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

ПНИПУ

Лабораторная работа №10  
«Информационные динамические структуры»

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

Проверила:   
доцент кафедры ИТАС   
О.А. Полякова

Пермь 2024 г.

**Постановка задачи:**

16 вар

1)Записи в линейном списке содержат ключевое поле типа

\*char(строка символов). Сформировать **двунаправленный**

**список.** Удалить элемент с заданным ключом. Добавить К

элементов в конец списка.

2) Однонаправленный список

3) Стек

4) Очередь

**Двунаправленный**

**список.**

**Анализ задачи:**

* Создаем структуру Node и List
* Создаем методы для работы со списками(deletekey, add\_node, print\_list, list\_filesave, list\_fileread, list\_del)
* Используя созданные методы, работаем со списками по заданию

**Блок-схема:**

**Код на языке с++:**

#include <fstream>

#include <iostream>

#include <string>

using namespace std;

struct Node

{

char\* key = new char[1];

Node\* prev\_node\_ptr = nullptr;

Node\* next\_node\_ptr = nullptr;

};

struct List

{

Node\* head\_node = nullptr;

Node\* tail\_node = nullptr;

int size = 0;

};

void deletekey(List& list, char key)

{

if (list.head\_node == nullptr)

{

cout << "Лист пуст" << endl;

return;

}

Node\* remove\_node = list.head\_node;

while ((remove\_node != nullptr) && (\*(remove\_node->key) != key))

{

remove\_node = remove\_node->next\_node\_ptr;

}

if (remove\_node == nullptr)

{

cout << "Нет такого ключа" << endl;

return;

}

if (list.head\_node == remove\_node)

{

list.head\_node = remove\_node->next\_node\_ptr;

}

if (list.tail\_node == remove\_node)

{

list.tail\_node = remove\_node->prev\_node\_ptr;

}

if (remove\_node->prev\_node\_ptr != nullptr)

{

remove\_node->prev\_node\_ptr->next\_node\_ptr = remove\_node->next\_node\_ptr;

}

if (remove\_node->next\_node\_ptr != nullptr)

{

remove\_node->next\_node\_ptr->prev\_node\_ptr = remove\_node->prev\_node\_ptr;

}

delete remove\_node;

list.size--; // Уменьшаем размер списка после удаления элемента

}

void add\_node(List& list, int index, char str)

{

Node\* new\_node = new Node;

\*(new\_node->key) = str;

list.size++;

if (list.head\_node == nullptr)

{

list.head\_node = new\_node;

list.tail\_node = new\_node;

return;

}

if ((index == -1) || (index == 0))

{

new\_node->next\_node\_ptr = list.head\_node;

list.head\_node->prev\_node\_ptr = new\_node;

list.head\_node = new\_node;

return;

}

if (index >= list.size - 1)

{

new\_node->prev\_node\_ptr = list.tail\_node;

list.tail\_node->next\_node\_ptr = new\_node;

list.tail\_node = new\_node;

return;

}

int counter = 0;

int from\_tail = list.size - 2 - index;

Node\* current\_node;

if (index < from\_tail)

{

current\_node = list.head\_node;

while (counter != index)

{

current\_node = current\_node->next\_node\_ptr;

++counter;

}

}

else

{

current\_node = list.tail\_node;

while (counter != from\_tail)

{

current\_node = current\_node->prev\_node\_ptr;

++counter;

}

}

new\_node->next\_node\_ptr = current\_node->next\_node\_ptr;

new\_node->prev\_node\_ptr = current\_node;

current\_node->next\_node\_ptr->prev\_node\_ptr = new\_node;

current\_node->next\_node\_ptr = new\_node;

}

void print\_list(List& list)

{

Node\* current\_node = list.head\_node;

if (current\_node == nullptr)

{

cout << "Лист пуст" << endl;

return;

}

cout << "[";

while (current\_node != nullptr)

{

cout << \*(current\_node->key) << ", ";

current\_node = current\_node->next\_node\_ptr;

}

cout << "\b \b\b \b]" << endl;

// Удаление указателя current\_node не нужно здесь

}

void list\_filesave(List& list)

{

Node\* current\_node = list.head\_node;

ofstream f("list\_filesave.dat");

while (current\_node != nullptr)

{

f << current\_node->key << '\n';

current\_node = current\_node->prev\_node\_ptr;

}

f.close();

}

void list\_fileread(List& list)

{

char buffer\_char[1];

string buffer\_str;

ifstream f("list\_savestate.dat");

while (!f.eof())

