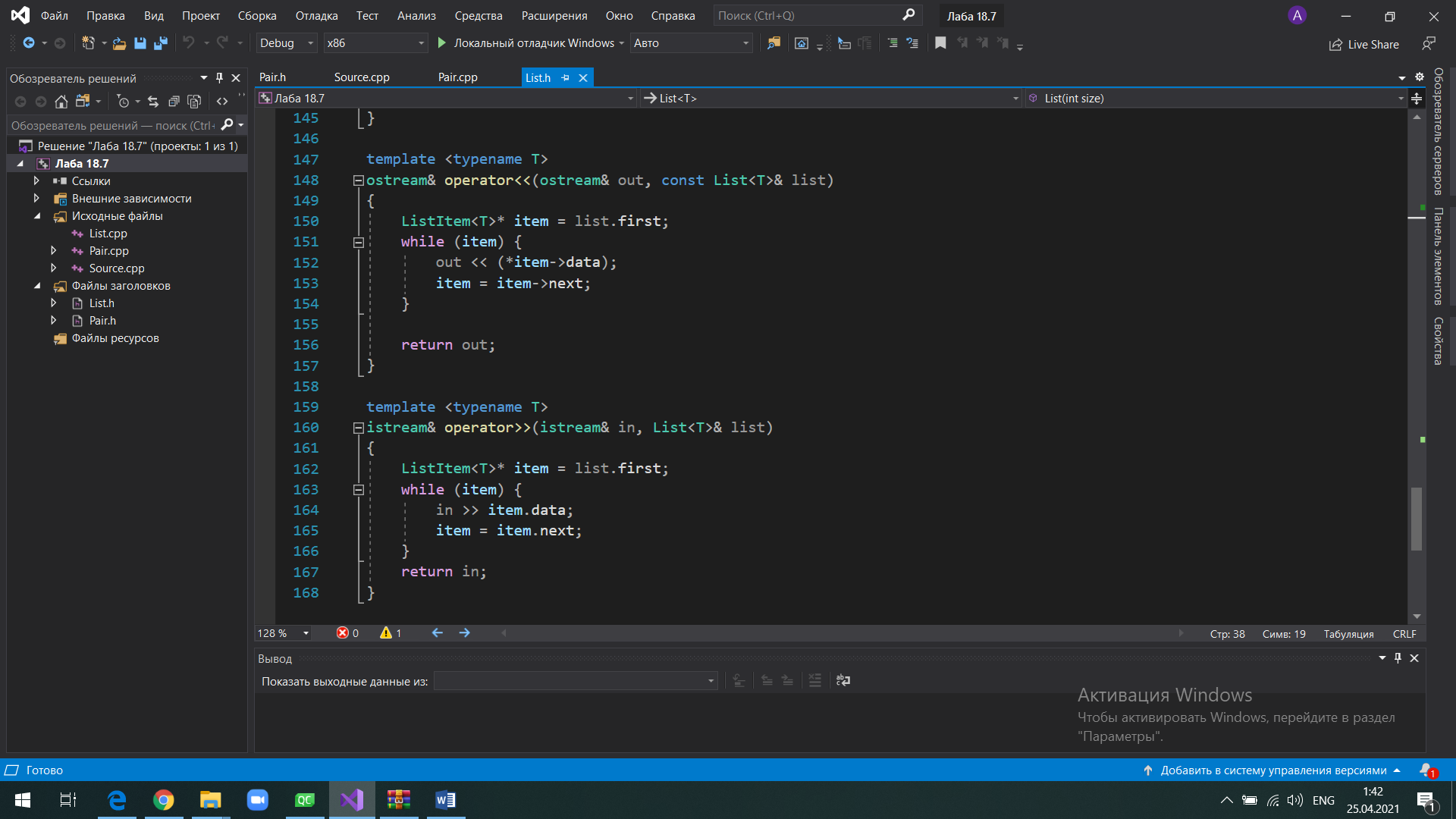
