**Лабоpатоpная pабота №1**

**Линейные списки**

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.

16. Даны стек и очеpедь целых чисел. Из элементов очереди, на котоpые

не делится сумма элементов стека сфоpмиpовать новый стек.

1. **Код программы решения с помощью массивов.**

#include <iostream>

#include <time.h>

using namespace std;

class Stack {

private:

int top;

int\* data;

int size;

int tmp = top;

public:

Stack(int capacity) {

data = new int[capacity];

top = -1;

tmp = -1;

size = capacity;

}

~Stack() {

delete[]data;

}

void push(int value) {

if (top < size - 1)

data[++top] = value;

tmp++;

}

bool empty() {

return top == -1;

}

int pop() {

if (!empty()) {

return data[top--];

}

else {

return -666;

}

}

bool canWatch() {

if (tmp == -1) {

tmp = top;

return false;

}

return true;

}

int watch() {

return data[tmp--];

}

void nullStack() {

top = -1;

}

};

class Queue {

private:

int\* data;

int head, tail, size, tmp;

int next(int n) {

return (n + 1) % size;

}

public:

Queue(int capacity) {

size = capacity + 1;

data = new int[size];

head = 0;

tail = capacity;

tmp = head;

}

bool empty() {

return next(tail) == head;

}

bool canWatch() {

if ( (tmp)==next(tail) || empty()) {

tmp = head;

return false;

}

return true;

}

void add(int value) {

if (next(next(tail)) == head) {

cout << "Queue overflow" << endl;

}

else {

tail = next(tail);

data[tail] = value;

}

}

int del() {

if (empty()) {

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

return 0;

}

else {

int d = data[head];

head = next(head);

return d;

}

}

int watch() {

int d = data[tmp];

tmp = next(tmp);

return d;

}

~Queue() {

delete[] data;

}

};

int main() {

srand(time(0));

//рандомное распледелние количества чисел в стеках и очередях

int nStack = 0; nStack = rand() % 100 + 50;

int nQueue = 0; nQueue = rand() % 10 + 50; cout << nQueue << endl;

//заполнение стэка1

Stack st(nStack);

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

st.push(rand() % 10);

}

// вывод стека1

cout << "Stack1: [ ";

while (st.canWatch()) {

cout << st.watch() << " ";

}

cout << "]\n";

//заполнение очереди

int k = 0;

int k1 = 0;

Queue q(nQueue);

cout << "Queue = [ ";

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

k1++;

k = rand() % 15 + 1;

q.add(k);

cout << k << " ";

}

cout << "]\n";

cout << "nQueue = "<< k1 << endl;

// вывод очереди

int k2 = 0;

cout << "Queue: [ ";

while (q.canWatch()) {

k2++;

cout << q.watch() << " ";

}

cout << "]\n";

cout << "nQueue = " << k2 << endl;

//Сумма элементов стека1

int sum = 0;

while (st.canWatch())

{

sum += st.watch();

}

cout << "Summa element stack1 = " << sum << endl;

//Заполнение стека2

Stack st2(nQueue);

while (q.canWatch()) {

int tmp = q.watch();

if ((sum % tmp) != 0) {

st2.push(tmp);

}

}

//Вывод стека2

if (st2.empty()) {

cout << "Stack2 is empty\n";

}

else {

cout << "Stack2: [ ";

while (st2.canWatch()) {

cout << st2.watch() << " ";

}

cout << "]\n";

}

system("pause");

return 0;

}

**Алгоритм**

Class Stack

Stack(int capacity) - конструктор по параметру размер стека.

~Stack() - деструктор

void push(int value) - добавление в конец стека value

bool empty() - проверка на пустоту стека

Int pop - удаление последнего элемента из стека.

bool canWatch() - проверка на читаемость стека.

Int watch() - передаёт считанный из стека элемент, не изменяя стек.

Void nullStack() - обнуление стека.

Class Queue

Queue(int capacity) - конструктор по параметру размера очереди.

~Queue() - деструктор

bool empty() - проверка на пустоту стека.

bool canWatch() - проверка на читаемость стека на читаемость

Int watch() - передаёт считанный из стека элемент, не изменяя очередь.