{

getline(f, buffer\_str);

buffer\_char[0] = buffer\_str[0];

if (buffer\_str != "")

{

add\_node(list, -1, buffer\_str[0]);

}

}

f.close();

}

void list\_del(List& list)

{

Node\* remove\_node;

while ((list).head\_node != nullptr)

{

remove\_node = (list).head\_node;

(list).head\_node = (list).head\_node->next\_node\_ptr;

delete remove\_node;

}

}

int main()

{

setlocale(0, "");

system("chcp 1251");

system("cls");

int n, k;

string str;

char ch;

List list;

cout << "Сколько элементов: ";

cin >> n;

cin.ignore();

for (int i = 1; i <= n; ++i)

{

cout << "Введите элемент: ";

cin >> ch;

add\_node(list, i, ch);

}

print\_list(list);

cout << "Введите ключ для удаления: ";

while (!(cin >> ch)) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

}

deletekey(list, ch);

print\_list(list);

cout << " индекс и кол-во(К) элементов для добавления: ";

cin >> n >> k;

cin.ignore();

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

{

cout << "Введите элемент: ";

getline(cin, str);

ch = str[0];

add\_node(list, i, ch);

}

print\_list(list);

cout << "Лист созранен в файл" << endl;

list\_filesave(list);

list\_del(list);

print\_list(list);

cout << "лист выгружен из файла" << endl;

list\_fileread(list);

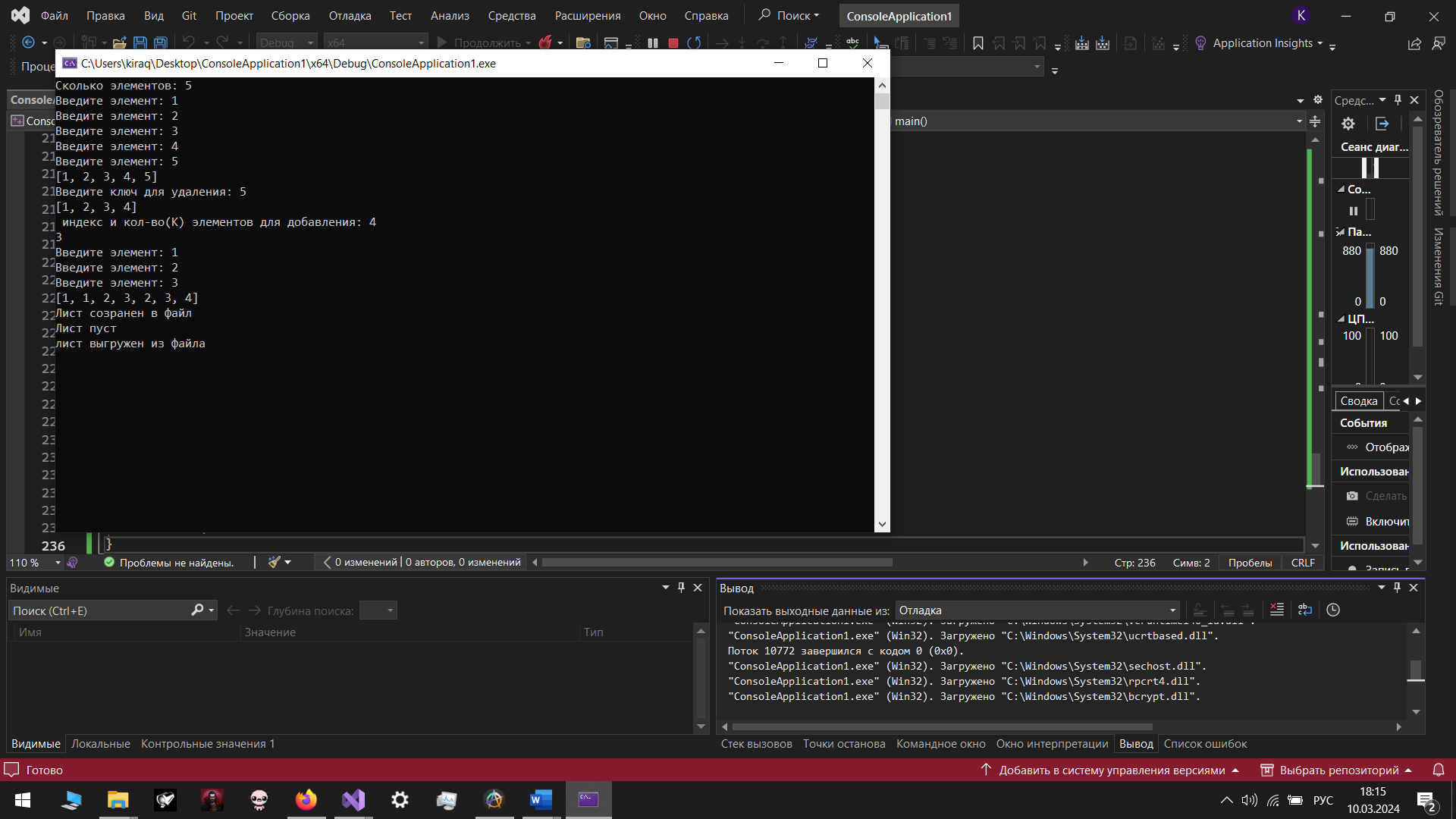
print\_list(list);