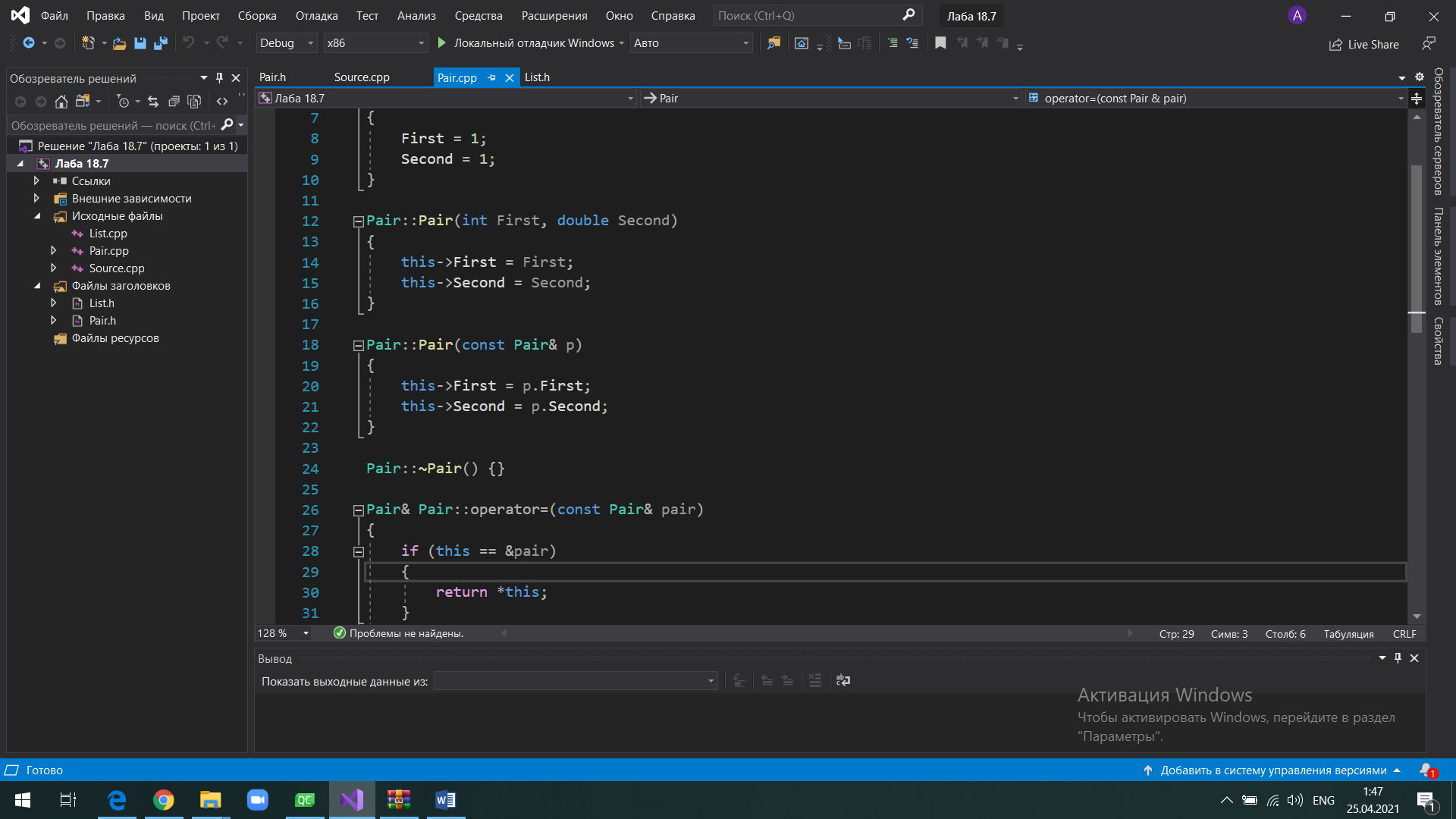
Доцент кафедры ИТАС