Int del() - удаление из начала очереди элемента

Void add(int value) - добавление в конец очереди value.

Алгоритм

Генируем размер стека (nStack) в диапазоне от 50 до 149 и размер очереди (nQueue) в диапазоне от 5 до 14.

Создаём и заполняем стек st размером nStack рандомными элементами в диапазоне от 0 до 9, а потом выводим элементы стека.

Создаём переменную для подсчёта суммы элементов стека и подсчитываем сумму стека st. Выводим сумму стека sum.

Создаём и заполняем очередь q размером nQueue рандомными элементами в диапазоне от 1 до 15, а потом выводим элементы

этой очереди.

Потом создаём стек st2 и заполняем его элементами очереди q, на которые нацело не делится сумма стека st (sum).

Если такой стек st2 оказался пустым выводим, что st2 пустой, иначе выводим элементы st2.

Алгоритм программы.

1. Случайно генерируем размеры стека в диапазоне от 50 до 149 и очереди в диапазоне от 5 до 14.

2 Создаём экземпляр класса Stack с именем st и выделяем память для хранения nStack int элементов.

3.Заполняем st рандомными значениями в диапазоне от 0 до 9.

4.Выводим стек st.

5.Объявляем переменную sum = 0. для подсчёта суммы. С помощью цикла while считаем сумму в стеке.

6.Создаём экземпляр класса Queue с именем q и выделяем память для хранения nQueue int элементов.

7.заполняем очередь q рандомными значениями от 1 до 15.

8.Выводим очередь q.

9.Создаём экземпляр класса Stack st2 и выделяем память для хранения nQueue int элементов.

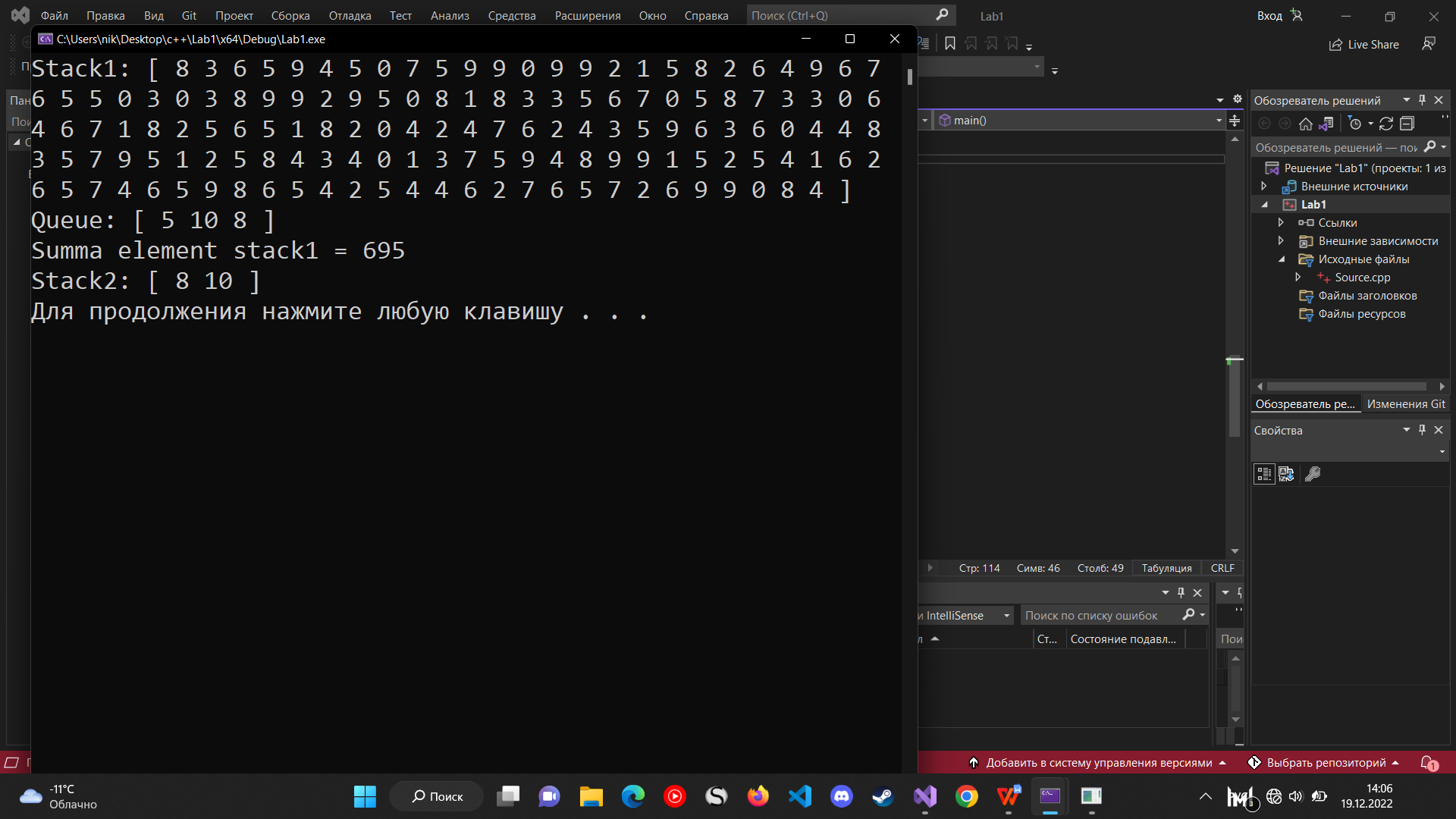
10.Заполняем стек значениями из очереди, которые не делят сумму стека st на цело.