list\_del(list);

return 0;

}

**Пример работы кода:**



**Вывод:**

Код работает как надо

**Однонаправленный список**

**Анализ задачи:**

* Создаем структуру Node и List
* Создаем методы для работы со списками(deletekey, add\_node, print\_list, list\_filesave, list\_fileread, list\_del)
* Используя созданные методы, работаем со списками по заданию

**Блок-схема:**

**Код на языке с++:**

#include <iostream>

#include <string>

using namespace std;

struct Node

{

char\* key;

Node\* next;

};

struct List

{

Node\* head;

Node\* tail;

int size;

};

void pop\_front(List& list)

{

if (list.head == nullptr)

{

return;

}

Node\* ptr = list.head;

list.head = list.head->next;

delete ptr;

list.size--;

}

void pop\_back(List& list)

{

if (list.tail == nullptr)

{

return;

}

if (list.head == list.tail)

{

delete list.head;

list.head = list.tail = nullptr;

list.size = 0;

return;

}

Node\* ptr = list.head;

while (ptr->next != list.tail)

ptr = ptr->next;

delete list.tail;

ptr->next = nullptr;

list.tail = ptr;

list.size--;

}

Node\* getAt(List& list, int index)

{

if (index < 0 || index >= list.size)

{

return nullptr;

}

Node\* ptr = list.head;

int n = 0;

while (ptr && index != n)

{

ptr = ptr->next;

n++;

}

return ptr;

}

void add\_node(List& list, int index, char data)

{

if (index < 0 || index > list.size)

{

return;

}

Node\* left = getAt(list, index - 1);

Node\* right = (left == nullptr) ? list.head : left->next;

Node\* ptr = new Node;

ptr->key = new char[2];

ptr->key[0] = data;

ptr->key[1] = '\0';

if (left == nullptr)

{

list.head = ptr;

}

else

{

left->next = ptr;

}

ptr->next = right;

if (right == nullptr)

{

list.tail = ptr;

}

list.size++;

}

void delete\_key(List& list, int index)

{

if (index < 0 || index >= list.size)

{

return;

}

if (index == 0)

{

pop\_front(list);

return;

}

Node\* left = getAt(list, index - 1);

if (left == nullptr || left->next == nullptr)

{

return;

}

Node\* ptr = left->next;

left->next = ptr->next;

if (ptr == list.tail)

{

list.tail = left;

}

delete ptr;

list.size--;

}

void print\_list(List& list)

{

Node\* ptr = list.head;

if (ptr == nullptr)

{

cout << "Лист пуст" << endl;

return;

}

cout << "[";

while (ptr != nullptr)

{

cout << ptr->key[0] << ", ";

ptr = ptr->next;

}

cout << "\b \b\b \b]" << endl;

}

int main()

{

setlocale(0, "");

system("chcp 1251");

system("cls");

int k;

string str;

List list = { nullptr, nullptr, 0 };

int n;

char ch;

cout << "Сколько элементов: ";

cin >> n;

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

{

cout << "Введите элемент: ";

cin >> ch;

add\_node(list, i, ch);

}

print\_list(list);

cout << "Введите индекс для удаления: ";

cin >> n;

delete\_key(list, n);

print\_list(list);

cout << " индекс и кол-во(К) элементов для добавления: ";

cin >> n >> k;

cin.ignore();

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

{

cout << "Введите элемент: ";

getline(cin, str);

ch = str[0];

add\_node(list, i, ch);

