Дисциплина **Структуры и алгоритмы обработки данных**

Лабораторная работа №1. **Линейные списки**

Студент **Бекошин Никита Вячеславович**

Группа **1ПИб-02-3оп-23-1пг**

**Отчет**

Вариант задания

**Вариант 20**

1. Даны два стека целых чисел. Из элементов пеpвого стека меньших

сpеднего аpифметического элементов втоpого сфоpмиpовать очередь.

Выполнить задания, используя для пpедставления очеpедей и стеков:

а) массивы; б) динамические списки.

Требования к программам:

1. Количество элементов исходных линейных списков заранее не определено

и задается случайным образом. При дальнейшей обработке считается, что

количество элементов списка не известно, т.е. обработка производится, пока

не достигнут конец списка.

2. Программа должна сформировать исходные линейные списки, вывести их содержимое

на экран (при этом данные из списков не должны быть потеряны), произвести

обработку и вывести содержимое итогового списка на экран.

Листинг с подробными комментариями

Задача №1:

#include <iostream>

#include <time.h>

using namespace std;

struct Queue1 { //структура очереди. #1

int head, tail, size; //в циклической очереди head всегда указывает (не в том смысле, что является указателем через “\*”) на начало очереди, а tail указывает на следующий свободный слот.

int\* data; //указатель на динамический массив, указывает на первый элемент массива,

//к тому же, не может указывать на какие-либо другие адреса других элементов этого массива

// кроме как на адрес первого элемента

};

void nullQueue(Queue1& g) { //сбрасывает или устанавливает значения head и tail в начальное состояние

g.head = 0, //первый индекс массива

/\* g.tail = g.size - 1; создает ложное впечатление, что очередь переполнена, как будто нельзя добавить первый элемент в очередь, так как tail уже указывает на последний элемент очереди, но функции empty, add исправляют эту ситуацию \*/

g.tail = g.size - 1; //последний индекс массива (значение, которое может изменяться,

//в отличие от head, который всегда имеет индекс массива 0)

}

void initQueue(Queue1& g, int capacity) { //Инициализация очереди, в качестве параметров получает

//ссылку на исходную структуру и количество желаемых элементов в очереди

g.size = capacity + 1; //благодаря зацикливанию, tail никогда не достигнет capacity + 1, если при достижении элемента с индексом capacity-1 очередь заполнится полностью, то элемент с индексом capacity – это уже элемент с нулевым индексом

g.data = new int[g.size]; //инициализация динамического одномерного массива, который нужен для

//хранения элементов очереди. Коротко говоря, основа для исходной структуры

nullQueue(g); //Вызов функции для инициализации переменных head,tail

}

/\*благодаря зацикливанию, tail никогда не достигнет capacity + 1\*/

int next(Queue1& g, int n) {//next возвращает индекс следующего элемента, которого предстоит добавить

return (n + 1) % g.size; //если очередь не переполнится, выведется индекс элемента, который выходит за границу массива

//эта функция вернет значение нуля, то есть индекса 1 элемента, если произойдет ситуация переполнения очереди, все пойдет по кругу - старые элементы будут заменяться старыми

}

/\*empty не помогает нам однозначно определить, очередь пустая или переполненная, она просто выводит 0 или 1, 0, если очередь переполнена, или если очередь пуста, 1 – если очередь не переполнена, но и не пуста\*/

bool empty(Queue1& g) {//Изначально, когда в очередь не добавлено элементов, то индекс головы

//равняется индексу хвоста

return next(g, g.tail) == g.head; //Очередь считается пустой, когда за хвостом следует голова,

//НО если между хвостом и головой есть хотя бы один свободный элемент, то очередь не считается пустой;

}

void add(Queue1& g, int value) { //функция добавления элемента в очередь, каждый раз при добавлении

//элемента индекс хвоста увеличивается на единицу.

if (next(g, next(g, g.tail)) == g.head) //Перед тем как положить новый элемент в конец очереди,

//проводится проверка на переполнение очереди.

cout << "Queue overflow" << endl;

else

{

g.tail = next(g, g.tail); //тот самый момент, когда индекс хвоста при добавлении элемента

//увеличивается на единицу (циклично)

g.data[g.tail] = value; //value - добавляемый в очередь новый элемент.

}

}

int del(Queue1& g) { //функция удаления элемента из очереди. При удалении элемента индекс головы

//увеличивается на единицу, а когда "догоняет" индекс хвоста, оказывается, что были удалены все элементы

if (empty(g)) { //проверяет на пустоту

cout << "Queue is empty" << endl;

return 0;

}

else

{

int d = g.data[g.head];//если очередь не пустая, то переменная d инициализируется значением

//элемента, имеющего индекс головы (чтобы при удалении сохранить элемент)

g.head = next(g, g.head); //Индекс головы увеличится

return d; //возвращается d

}

}

struct Stack1 { //структура стека

int top; //вершина стека

int\* data; //указатель на адрес первого элемента массива

int size;

};

void nullStack(Stack1& st) { //функция присваивает начальное значение top

st.top = -1; //в стеке ничего нет; Можно подумать, что поле top может при инициализации быть любым

//отрицательным числом, но суть в том, что когда добавляется элемент в массив, то он будет иметь индекс 0,

//и top должен увеличиться на единицу, чтобы занять место индекса первого элемента стека\* /

}

void InitStack(Stack1& st, int capacity) {

st.data = new int[capacity]; //обращение к оператору new, чтобы выделить память в динамической /new int[capacity], значит от элементы будут иметь индекс от 0 до capacity-1

//области размером в 'capacity'

st.size = capacity; //размер стека

nullStack(st);

}

void push(Stack1& st, int value) {

if (st.size - 1 > st.top) // st.size-1, представим массив из одного элемента: изначально top= -1;

//установленный размер массива size=1; когда добавляется элемент, top теперь 0, добавленный элемент

//имеет индекс 0, и очевидно, что стек переполнен, но чтобы учесть факт этого, нужно проверять:

// // st.size-1>st.top (1-1)>0 => 0>0 - условие нарушено - стек переполнен

/\*if (sizeof(st.data) / sizeof(int) - 1>st.top) /\* ненужный код\*/

st.data[++st.top] = value; //положили новый элемент на "верхушку" стека; Префиксная форма увеличения

//означает означает, что перед тем как закинуть значение в элемент, индекс увеличивается на единицу

else //элемент value записывается в элемент массива с индексом новой вершины стека

cout << "Stack overflow" << endl; //сообщение - стек переполнен

}

int pop(Stack1& st) { //функция удаления верхнего элемента из стека

return st.data[st.top--]; //Сначала возвращает элемент стека st.data[st.top], а потом уменьшает st.top

//на единицу, что соответствует логике работы стека (LIFO - последним вошел - первым и вышел)

}

bool empty(Stack1& st) { //проверка на пустоту стека; вызыывается в main для проверки - "если НЕ empty,

//то вызвать функцию pop - вытащить верхний элемент стека"

return st.top == -1; //возвращает ноль или единицу. Ноль - когда не пуст, единица - когда пуст

}

void main() {

setlocale(LC\_ALL, "rus");//русификация

srand(time(0));//создание гнёздышка для случайного числа

Stack1 st2; //определение стека 2

Stack1 st1; //определение стека 1

Queue1 qu3; //определение очереди

cout << "Введите количество элементов в стеках и будущей очереди" << endl;

int rdm; //количество элементов

cin >> rdm;

cout << "Количество элементов:" << endl;

cout << rdm << endl;

InitStack(st2, rdm);//инициализация стека 2 размером rdm

InitStack(st1, rdm);//инициализация стека 1 размером rdm

initQueue(qu3, rdm);//инициализация очереди размером rdm

int sr\_ar = 0; //Инициализация среднего арифметического

int sum = 0;//инициализация суммы для расчета среднего арифметического

for (int i = 1, random\_number = rand() % 10; i <= rdm; i++)//заполнение второго стека

{

push(st2, random\_number); //вызов функции добавлении элемента в стек

sum += random\_number;

random\_number = rand() % 10;

}

sr\_ar = (double)sum / rdm;//расчет среднего арифметического

//sr\_ar = 5;

for (int i = 1, random\_number = rand() % 10; i <= rdm; i++) //заполнение первого стека

{

push(st1, random\_number);//положить элемент в стек

random\_number = rand() % 10;//будущий элемент стека - случайное число от 0 до 9

}

int a;

cout

<< "Первый стек массивом" << endl;

while (!empty(st1)) { //цикл пока с проверкой на пустоту стека 1

a = pop(st1);//вытащить верхний элемент стека 1

cout << a << " "; //вывести верхний элемент стека 1

if (a < sr\_ar) //выбрать те, что меньше среднего арифметического значений элементов 2 стека

add(qu3, a); //добавить такие элементы в очередь 3

}

cout << endl;

cout << "Второй стек массивом" << endl;

while (!empty(st2)) { //цикл пока с проверкой на пустоту стека 2

a = pop(st2);//вытащить верхний элемент стека 2

cout << a << " "; //вывести верхний элемент стека 2

}

cout << endl;

cout << "Среднее арифметическое второго стека массивом:" << endl;

cout << sr\_ar << endl; //вывод ср.арифм.значения элементов 2 стека

if (empty(qu3)) //проверка на пустоту, чтобы понять, заполнилась ли очередь или нет, так как такое

//вполне возможно, учитывая случайность заполнения стеков

cout << "очередь не пополнилась, так как нет ни одного элемента, которые меньше ср.арифм.значения" << endl;

cout << "Элементы очереди (<" << sr\_ar << ")" << endl;

while (!empty(qu3)) { //цикл пока с проверкой на пустоту очереди 3

a = del(qu3);//вытащить элемент головы из очереди

cout << a << " "; //вывод вытащенного элемента

}

}

Задача №2:

#include <iostream>

#include <time.h>

class Queue {

private:

struct Node { //Определяет структуру узла связного списка

int data; //хранит данные, которые хранятся в очереди

Node\* next; //Поле в структуре узла связного списка,

//которое хранит указатель на следующий узел

};

Node\* head, \* tail; //head - указывает на первый элемент списка:

//head является точкой входа в список, голова использует

//'проводника' - указателя next, который помогает двигаться от

//элемента к элементу в очереди.