Полякова О.А.

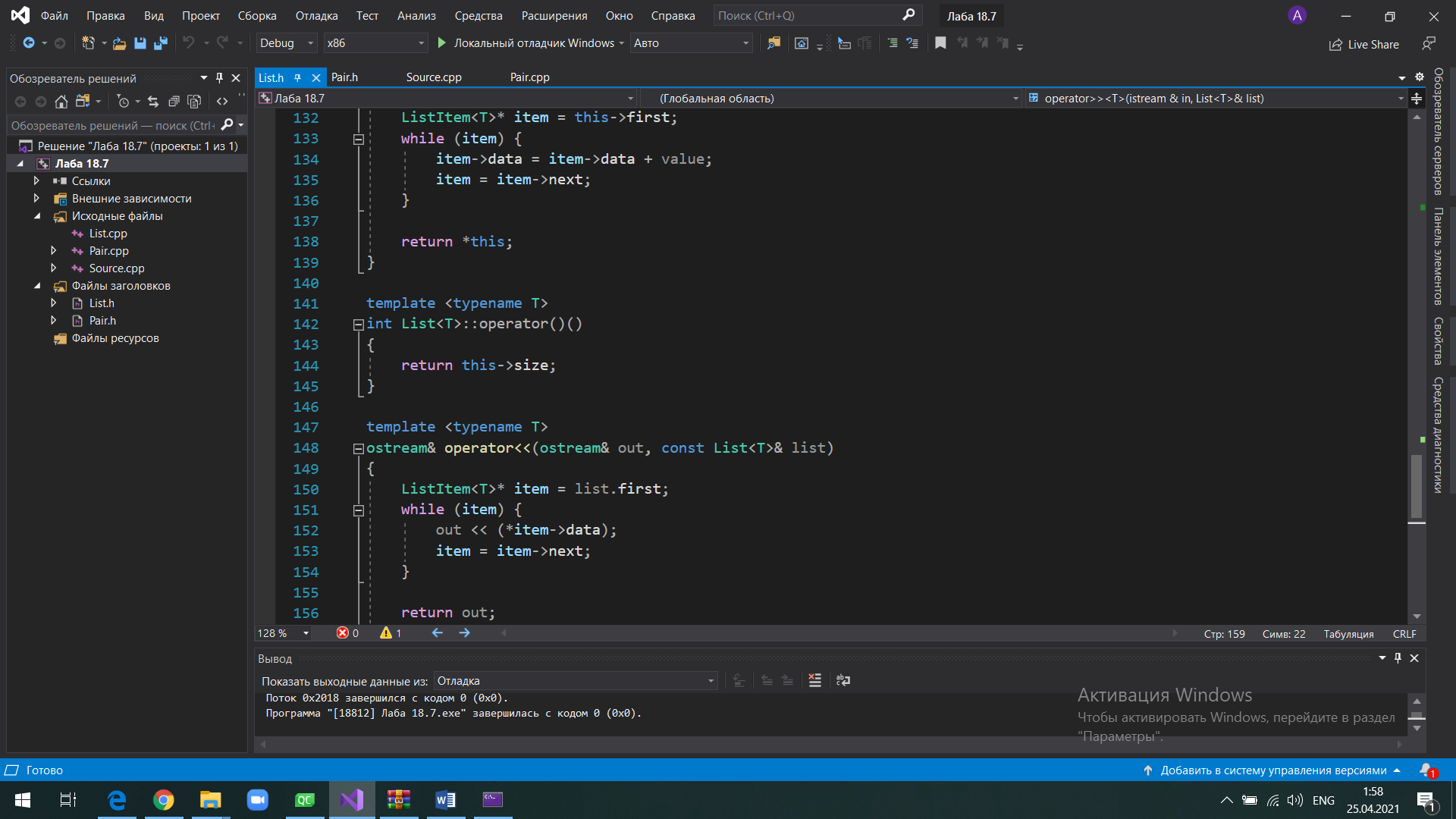