11. Выводим стекst2

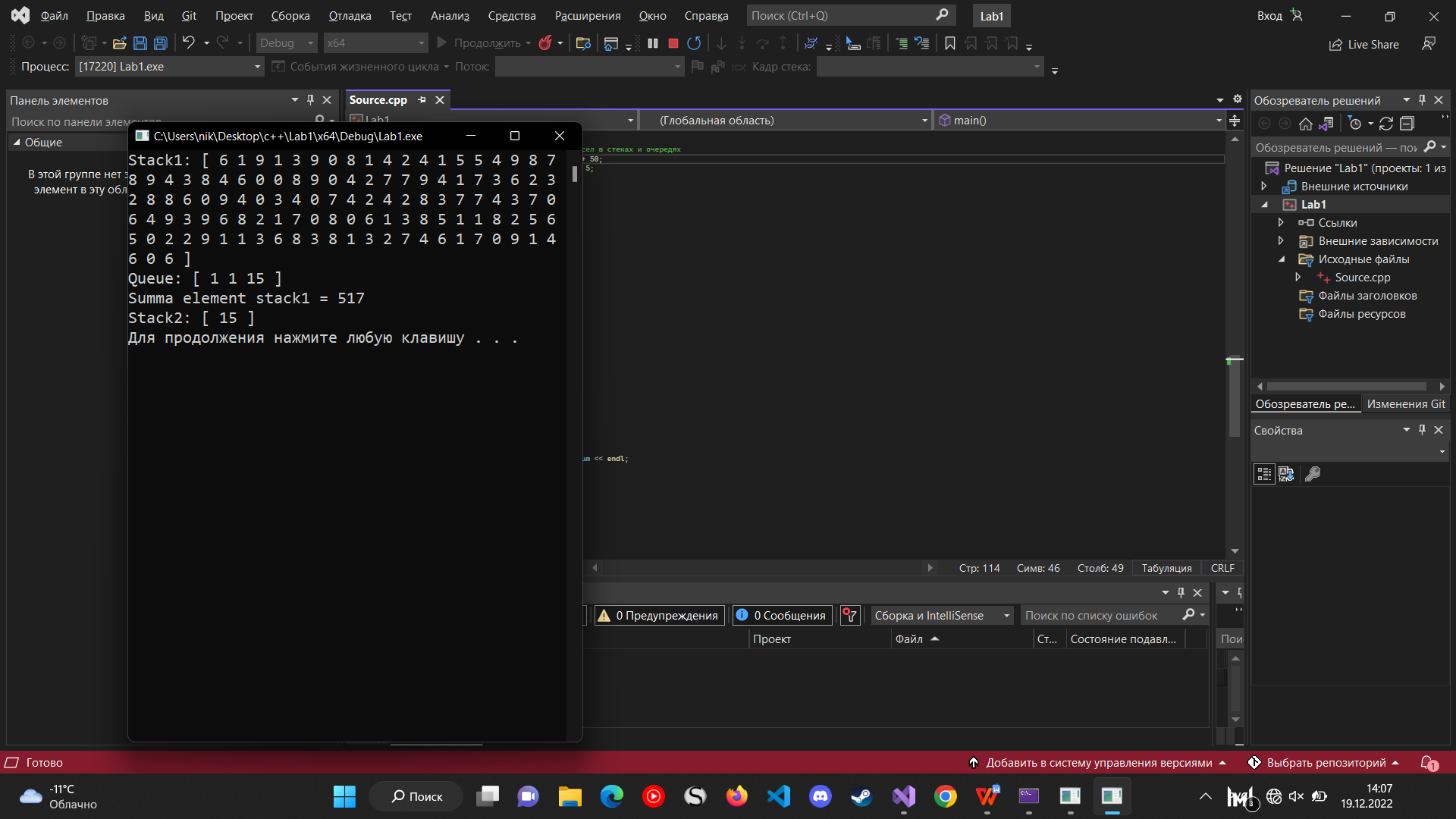
Конец программы.