// head позволяет добавлять и удалять элементы в начале списка

public:

Queue() { //конструктор по умолчанию

head = NULL; //нулевые значения головы

tail = NULL; //и хвоста

}

/\*Методы класса - компонентные функции\*/

bool empty() { //проверка на то, указывает ли голова на что-то

return head == NULL; //возвращение 0 или 1 в зависимости от истинности равенства; если оно

//справедливо, то вернет 1

}

void add(int value) {

if (empty()) {//проверка на пустоту, если пуст, то:

head = new Node; //Выделяем память для Node в 'куче' - создается новый объект Node, так как head - указатель

//он будет хранить адрес нового узла типа Node и указывать на него. Вкратце, создается копия структуры Node

// в 'куче' со всеми её полями data, \*next

//head присваивается адрес нового узла типа Node, эту функцию присваивания выполняет оператор new

head->data = value; //head обращается к полю data нового узла

//значение value присваивается полю data нового узла

head->next = NULL; //устанавливает указатель next первого

//узла в NULL, чтобы пометить узел как последний, это

// изменение в NULL не уничтожает узел и не освобождает

// память.

tail = head; //голова и хвост указывают на первый элемент очереди

}

else {//если очередь не пуста, то:

tail->next = new Node; //просим у оператора new выделить

//память в динамической области для следующего узла, чтобы

//tail указывал на этот следующий узел

tail = tail->next; //после выделенной памяти, хвост теперь

//будет указывать на следующий узел

tail->data = value;// Полю data присваивается

// значение добавляемого элемента

tail->next = NULL; //устанавливает указатель next последнего

//узла в NULL, чтобы пометить узел как последний, то есть

//как конец списка

}

}

int del() {

if (empty()) { //проверка на переполненную очередь

std::cout << "Queue is empty" << std::endl;

return 0;

}

else {

int d = head->data; //head указывает на поле data,

//чтобы вытащить первый элемент из очереди, потом второй...третий

Node\* tmp = head; //дубликат указателя головы

head = head->next;//указатель head теперь указывает на

//следующий узел,

delete(tmp); //освобождение динамической памяти

return d; //возвращает значение, достанное из списка

}

}

void nullQueue() { //очищает стек, удаляя все его элементы (освобождает занятую им память)

Node\* tmp; //указатель на тип узла

while (!empty()) { //пока очередь не является пустой, то:

tmp = head;//указателю tmp присваивается значение указателя head

head = head->next; //перемещение головы на следующий узел списка

delete(tmp); //освобождение памяти

}

}

};

//узел - базовый элемент, хранит информацию и "сцепку" с другими узлами, образуя структуру

//Связанный список: узел содержит данные и укзаатель на следующий узел в списке

//Узел содержит: данные - хранимая информация в узле (целое число, строка, структура);

//Узел содержит: указатель или несколько указателей (в зависимости от типа структуры данных),

//указывают на другие узлы в структуре. Эти указатели позволяют "связывать" узлы друг с другом

struct Node { //Node - это узел, образующий структуру

//(мое предположение) это структура, являющаяся узлом

int data; //поле для хранения данных, которые можно хранить в узле

Node\* next; //поле next (это не узел) - указатель на следующий элемент списка (узла)

};

/\*Указатели позволяют передавать структуры в функции по ссылке\*/

void InitStack(Node\*& top) {//Задача переменной top указывать на адрес другой переменной и брать её

//адрес в качестве своего значения.

//InitStack инициализирует стек пустым и устанавливает указатель на верх стека (top) в NULL.

//Стек начинается пустым, и top будет указывать на первый элемент, когда будет добавлено

//что-то в стек.

top = NULL; //top - вершина стека (top управляет доступом к верхнему узлу стека, позволяя

//добавлять и удалять элементы

}

void push(Node\*& top, int value) { //2-й параметр - значение будущего элемента стека (сверху)

//push добавляет новый узел в стек с заданным значением value

Node\* tmp = new Node; //(Мое предположение) tmp содержит адрес этого узла; New Node - динамическая память

//Создается новый узел tmp и выделяется для него память с помощью new

// (new выделяет память для новых узлов)

tmp->next = top; //поле next теперь указывается на ноль (мое предположение, не обязательно правильное)

//(&top - возвращает адрес указателя top, а не адрес, на который он указывает)

// top - устанавливает указатель next нового узла на тот же адрес, на который указывает

// top, а не на адрес самого указателя top.

//Указатель next нового узла устанавливается в текущее значение top (то есть он указывает

//на предыдущий верх стека)

top = tmp;//(Предположение КМ) top будет указывать адрес узла, в котором содержится tmp

//(телепорт на 16 строку - зацикливание)

//top перемещается на новый узел

top->data = value; //В поле data нового узла записывается значение value

}

int pop(Node\*& top) { //извлекает верхний элемент из стека и возвращает его значение (First Out)

Node\* tmp = top; /\* Новый узел динамической памяти создается. В tmp записываем переменную и

//запоминаем адрес. Иными словами, tmp содержит адрес этого узла\*/

//создается временный указатель tmp, который указывает на верхний узел

int d = top->data; //извлекается значение из верхнего узла

top = top->next; //top перемещается на следующий узел

delete(tmp);//удалятся старый верхний узел из памяти с помощью delete

//(delete освобождает память, когда узел больше не нужен)

return d; //возвращается извлеченное значение

}

bool empty(Node\*& top) { //проверяет, пустой ли стек

return top == NULL;

//возвращает true, если стек пустой (top==NULL), иначе false

}

void nullStack(Node\*& top) {//очищает стек, удаляя все его элементы (освобождает занятую им память)

Node\* tmp; //определение указателя на тип узла

while (!empty(top)) { //пока вершина стека не пустая (ложь), то цикл проходит по всем узлам,

//удаляя каждый из них с помощью delete

tmp = top; //(Мое предположение) Она будет хранить адрес этого узла, хранящийся в динамической памяти

top = top->next; //'верхушка' стека теперь указывает на следующий узел в стеке

//чтобы удалить этот узел через delete(tmp)

delete(tmp); //освобождение динамической памяти, выделенной для tmp

}

}

void main() {

srand(time(0)); //создание гнёздышка для случайного числа

setlocale(LC\_ALL, "Russian"); //русификация

Node\* st2, \* st1;//создается указатель st2,st1 на тип Node\*

Queue qu3;

InitStack(st2); //Инициализируется стек с помощью InitStack(st2)

//Добавляются элементы в стек с помощью push

int d = 20; //диапазон генерации случайного числа

int rdm;

std::cout << "Введите количество элементов в стеках и будущей очереди" << std::endl;

std::cin >> rdm;

std::cout << "Количество элементов:" << std::endl;

std::cout << rdm << std::endl; //вывод количества элементов

//количество элементов в стеке определяется случайным числом от 0 до d-1

/\*if (rdm == 0) //проверка на нулевое количество

while (rdm == 0) //цикл - успешно сгенерировать число, отличное от нуля

rdm = rand() % d;

cout << "Количество элементов:" << endl;

cout << rdm << endl; //вывод количества элементов\*/

int sr\_ar = 0; //инициализация под средне-арифметическое

int sum = 0;//инициализация под вспомогательную сумму

for (int i = 1, random\_number = rand() % 10; i <= rdm; i++)//заполнение 2 стека

{

push(st2, random\_number); //Положить элемент в стек

sum += random\_number; //считает сумму всех значений элементов

random\_number = rand() % 10; //После добавления элемента в стек происходит

//новая генерация случайного числа

}

sr\_ar = (double)sum / rdm;//Ср.арифм.значение - сумма значений элементов

//делится на количество элементов

InitStack(st1);//Инициализируется стек с помощью InitStack(st1)

for (int i = 1, random\_number = rand() % 10; i <= rdm; i++) //заполнение 1 стека

{

push(st1, random\_number);//происходит добавление этого элемента в стек

random\_number = rand() % 10; //генерация числа для следующего элемента стека

/\*if (random\_number < sr\_ar)//если сгенерированное значение меньше ср.арифм., то:

{

push(st1, random\_number);//происходит добавление этого элемента в стек

random\_number = rand() % 10; //генерация числа для следующего элемента стека

}

else

{

//while(random\_number < sr\_ar)

random\_number = rand() % 10; //Повторная генерация числа, т.к. прошлое оказалось

//больше ср.арифм.значения

i--; //чтобы не произошло выхода за границы массива и не прозвучало "Stack overflow"

//оно уменьшается на единицу, чтобы компенсировать увеличение на единицу

}\*/

}

int a;

std::cout << "Первый стек списком" << std::endl;

while (!empty(st1)) { //Если стек не является пустым, то:

a = pop(st1);//Присваивает извлеченное (сверху) возвращаемое значение

//верхнего элемента из стека (верхний элемент - элемент, который добавлен последним)

std::cout << a << " "; //вывод такого элемента

if (a < sr\_ar)

qu3.add(a);

}

std::cout << std::endl;

std::cout << "Второй стек списком" << std::endl;

while (!empty(st2)) { //в цикле извлекаются элементы из стека с помощью pop

a = pop(st2); //вызов функции вытащить первый элемент из очереди

std::cout << a << " "; //выводится значение извлеченного элемента

}

std::cout << std::endl;

std::cout << "Среднее арифметическое второго списка:" << std::endl;

std::cout << sr\_ar << std::endl;//вывод ср.арифм.значения

if (qu3.empty())//если очередь пуста, то:

std::cout << "очередь не пополнилась, так как нет ни одного элемента, которые меньше ср.арифм.значения" << std::endl;

std::cout << "Элементы очереди (<" << sr\_ar << ")" << std::endl;

while (!qu3.empty()) { //пока очередь не пуста, то:

a = qu3.del();//По одному удаляются элементы из этой очереди,

//начиная с первого элемента, доступ к которому имеет голова

std::cout << a << " ";

}

}

Скриншоты кода

Задача №1:

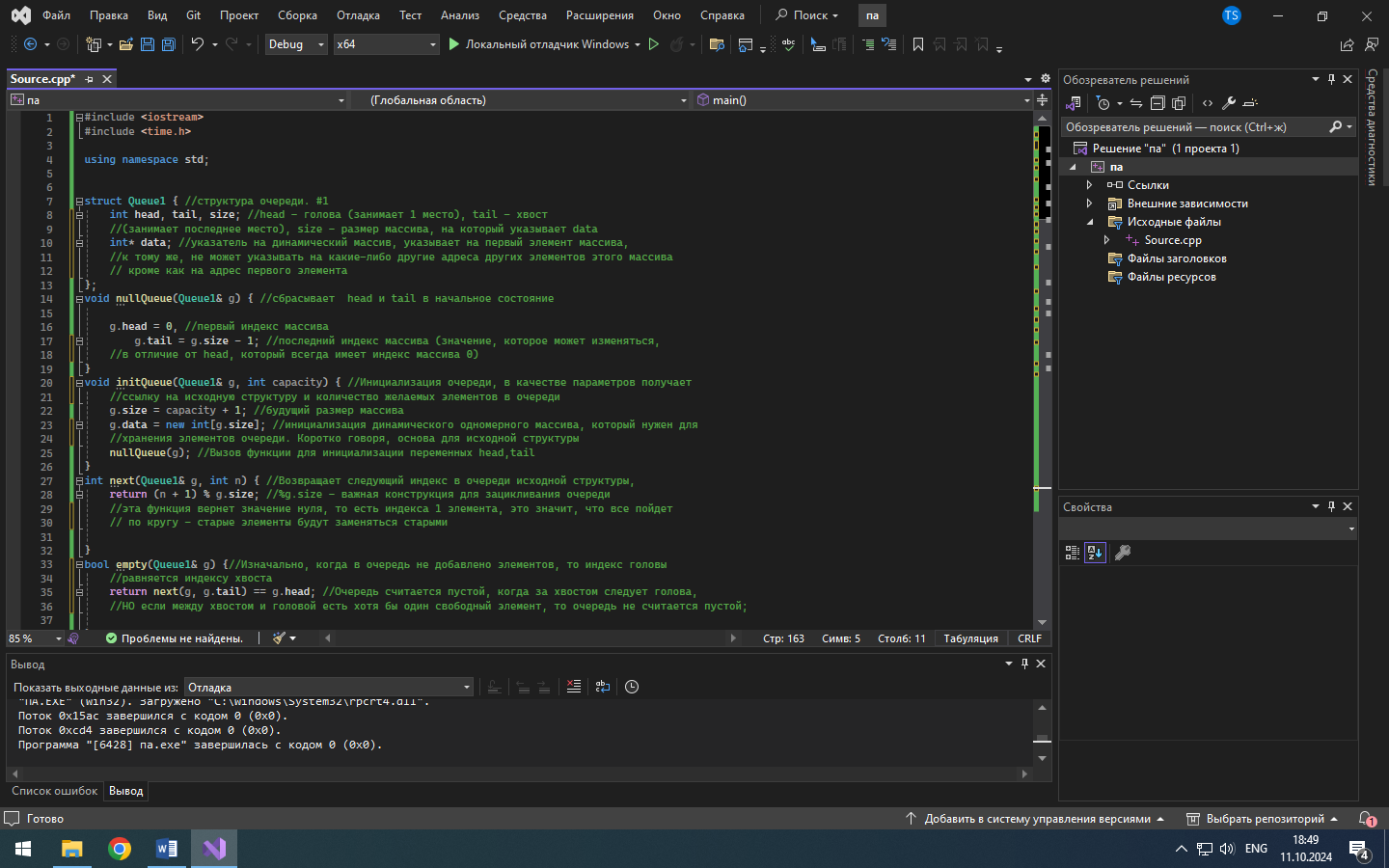


Рис. 1



Рис. 2

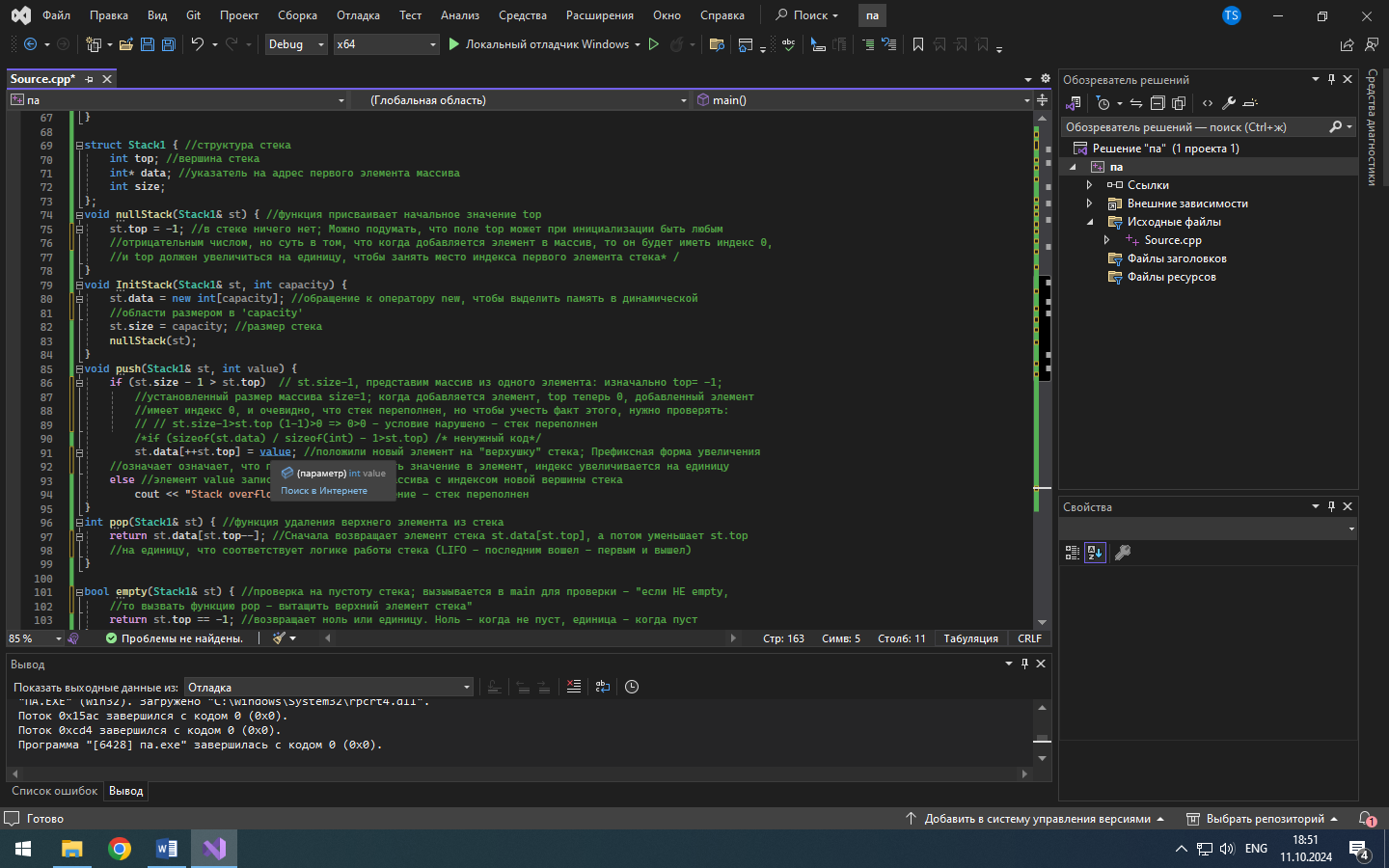


Рис. 3

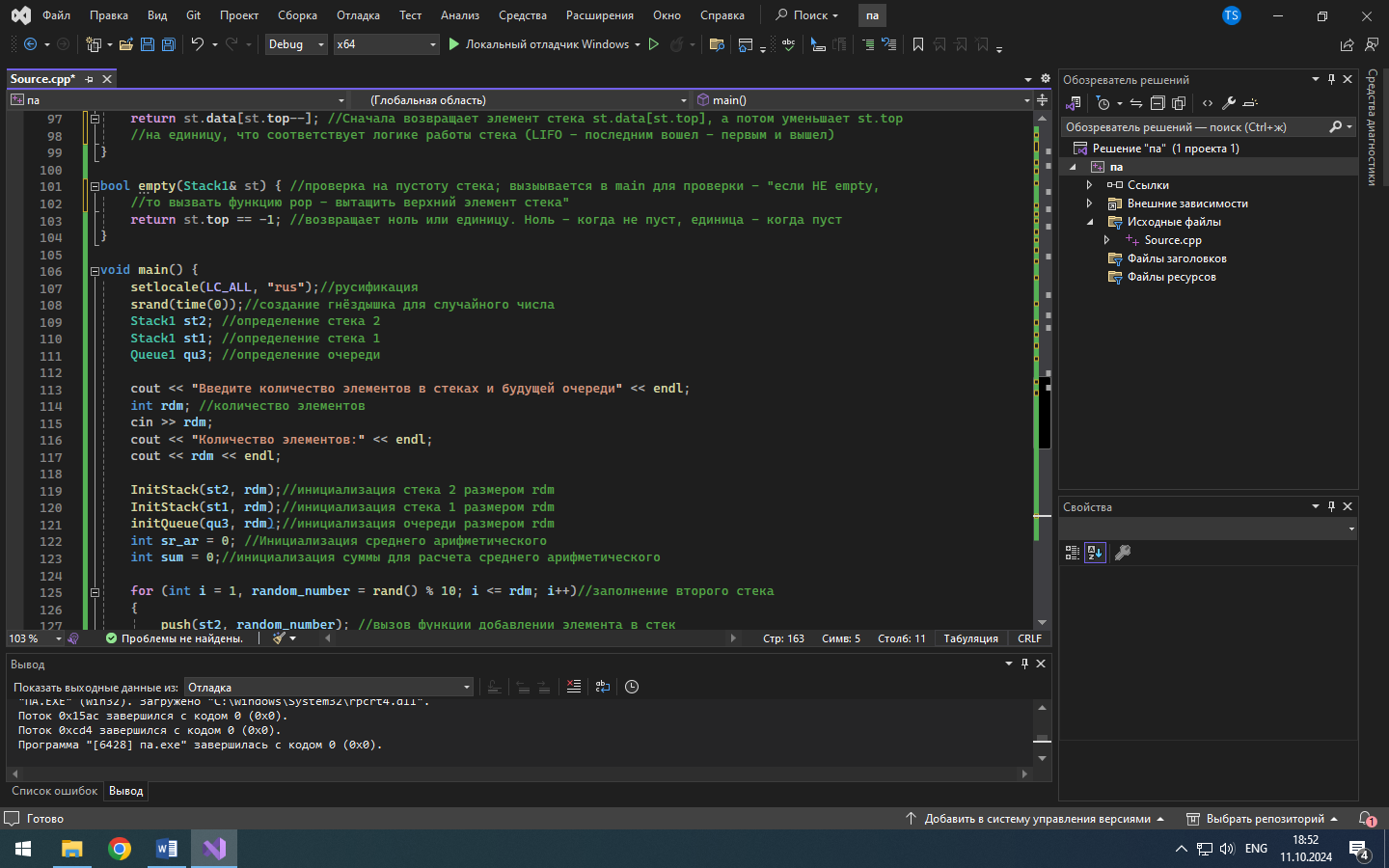