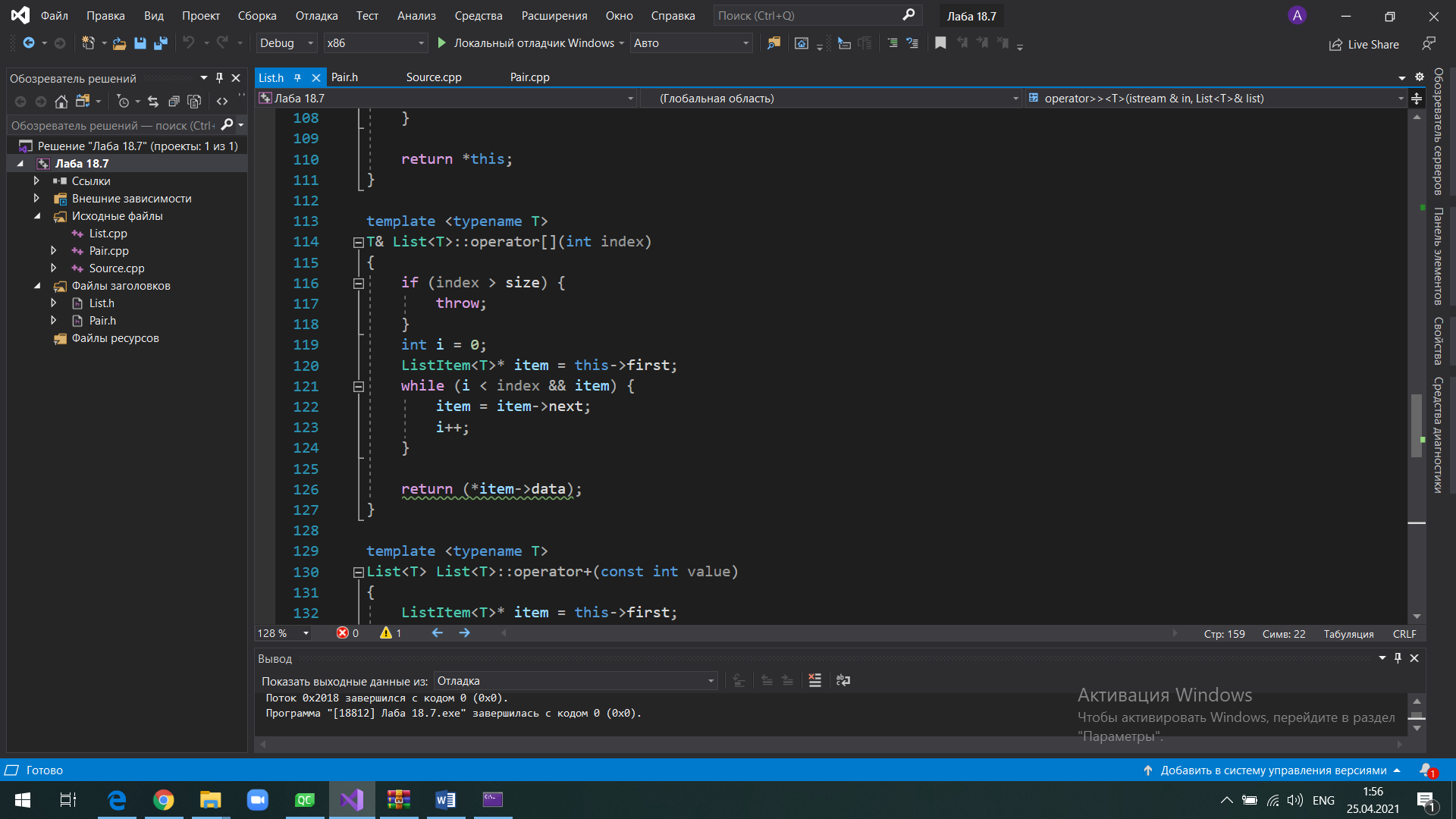
Пермь, 2021