**Тесты**

Тест 1



Тест 2



1. **Код программы решения через динамические списки.**

#include <iostream>

#include <time.h>

using namespace std;

//Стек

class NodeStack {

private:

struct Node {

int data;

Node\* next;

bool watched;

};

Node\* top;

public:

//конструктор по умолчанию

NodeStack() {

top = NULL;

}

//добавление в стек

void push(int value) {

Node\* tmp = new Node;

tmp->next = top;

top = tmp;

tmp->data = value;

tmp->watched = false;

}

//удаление из стека

int pop() {

Node\* tmp = top;

int d = top->data;

top = top->next;

delete(tmp);

return d;

}

//проверка на пустоту стека

bool empty() {

return top == NULL;

}

//проверка на возможность считывания

bool canWatch() {

Node\* tmp = top;

while (tmp != NULL) {

if (tmp->watched == false) {

return true;

}

tmp = tmp->next;

}

return false;

}

//считывание элементов стека

int watch() {

Node\* tmp = top;

while (tmp->watched) {

tmp = tmp->next;

}

int d = tmp->data;

tmp->watched = true;

return d;

}

//очистка для прочитанного стека

void clearWatch() {

Node\* tmp = top;

while (tmp != NULL) {

if (tmp->watched == false) {

break;

}

tmp->watched = false;

tmp = tmp->next;

}

}

//обнуление стека

void nullStack() {

Node\* tmp;

while (!empty()) {

tmp = top;

top = top->next;

delete(tmp);

}

};

};

//Очередь

class Queue {

private:

struct Node {

int data;

Node\* next;

};

Node\* head, \* tail,\* tmp;

public:

//конструктор очереди

Queue() {

head = NULL;

tail = NULL;

tmp = NULL;

}

//проверка на пустоту очереди

bool empty() {

return head == NULL;

}

//добавление элемента в конец очереди

void add(int value) {

if (empty()) {

head = new Node;

head->data = value;

head->next = NULL;

tail = head;

tmp = head;

}

else {

tail->next = new Node;

tail = tail->next;

tail->data = value;

tail->next = NULL;

}

}

//удаление элемента в начале очереди

int del() {

if (empty()) {

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

return 0;

}

else {

int d = head->data;

Node\* tmp = head;

head = head->next;

delete(tmp);

return d;

}

}

//проверка на читаемость очереди

bool canWatch() {

if (tmp == NULL) {

tmp = head;

return false;

}

return true;

}

//считывание элемента стека

int watch() {

if (canWatch()) {

int d = tmp->data;

tmp = tmp->next;

return d;