Рис. 5

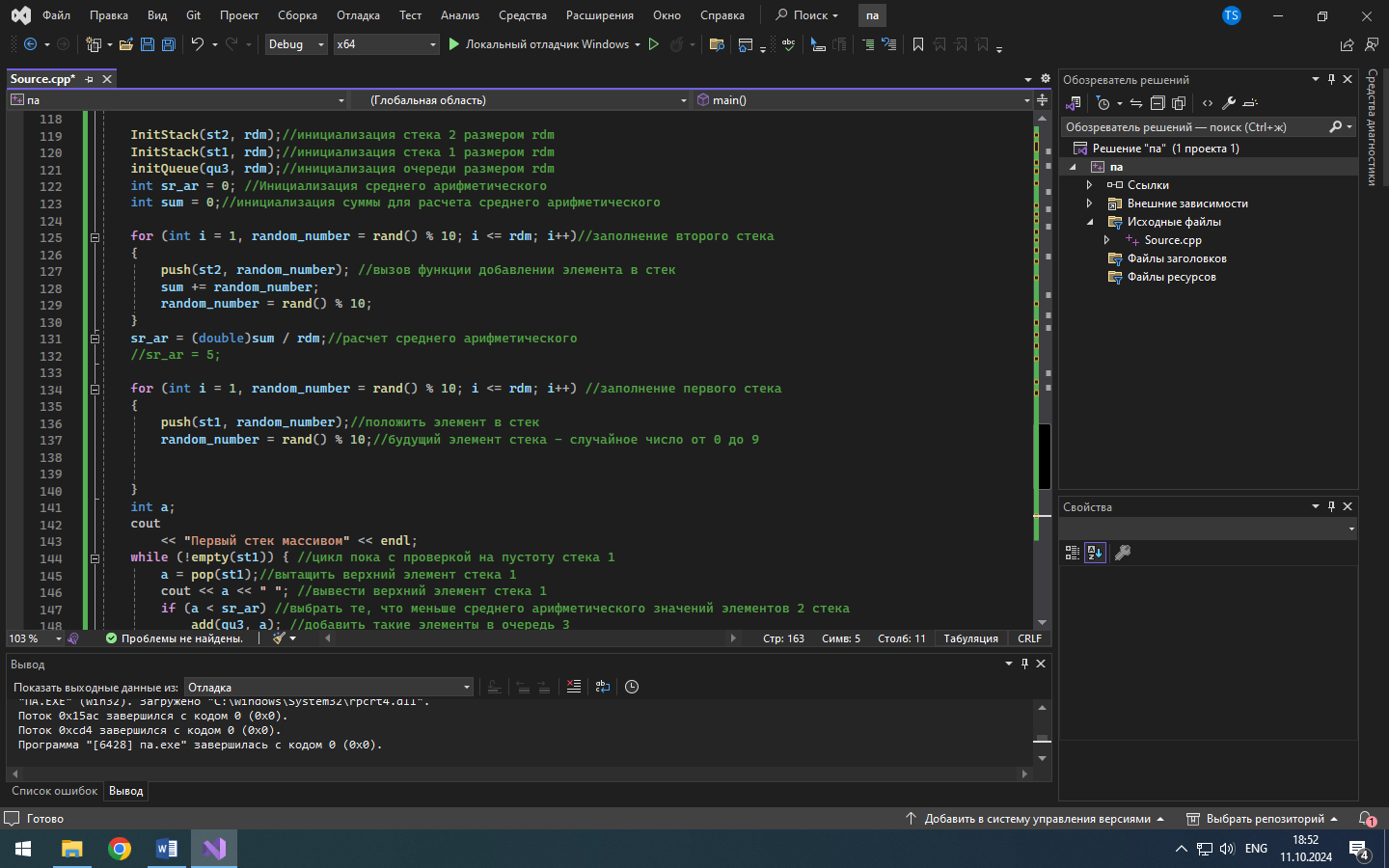


Рис. 6

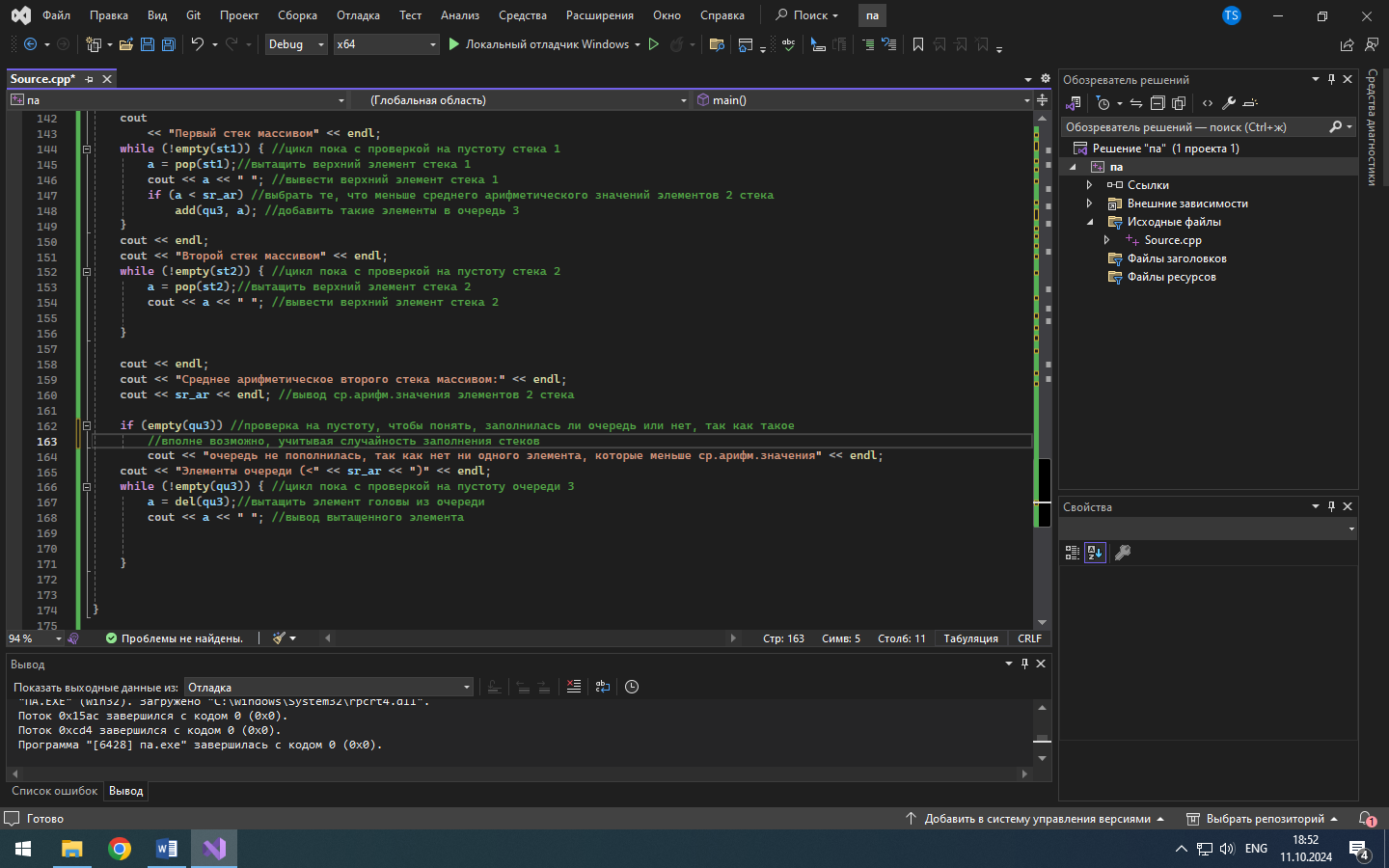


Рис. 7

Задача №2 (2 скриншота):