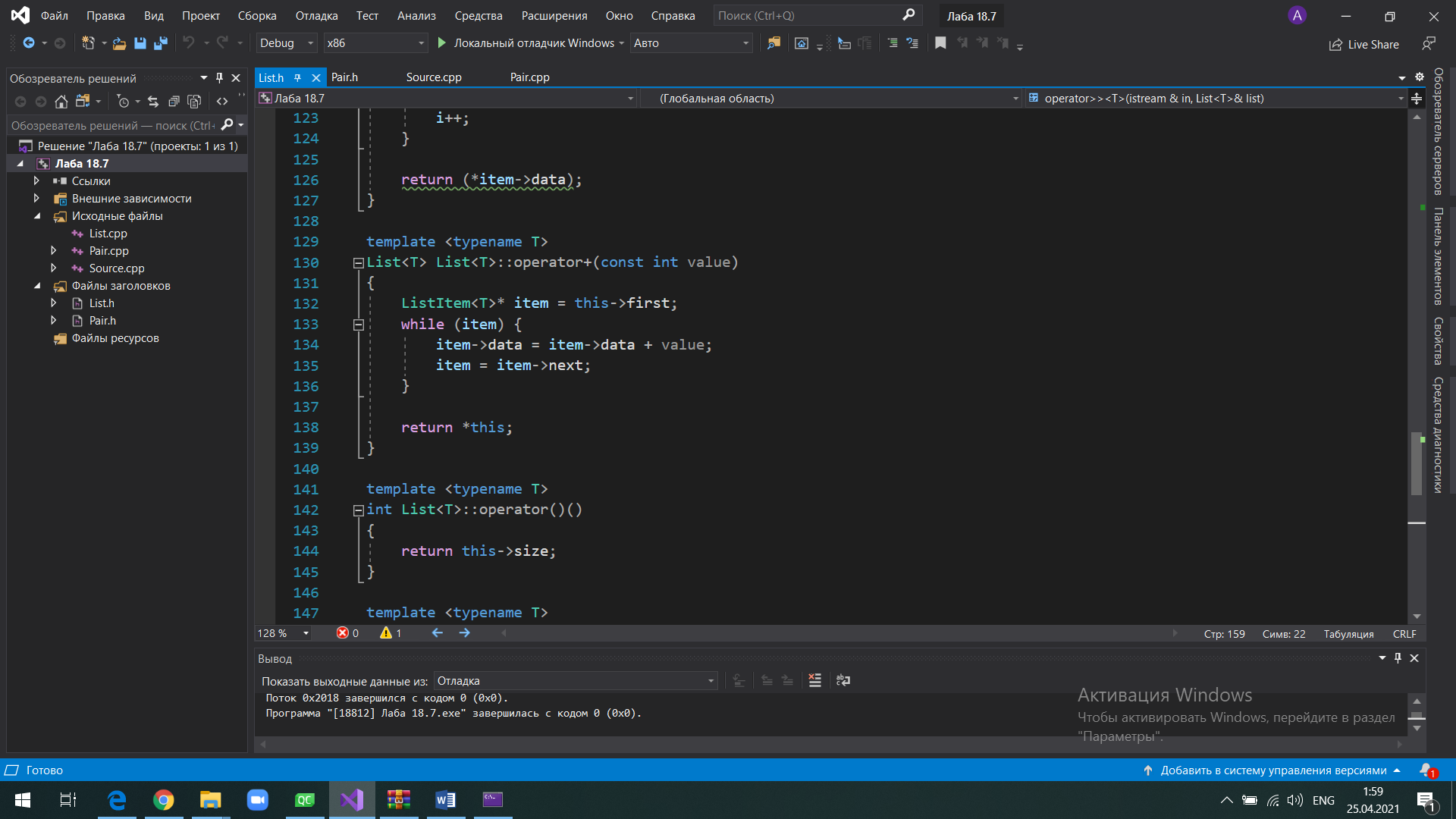
1. **Постановка задачи**



1. **Анализ задачи**
2. Для решения задачи необходимо:
   1. Реализовать класс шаблонный класс List.
   2. Реализовать перегрузки операторов из варианта
3. В ходе работы были использованы следующие типы данных:
   1. Класс Pair ЛР2.
   2. Класс List.  
        
      
4. Для решения задачи данные были представлены в следующем виде:
   1. Создается объект класса List типа Pair
5. Для решения задачи данные были представлены в следующем виде:
   1. Аналогично ЛР6, для класса List реализованы перегрузки операторов ввода и вывода:  
      
6. Поставленные задачи будут решены следующими действиями:
   1. Для классов Pair и List реализованы конструкторы и деструкторы.  
        
      

5.2 Аналогично ЛР6 реализованы требуемые перегрузки операторов для классов, используемых в решении.