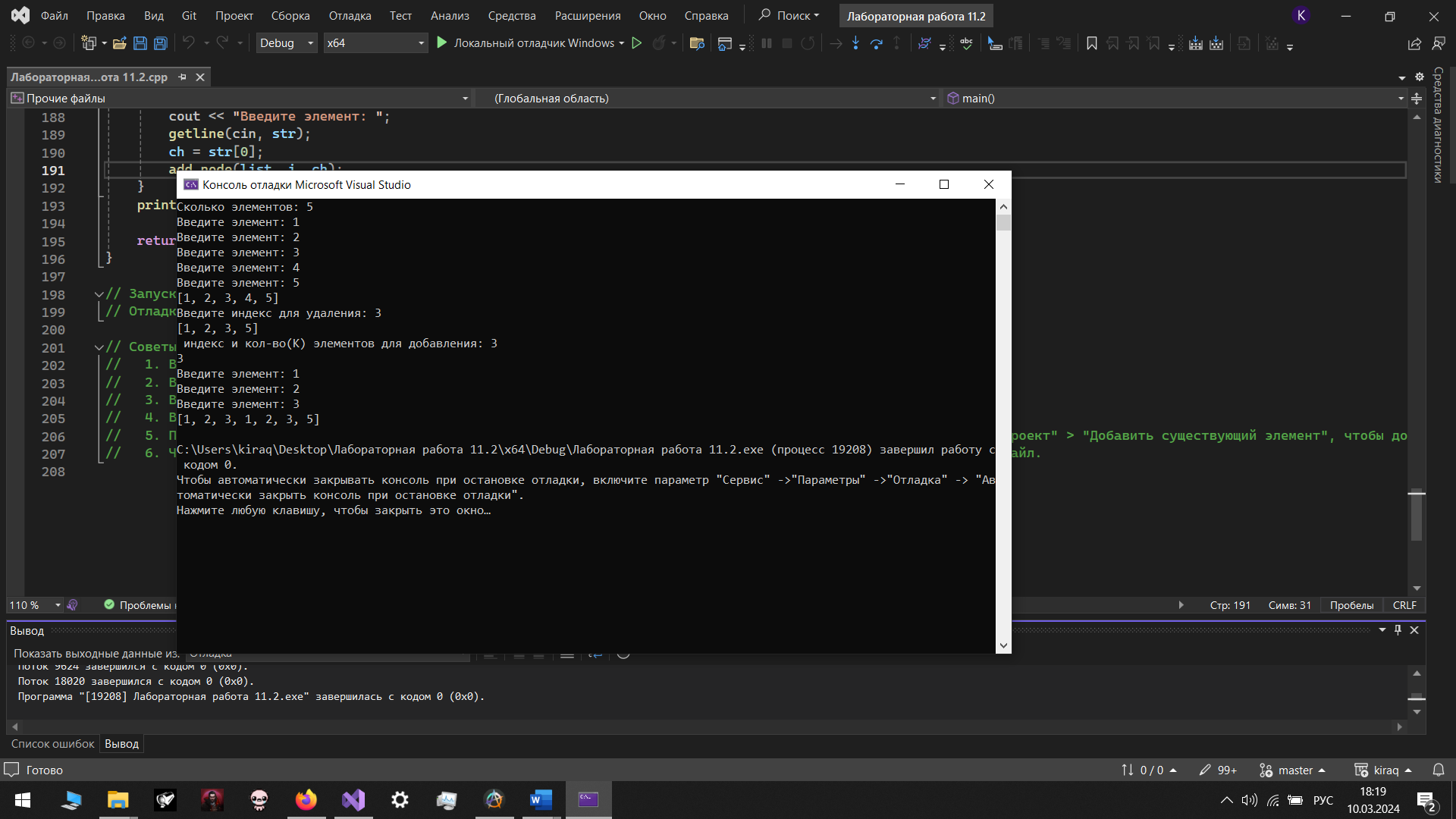
}

print\_list(list);

return 0;