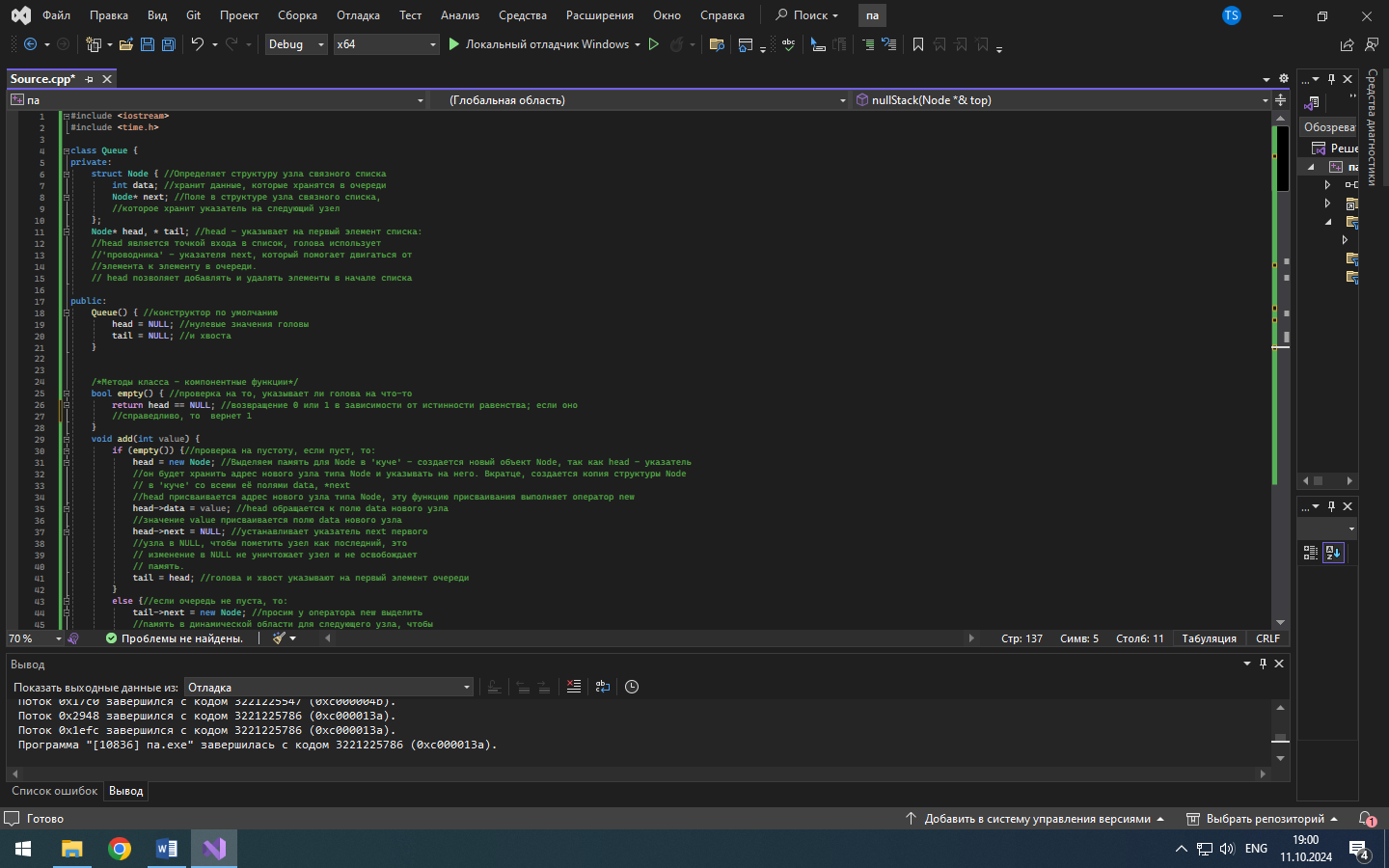


Рис. 8

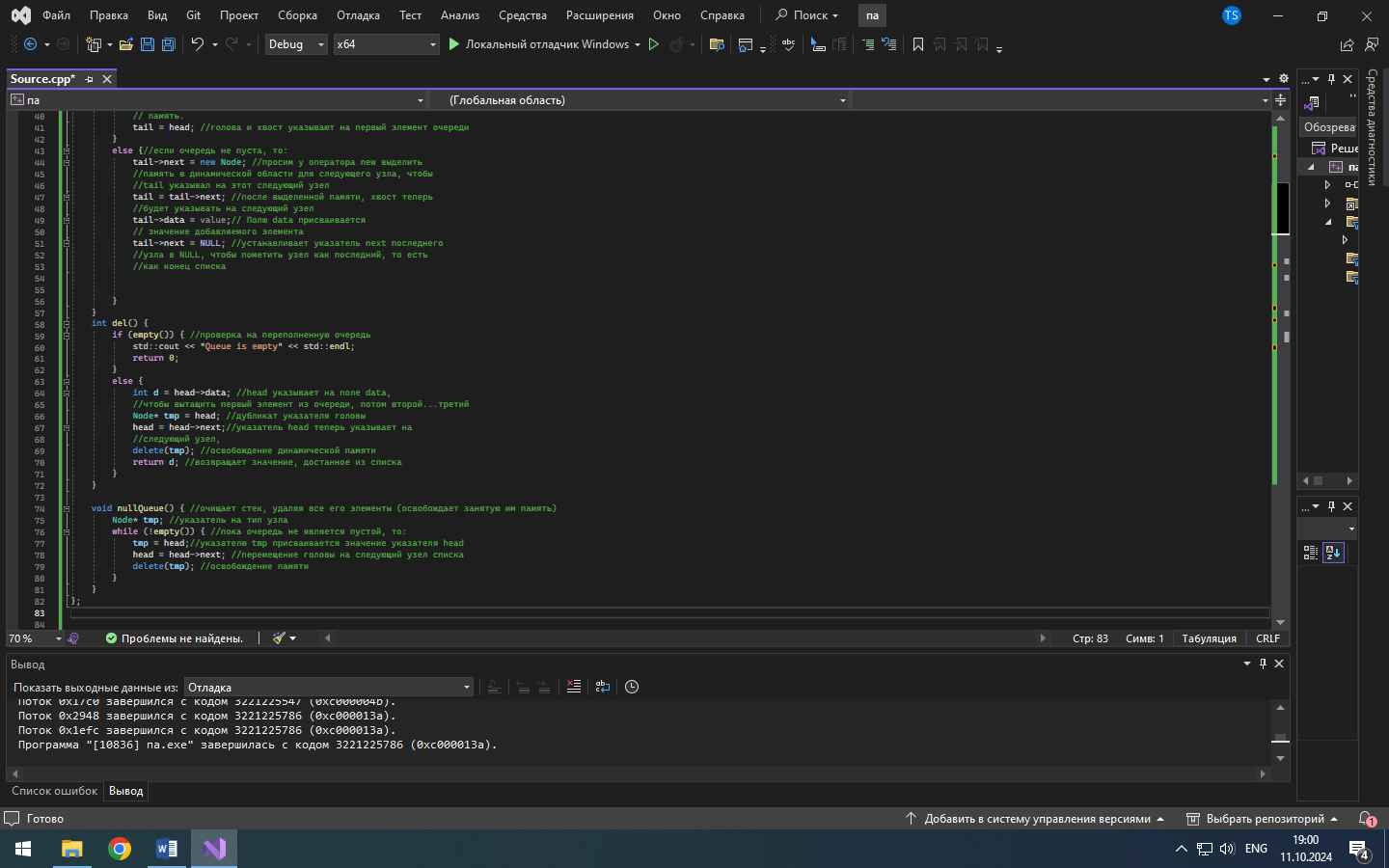


Рис. 9

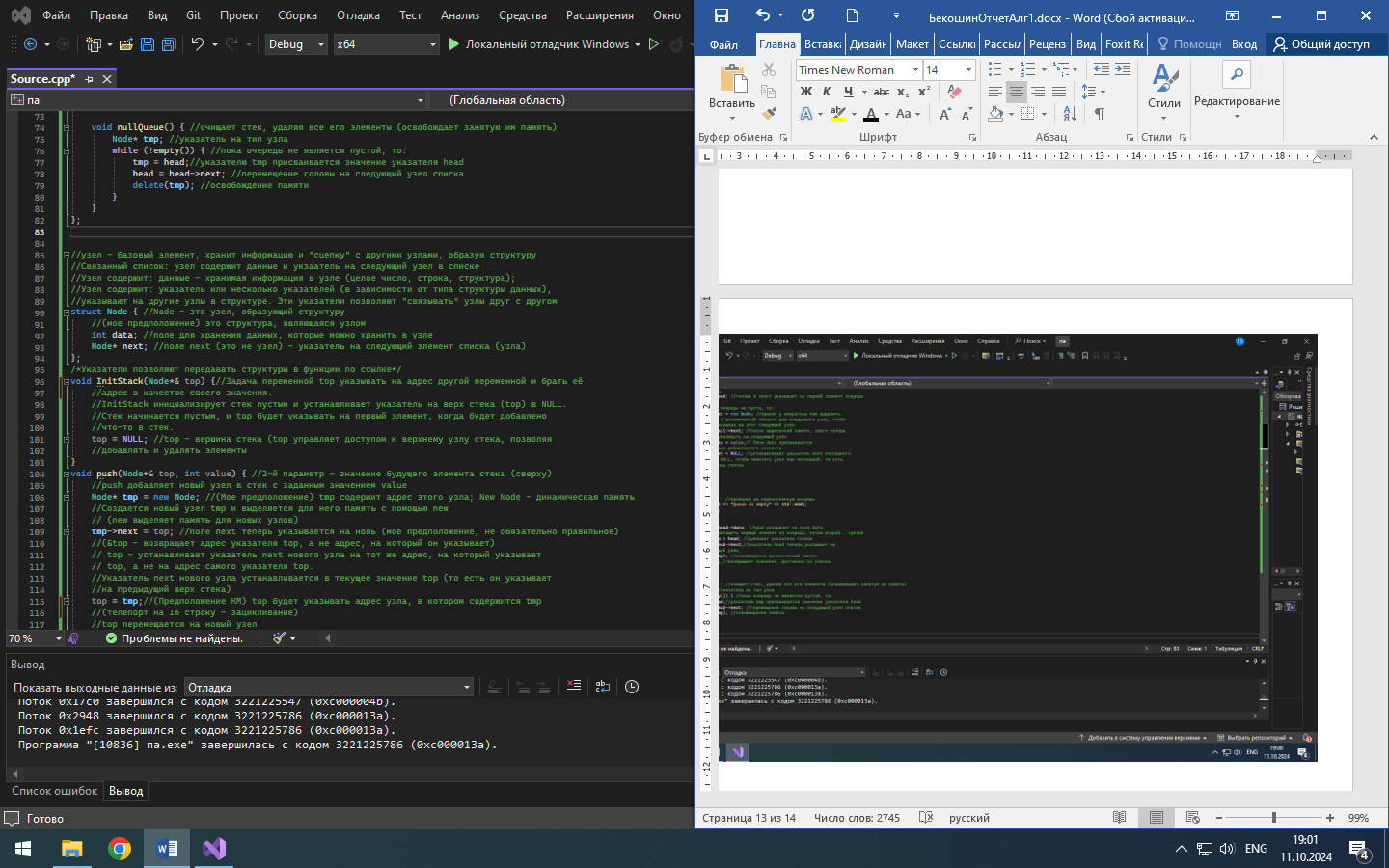


Рис. 10