1. **Код программы**

**List.h :**

#pragma once

#include <iostream>

using namespace std;

template <typename T>

struct ListItem

{

T\* data;

ListItem<T>\* next;

~ListItem() {

delete this->data;

}

};

template <class T>

class List

{

int size;

ListItem<T>\* first;

ListItem<T>\* last;

void addItem(T& value);

public:

List(int size);

List(List<T>& list);

~List();

List& operator=(const List<T>& list);

T& operator[](int index);

List operator+(const int value);

int operator()();

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

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

};

template <class T>

List<T>::List(int size)

{

this->last = 0;

this->first = 0;

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

T\* data = new T;

cin >> (\*data);

this->addItem(\*data);

}

}

template <typename T>

List<T>::List(List<T>& list)

{

\*this = list;

}

template <typename T>

List<T>::~List()

{

ListItem<T>\* current = this->first;

while (current) {

ListItem<T>\* rem = current;

current = current->next;

delete rem;

}

}

template <typename T>

void List<T>::addItem(T& value)

{

ListItem<T>\* item = new ListItem<T>;

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++;

}

template <typename T>

List<T>& List<T>::operator=(const List<T>& list)

{

if (this == &list)

{

return \*this;

}

// очистка

this->~List();

this->size = 0;

this->last = 0;

this->first = 0;

// наполнение

ListItem<T>\* item = list.first;

while (item) {

T\* data = new T(\*(item->data));

this->addItem(\*data);

item = item->next;

}

return \*this;

}

template <typename T>

T& List<T>::operator[](int index)

{

if (index > size) {

throw;

}

int i = 0;

ListItem<T>\* item = this->first;

while (i < index && item) {

item = item->next;

i++;

}

return (\*item->data);

}

template <typename T>

List<T> List<T>::operator+(const int value)

{

ListItem<T>\* item = this->first;

while (item) {

item->data = item->data + value;

item = item->next;

}

return \*this;

}

template <typename T>

int List<T>::operator()()

{

return this->size;

}

template <typename T>

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

{

ListItem<T>\* item = list.first;

while (item) {

out << (\*item->data);

item = item->next;

}

return out;

}

template <typename T>

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

{

ListItem<T>\* item = list.first;

while (item) {

in >> item.data;

item = item.next;

}

return in;

}

**Pair.h :**

#pragma once

#include <iostream>

using namespace std;

class Pair

{

int First;

double Second;

public:

Pair();

Pair(int First, double Second);

Pair(const Pair& p);

~Pair();

Pair& operator=(const Pair& pair);

friend istream& operator>>(istream& in, Pair& pair);

friend ostream& operator<<(ostream& out, const Pair& pair); };

**Pair.cpp :**

#include <iostream>

#include "Pair.h"

using namespace std;

Pair::Pair()

{

First = 1;

Second = 1;

}

Pair::Pair(int First, double Second)

{

this->First = First;

this->Second = Second;

}

Pair::Pair(const Pair& p)

{

this->First = p.First;

this->Second = p.Second;

}

Pair::~Pair() {}

Pair& Pair::operator=(const Pair& pair)

{

if (this == &pair)

{

return \*this;

}

this->First = pair.First;

this->Second = pair.Second;

return \*this;

}

istream& operator>>(istream& in, Pair& pair)

{

cout << "Введите пару чисел через пробел: " << endl;

in >> pair.First >> pair.Second;

return in;

}

ostream& operator<<(ostream& out, const Pair& pair)

{

out << pair.First << " : " << pair.Second << endl;

return out;

}

**Main.cpp :**

#include <iostream>

#include "List.h"

#include "Pair.h"

using namespace std;

int main()

{

system("color F0");

setlocale(LC\_ALL, "rus");

List<Pair> list\_a(5);

List<Pair> list\_b(list\_a);

List<Pair> list = list\_b;

cout << list << endl;