}

**Пример работы кода:**



**Вывод:**

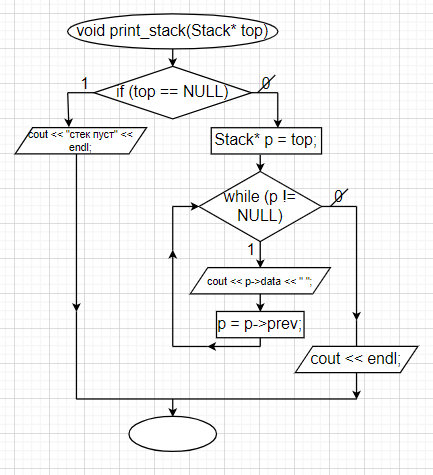
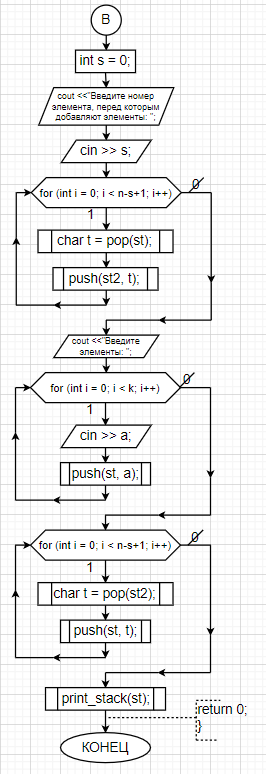
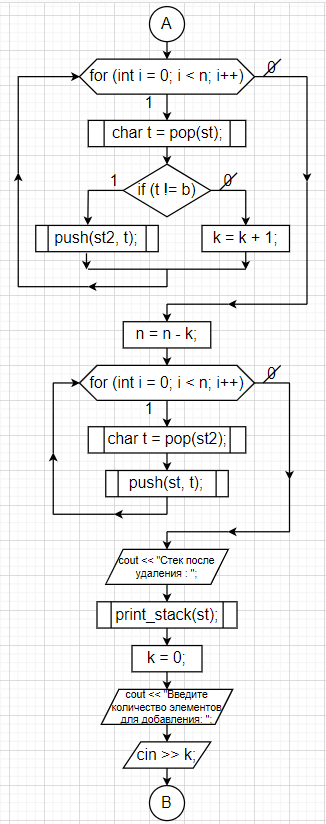
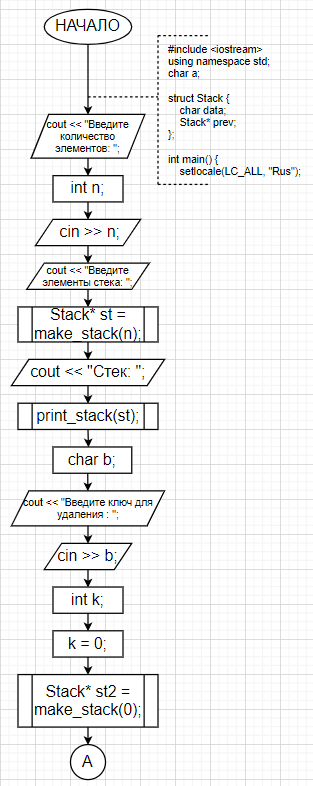
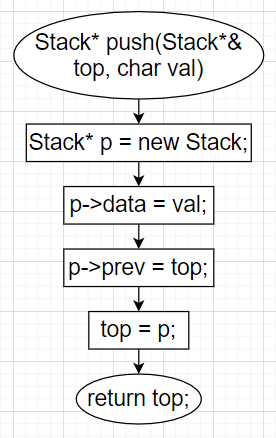
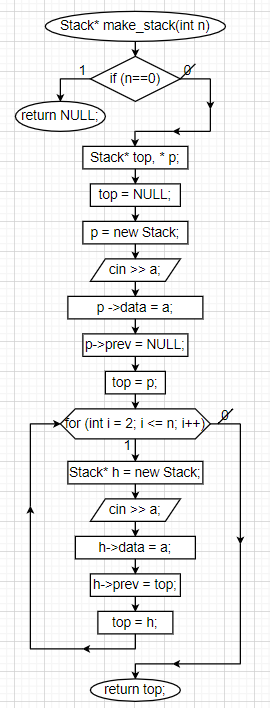
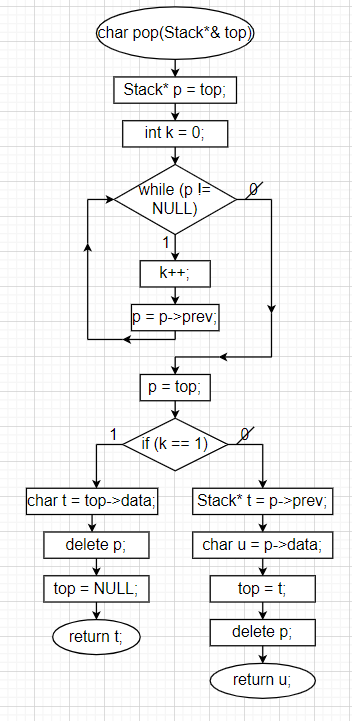
Код работает как надо