}

return 0;

}

//обнуление для очереди

void nullQueue() {

Node\* tmp;

while (!empty()) {

tmp = head;

head = head->next;

delete (tmp);

}

}

};

int main() {

srand(time(0));

//рандомное распледелние количества чисел в стеках и очередях

int nStack = 0; nStack = rand() % 100 + 50;

int nQueue = 0; nQueue = rand() % 10 + 5;

//создание стека1 и его заполнение

NodeStack st;

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

st.push(rand()%10);

}

// вывод стека1

cout << "Stack = [ ";

while (st.canWatch()) {

cout << st.watch() << " ";

}

cout << "]\n";

//очистка прочитанного стека1

st.clearWatch();

//Вывод суммы элементов стека

int sum = 0;

while (st.canWatch()) {

sum += st.watch();

}

cout << "Summa element stack1 = " << sum << endl;

//Создание и заполнение очереди

Queue q;

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

q.add(rand() % 15 + 1);

}

//Вывод элементов очереди

cout << "Queue = [ ";

while (q.canWatch()) {

cout << q.watch() << " ";

}

cout << "]\n";

//Создание и заполнение стека2 согласно условию

NodeStack st2;

while (q.canWatch()) {

int tmp = q.watch();

if (sum % tmp != 0) {

st2.push(tmp);

}

}

//Вывод элементов стека2

if (st2.empty()) {

cout << "Stack2 is empty\n";

}

else {

cout << "Stack2: [ ";

while (st2.canWatch()) {

cout << st2.watch() << " ";

}

cout << "]\n";

}

system("pause");

return 0;

}

**Алгоритм**

Class NodeStack

NodeStack() - конструктор по умолчанию

void push(int value) - добавление в конец стека value и метки false.

bool empty() - проверка на пустоту стека

Int pop - удаление последнего элемента из стека.

bool canWatch() - проверка на читаемость стека.

void clearWatch() - очищает считанный стек от меток true.

Int watch() - передаёт считанный из стека элемент, не изменяя стек,но меняет метку на true.

Void nullStack() - обнуление стека.

Class Queue

Queue() - конструктор по умолчанию.

bool empty() - проверка на пустоту стека.

bool canWatch() - проверка на читаемость стека на читаемость

Int watch() - передаёт считанный из стека элемент, не изменяя очередь.

Int del() - удаление из начала очереди элемента

Void add(int value) - добавление в конец очереди value.

Void nullQueue() - обнуление для очереди.

Алгоритм

* Генируем размер стека (nStack) в диапазоне от 50 до 149 и размер очереди (nQueue) в диапазоне от 5 до 14.
* Создаём и заполняем стек st размером nStack рандомными элементами в диапазоне от 0 до 9, а потом выводим элементы стека.
* Очищаем наш прочитанный стек от меток, для дальнейшего прочтения.
* Создаём переменную для подсчёта суммы элементов стека и подсчитываем сумму стека st. Выводим сумму стека sum.
* Создаём и заполняем очередь q размером nQueue рандомными элементами в диапазоне от 1 до 15, а потом выводим элементы
* этой очереди.
* Потом создаём стек st2 и заполняем его элементами очереди q, на которые нацело не делится сумма стека st (sum).
* Если такой стек st2 оказался пустым выводим, что st2 пустой, иначе выводим элементы st2.

1. Случайно генерируем размер стека (nStack) в диапазоне от 50 до 149 и размер очереди(nQueue) в диапазоне от 5 до 14.

2 Создаём экземпляр класса Stack с именем st.

3.Заполняем st рандомными значениями в диапазоне от 0 до 9.

4.Выводим стек st.

5. Очищаем прочитанный стек st от меток с помощью функции void clearWatch().

6.Объявляем переменную sum = 0. для подсчёта суммы. С помощью цикла while считаем сумму в стеке.

7.Создаём экземпляр класса Queue с именем q .

8.Заполняем очередь q рандомными значениями от 1 до 15.

9.Выводим очередь q.

10.Создаём экземпляр класса Stack st2.