Рис. 11

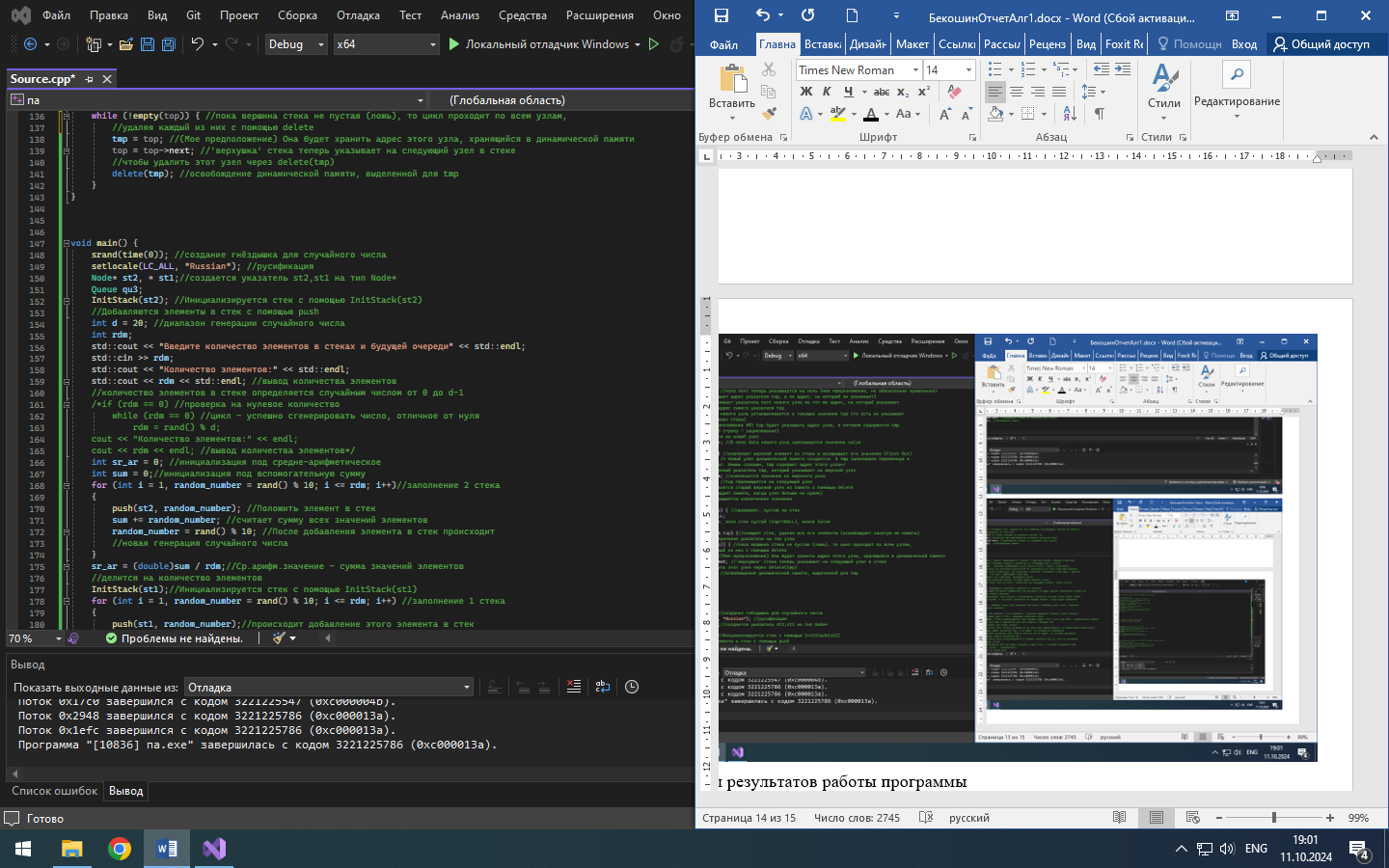


Рис. 12

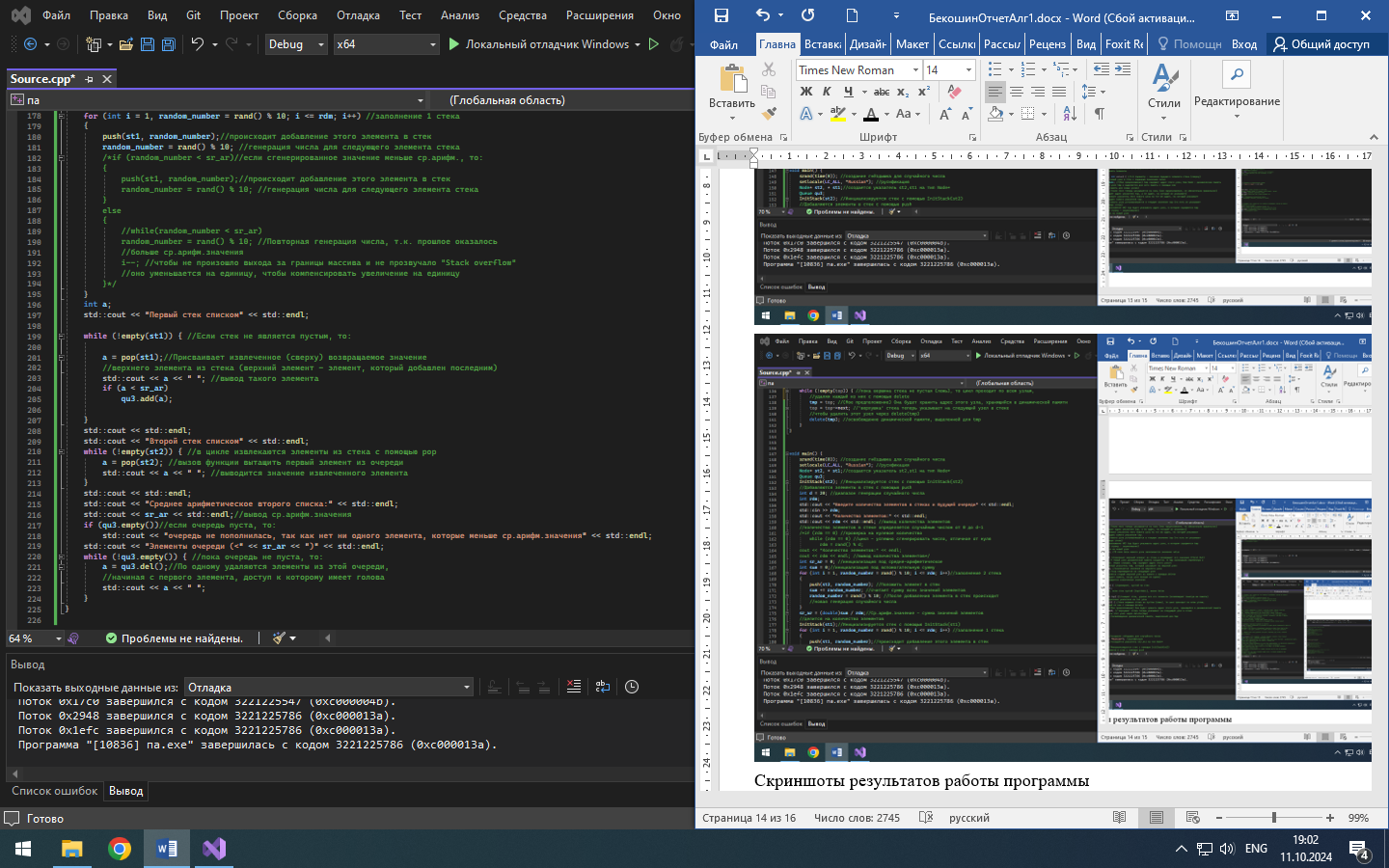


Рис. 13

Скриншоты результатов работы программы

Задача №1:

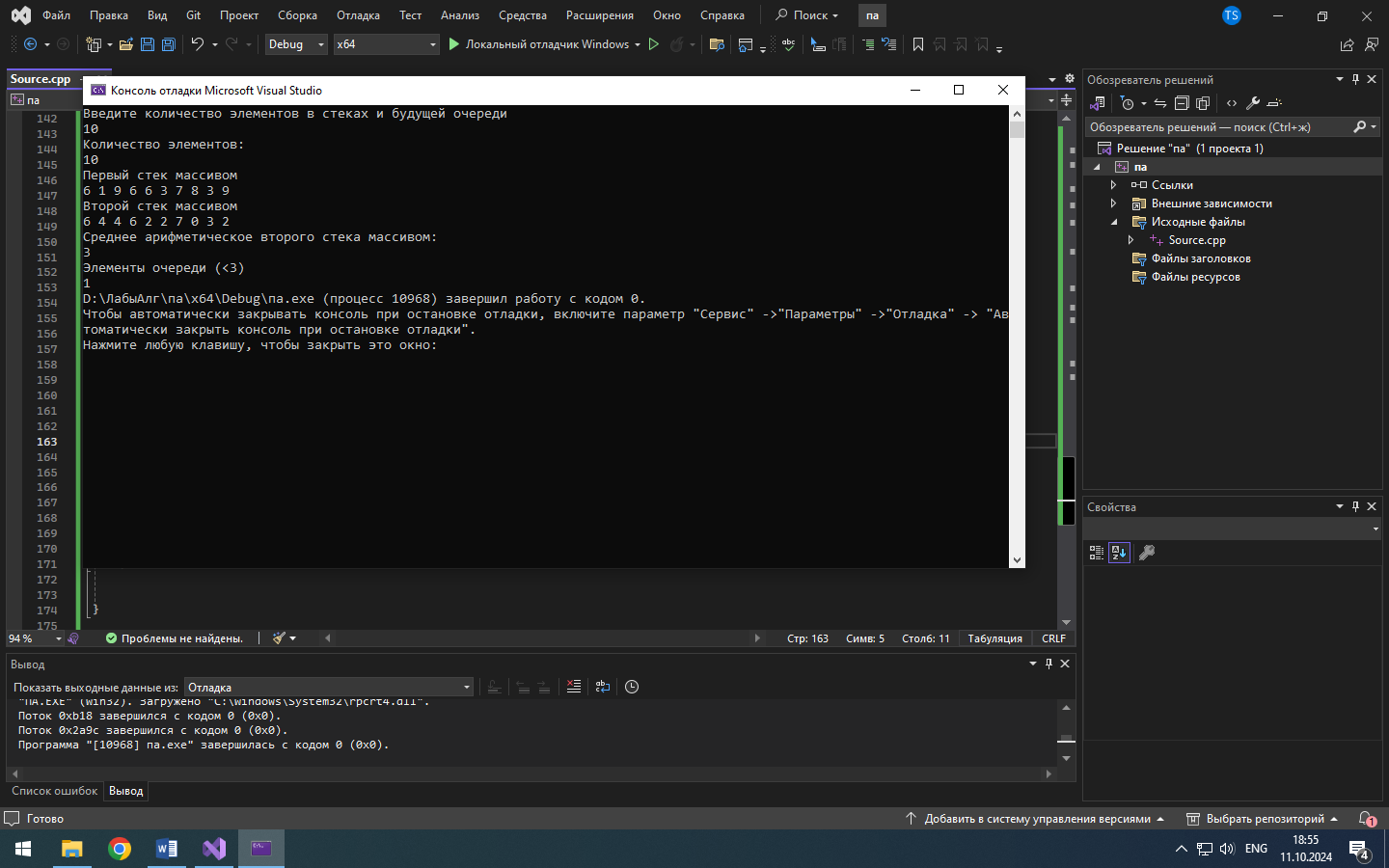


Рис. 14

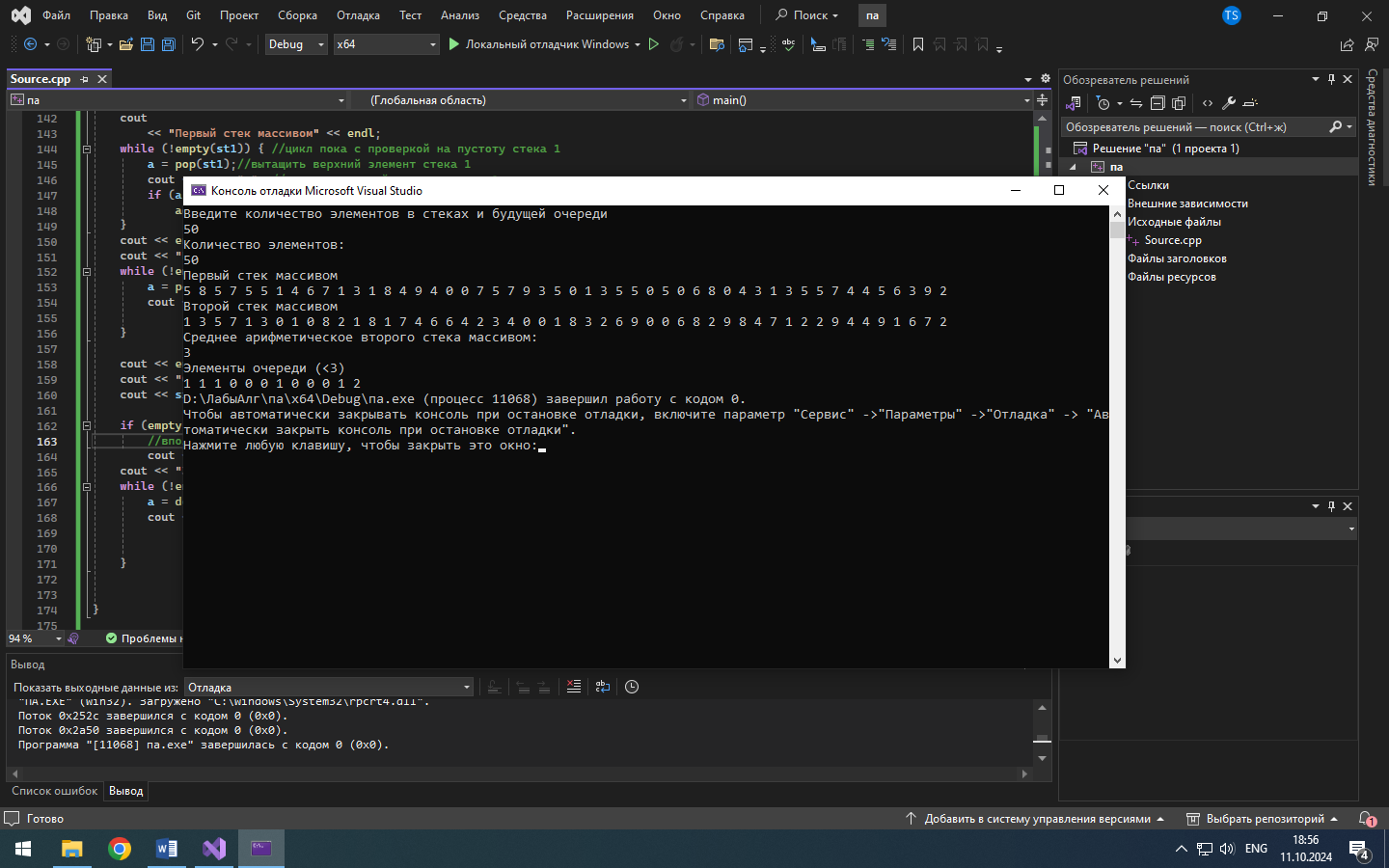


Рис. 15

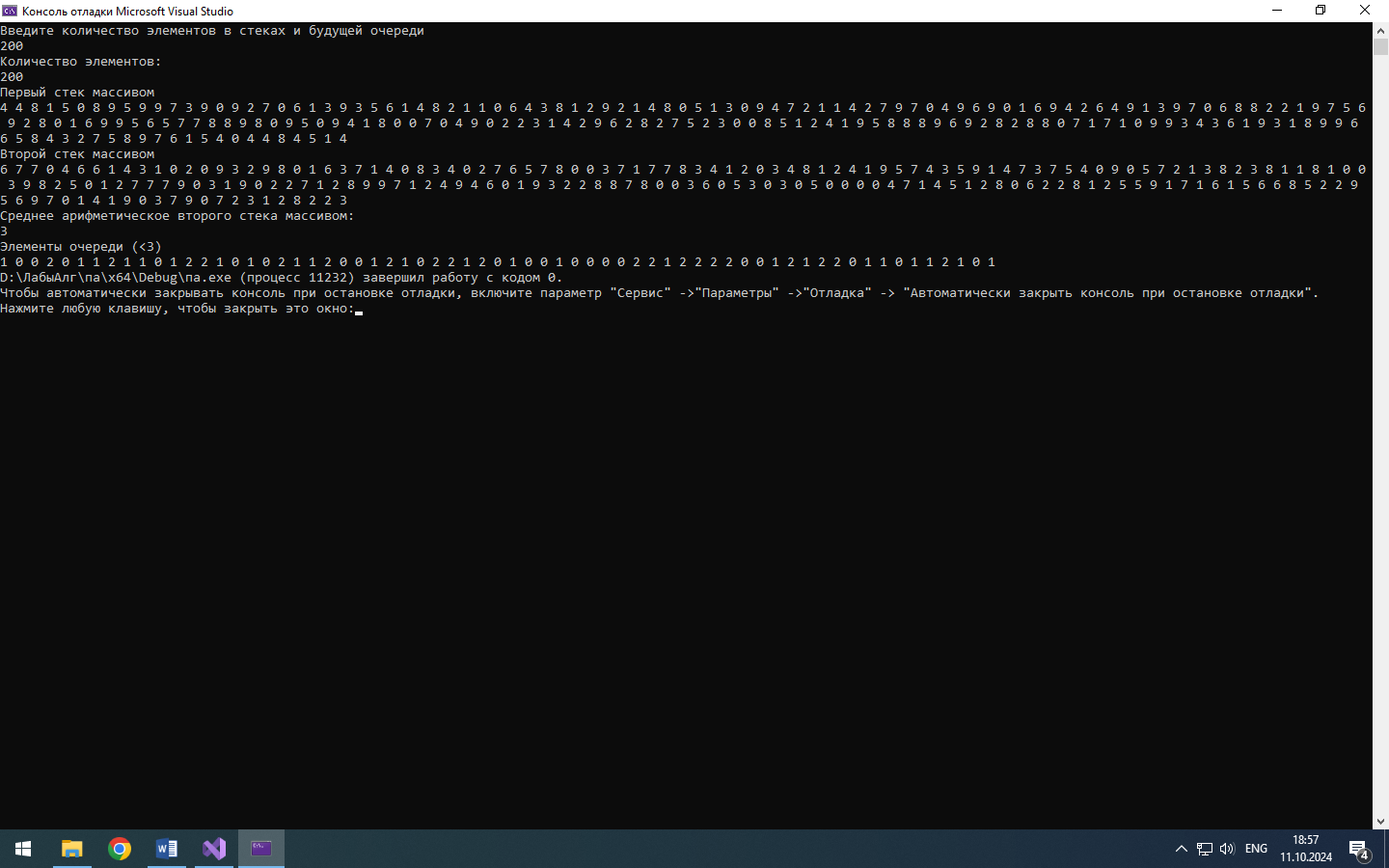


Рис. 16