**Стеки**

**Анализ задачи:**

* Создаем структуру Stack
* Создаем методы для работы со списками(made\_stack, push, print\_stack, pop)
* Используя созданные методы, работаем со стеком по заданию

**Блок-схема:**

**Код на языке с++:**

#include <iostream>

using namespace std;

struct Stack {

char data;

Stack\* prev;

};

Stack\* made\_stack(int n) {

char a;

if (n == 0) {

return NULL;

}

else {

Stack\* top = NULL;

Stack\* p;

cin >> a;

p = new Stack;

p->data = a;

p->prev = NULL;

top = p;

for (int i = 2; i <= n; i++) {

Stack\* h = new Stack;

cin >> a;

h->data = a;

h->prev = top;

top = h;

}

return top;

}

}

Stack\* push(Stack\*& top, char val) {

Stack\* p = new Stack;

p->data = val;

p->prev = nullptr;

if (top == nullptr) {

top = p;

}

else {

Stack\* current = top;

while (current->prev != nullptr) {

current = current->prev;

}

current->prev = p;

}

return top;

}

void print\_stack(Stack\* top) {

if (top == nullptr) {

cout << "Стек пуст." << endl;

}

else {

Stack\* p = top;

cout << "Стек: ";

while (p != nullptr) {

cout << p->data << " ";

p = p->prev;

}

cout << endl;

}

}

char pop(Stack\*& top) {

if (top == nullptr) {

cerr << "Стек пуст, невозможно выполнить операцию pop." << endl;

return;

}

Stack\* p = top;

char data = p->data;

top = top->prev;

delete p;

return data;

}

int main() {

setlocale(0, "");

system("chcp 1251");

system("cls");

cout << "кол-во элементов: ";

int n;

char a;

cin >> n;

cout << "Введите элементы стека: ";

Stack\* st = made\_stack(n);

print\_stack(st);

char b;

cout << "Введите ключ для удаления: ";

cin >> b;

int k = 0;

Stack\* st2 = nullptr;

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

char t = pop(st);

if (t != b) {

push(st2, t);

}

else {

k++;

}

}

n = n - k;

while (st2 != nullptr) {

char t = pop(st2);

push(st, t);

}

cout << "Стек после изменений: ";

print\_stack(st);

k = 0;

cout << "Введите кол-во элементов для добавления: ";

cin >> k;

int s = 0;

cout << "Введите номер элемента, перед которым добавляют элементы: ";

cin >> s;

for (int i = 0; i < n - s + 1; i++) {

char t = pop(st2);

push(st2, t);

}

cout << "Введите элементы: ";

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

cin >> a;

push(st, a);

}

for (int i = 0; i < n - s + 1; i++) {

char t = pop(st2);

push(st, t);