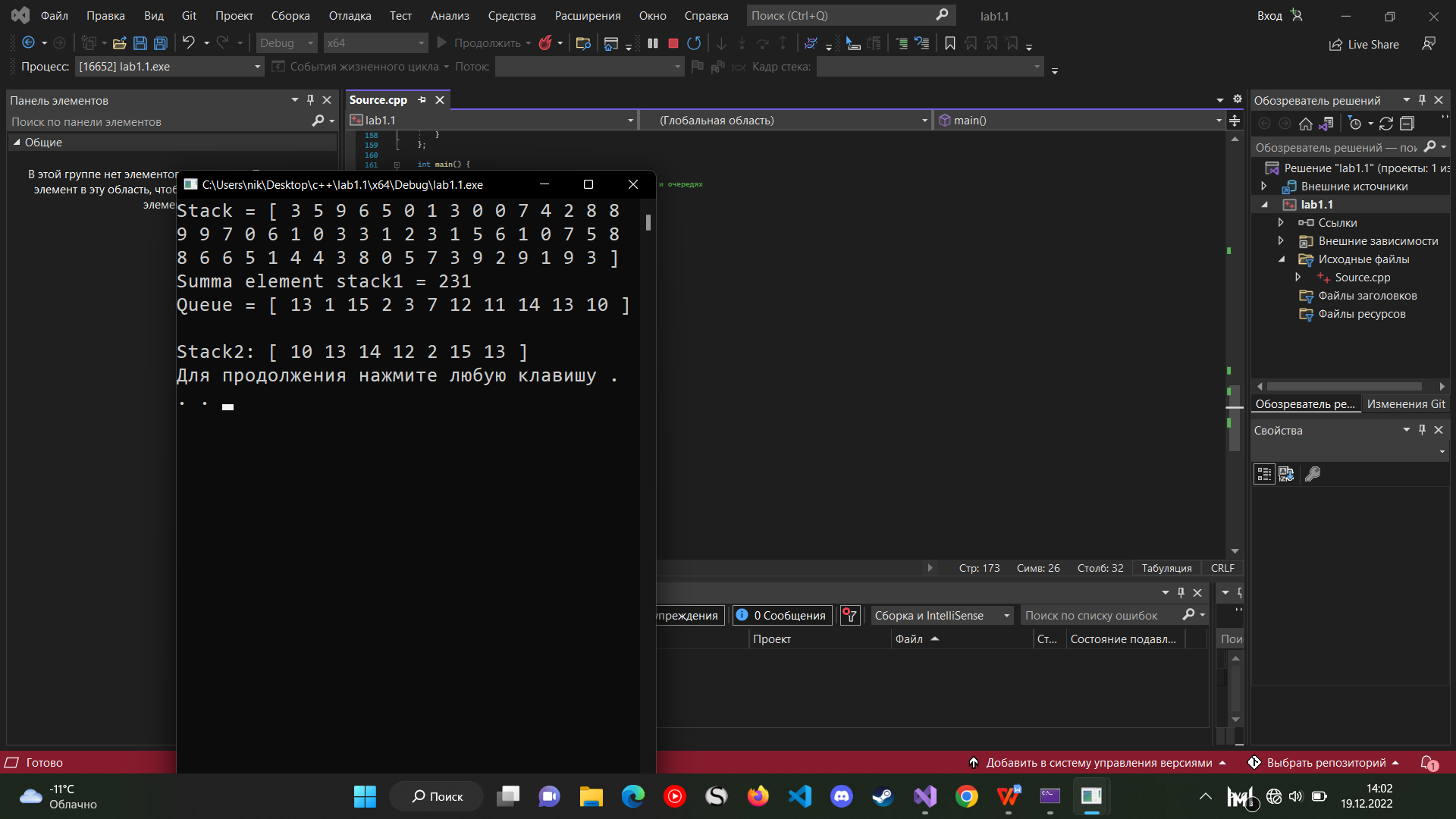
9.Заполняем стек значениями из очереди, которые не делят сумму стека st на цело.

10. Выводим стек st2

11. Конец программы.

**Тесты программы**

Тест 1.



Тест2.