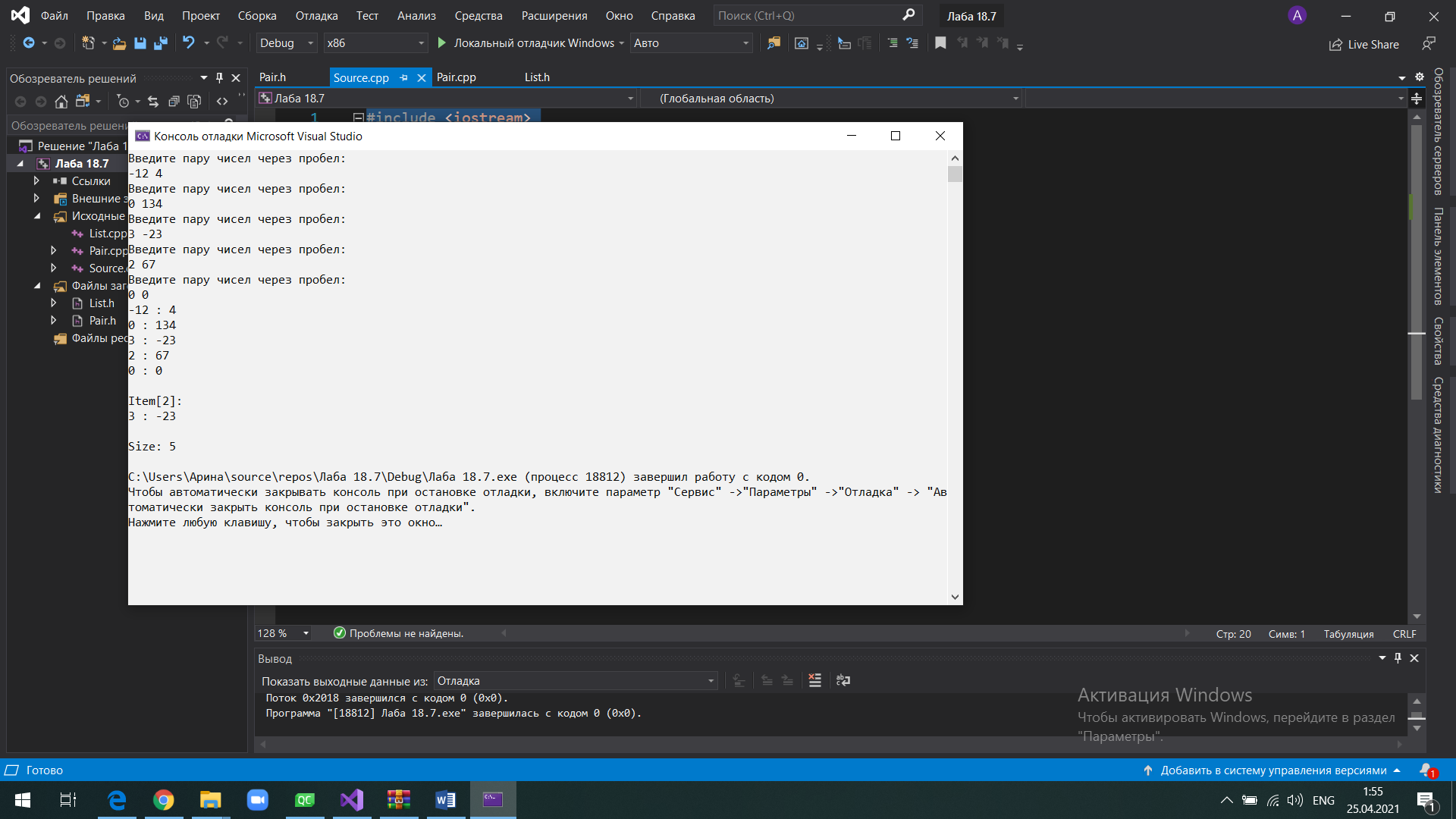
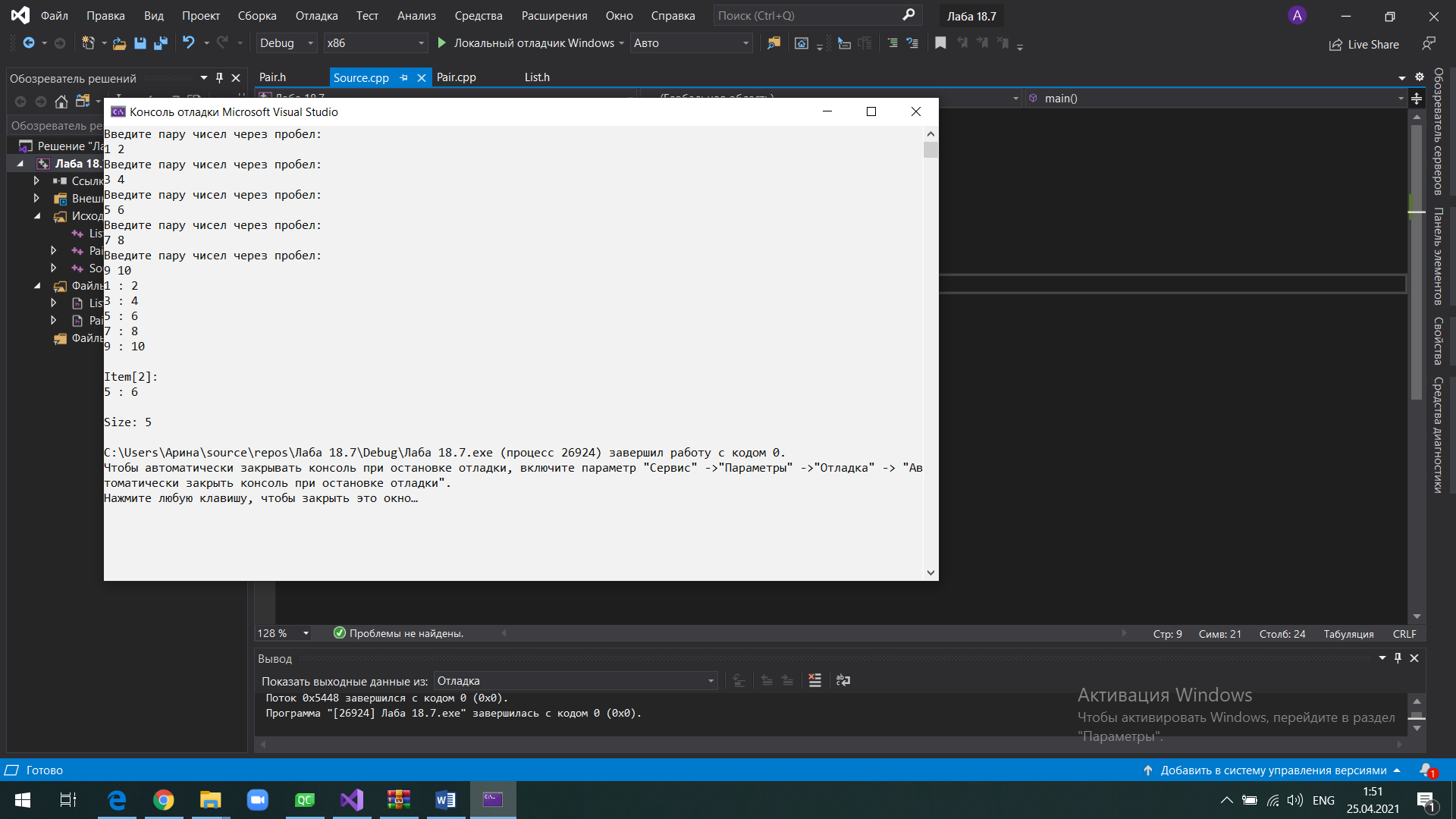
cout << "Item[2]: " << endl << list[2] << endl;

cout << "Size: " << list() << endl;

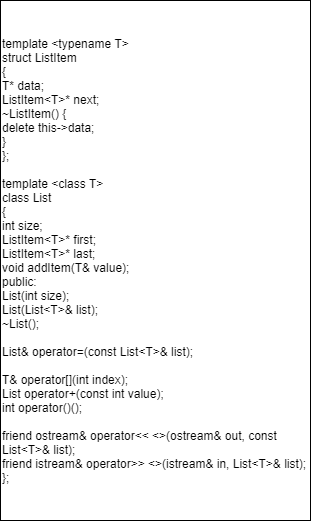
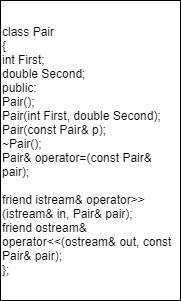
return 0;

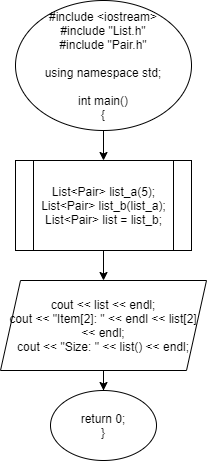
}

1. **Выполнение программы**



1. **Блок-схема**





1. **Ответы на вопросы**