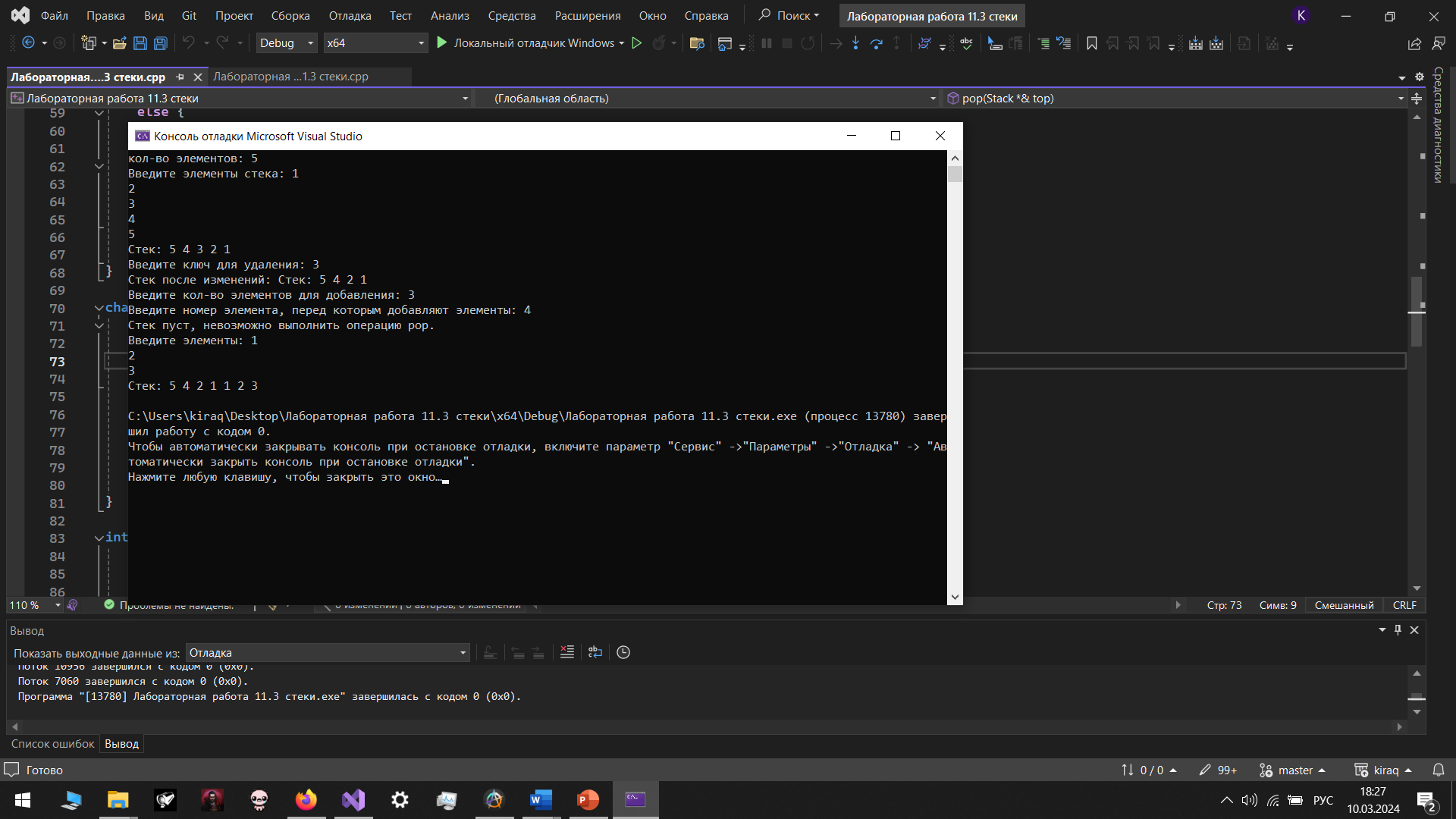
}

print\_stack(st);

return 0;