Задача №2

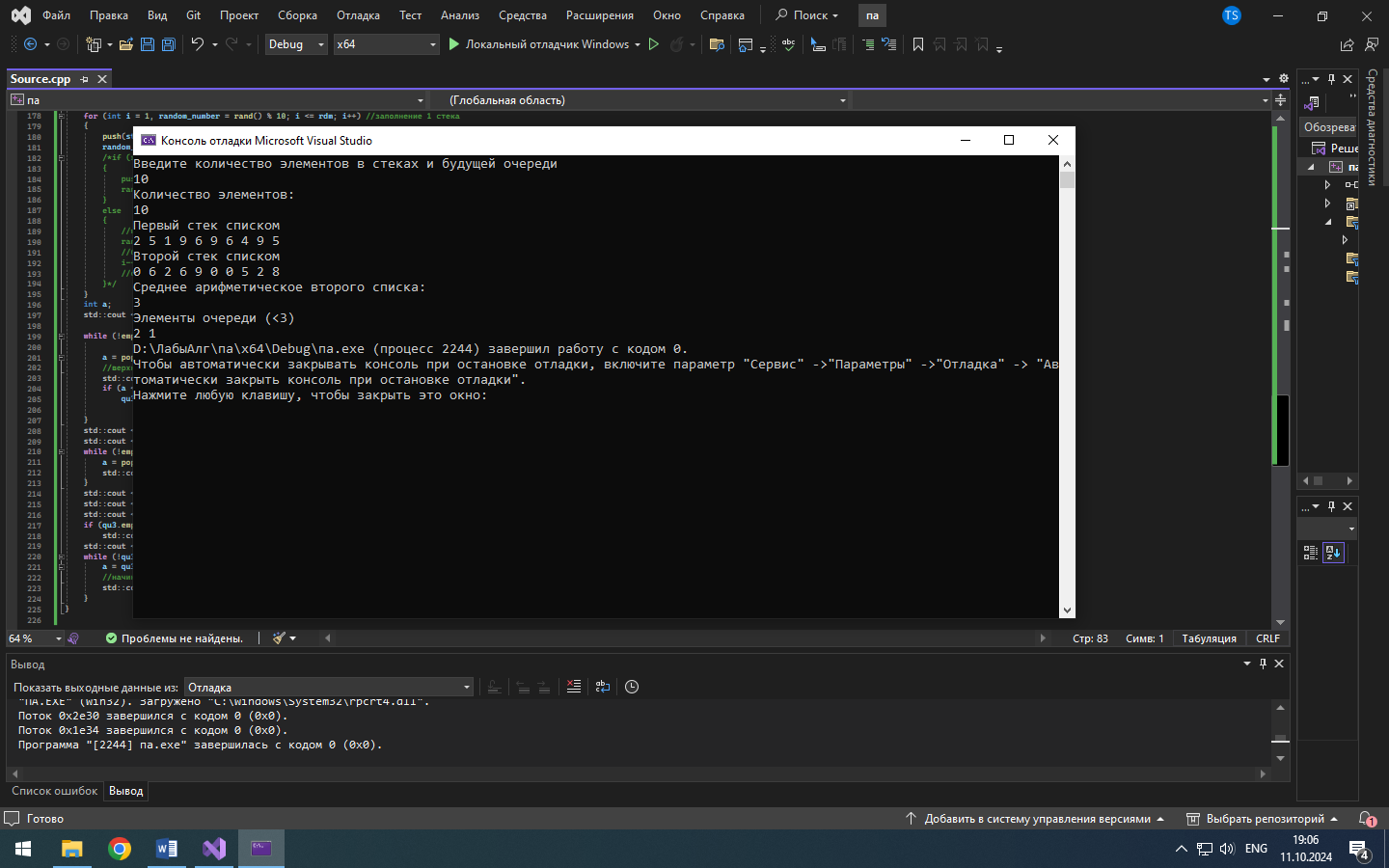


Рис. 17

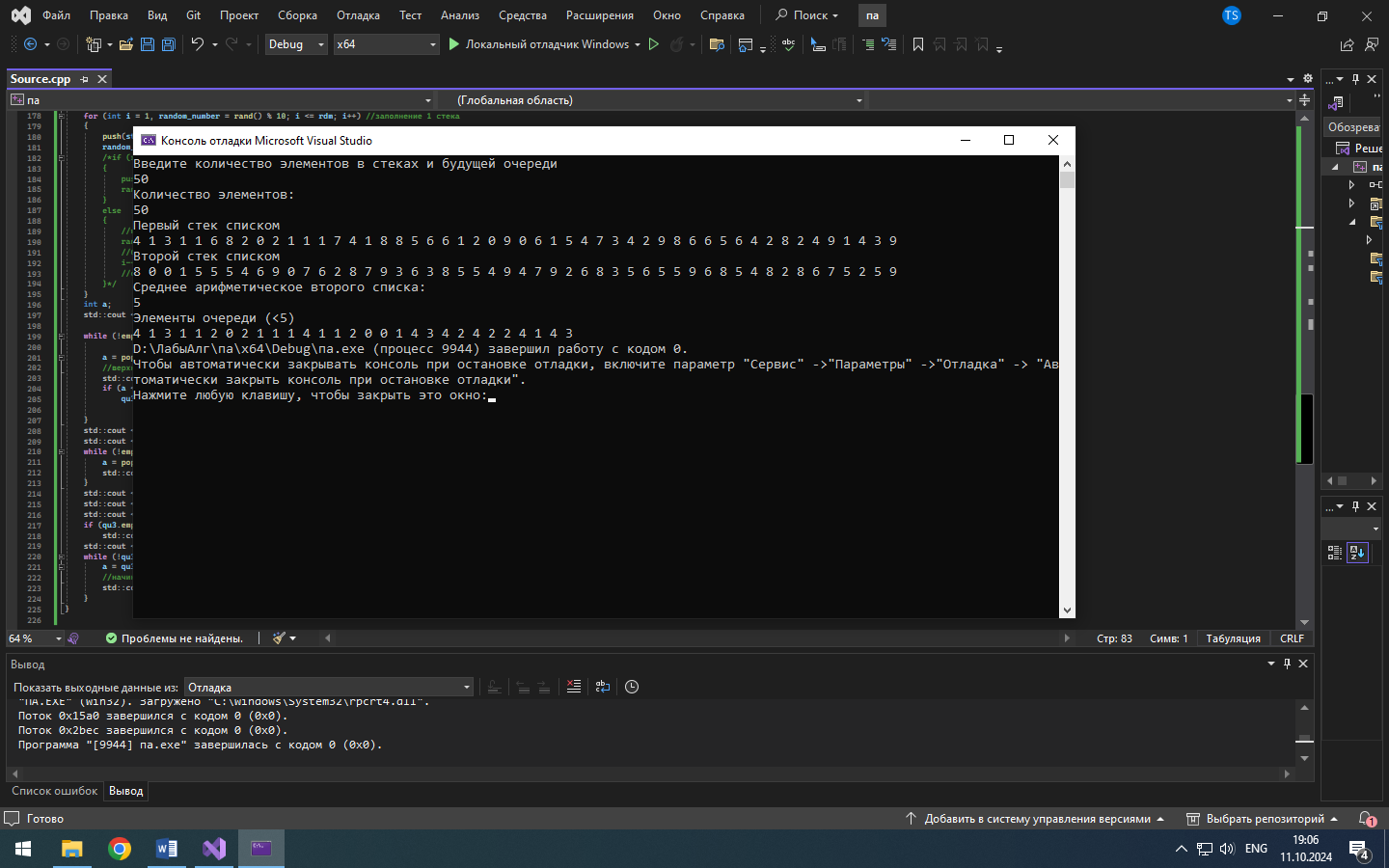


Рис. 18

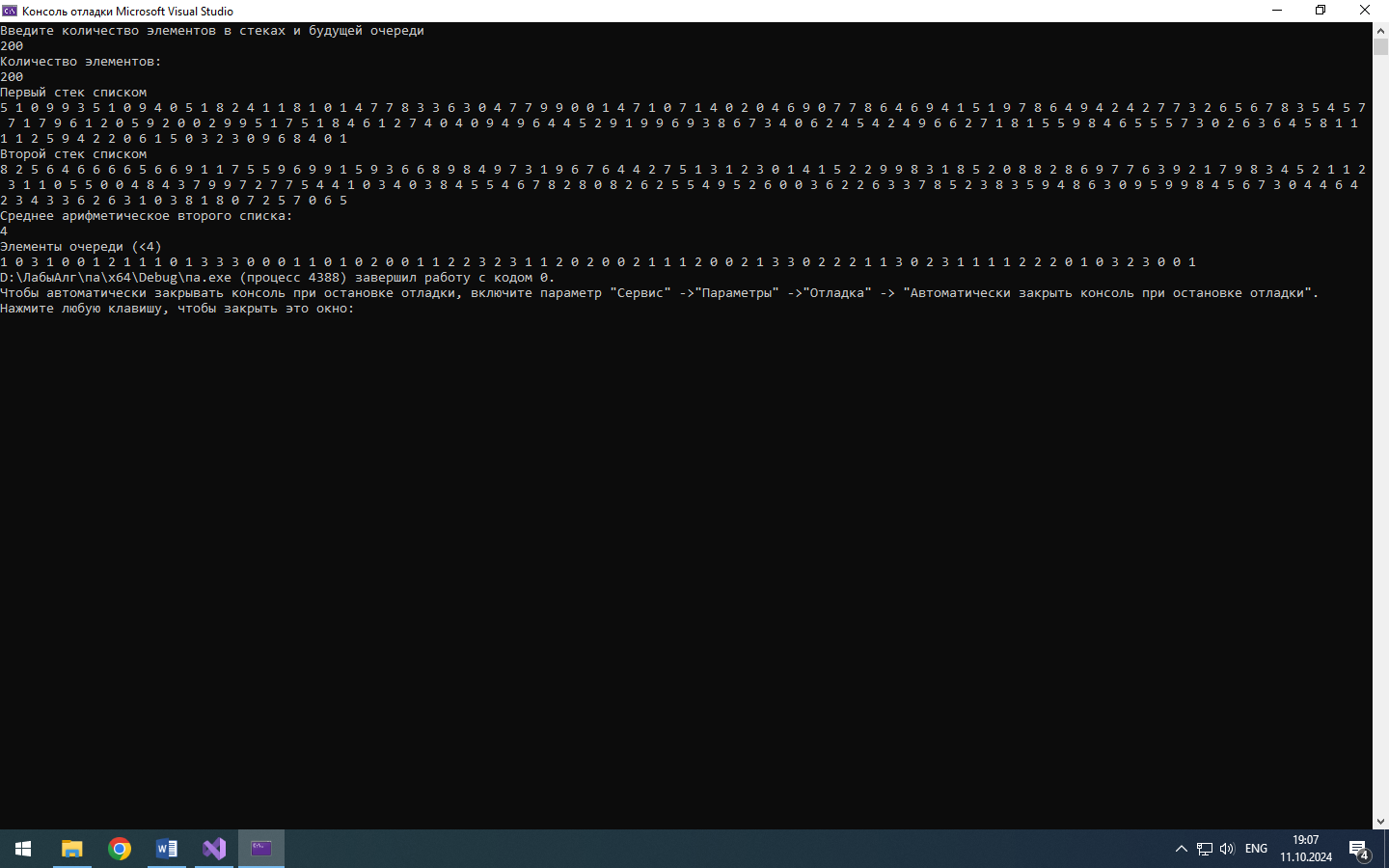


Рис. 19