}

**Пример работы кода:**



**Вывод:**

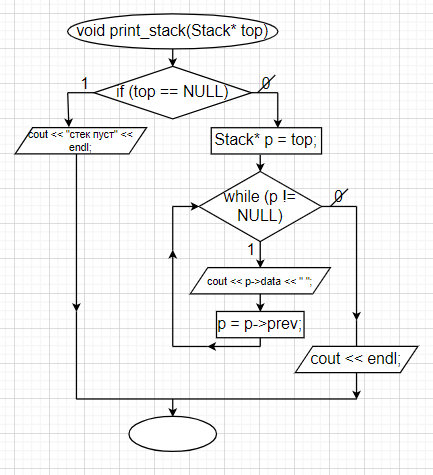
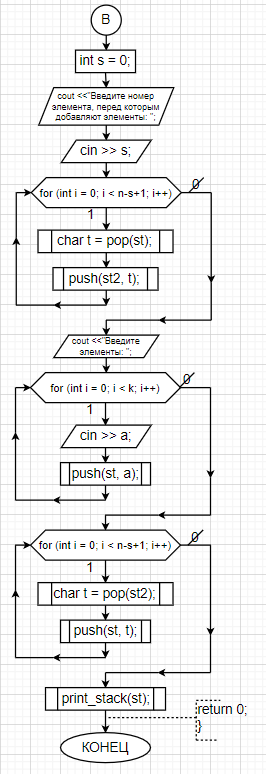
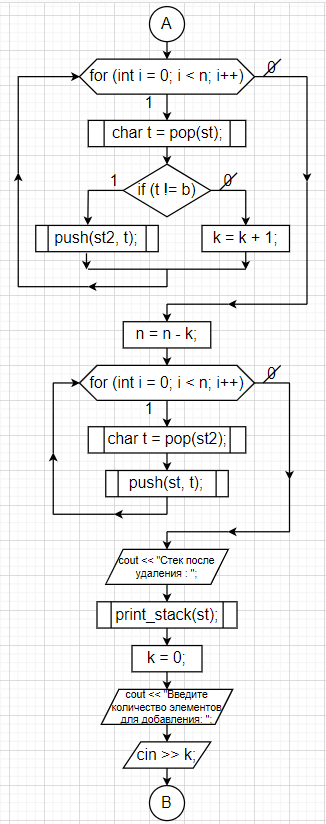
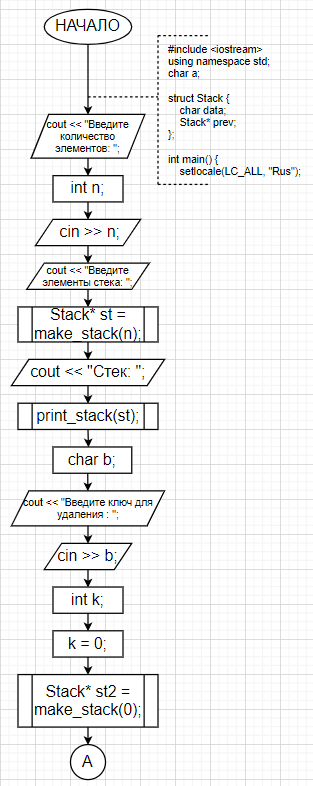
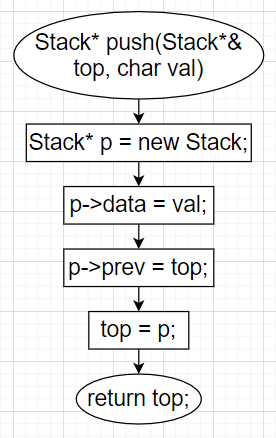
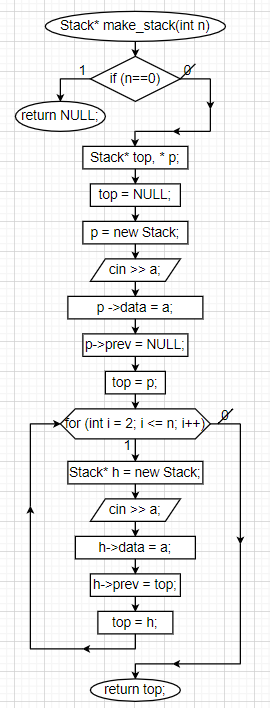
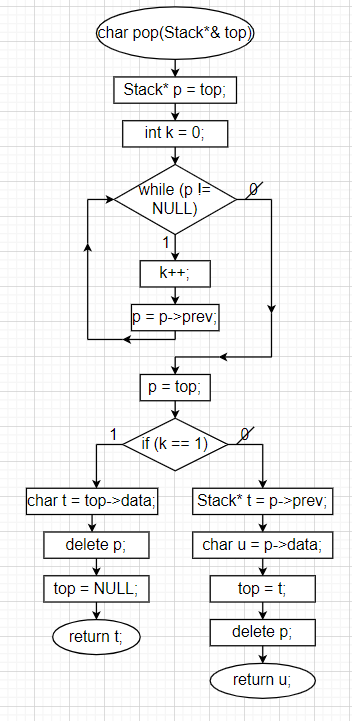
Код работает как надо

**Очередь**

**Анализ задачи:**

* Создаем структуру Stack
* Создаем методы для работы со списками(new\_queue, init\_queue, push, print\_queue, pop, delete\_key, insert, delete\_queue)
* Используя созданные методы, работаем со стеком по заданию

**Блок-схема:**

**Код на языке с++:**

#include <iostream>

using namespace std;

int n;

char key;

template <typename T>

struct Node

{

T data;

Node<T>\* next;

};

template <typename T>

struct Queue

{

int size;

Node<T>\* head;

Node<T>\* tail;

};

template <typename T>

void new\_queue(Queue<T>& q, int n)

{

T a;

cout << "Введите элемент: ";

cin >> a;

init\_queue(q, a);

for (int i = 2; i <= n; i++)

{

cout << "Введите элемент: ";

cin >> a;

push(q, a);

}

}

template <typename T>

void init\_queue(Queue<T>& q, const T& value)

{

Node<T>\* ptr = new Node<T>();

ptr->data = value;

q.head = ptr;

q.tail = ptr;

q.size = 1;

}

template <typename T>

void push(Queue<T>& q, const T& value)

{

Node<T>\* ptr = new Node<T>();

q.size++;

ptr->data = value;

ptr->next = nullptr;

q.tail->next = ptr;

q.tail = ptr;

}

template <typename T>

void print\_queue(Queue<T>& q)

{

Node<T>\* tmp = q.head;

cout << "Начало -> ";

while (tmp != nullptr)

{

cout << tmp->data << " -> ";

tmp = tmp->next;

}

cout << "Конец" << endl;

}

template <typename T>

void pop(Queue<T>& q)

{

Node<T>\* tmp = q.head;

q.head = q.head->next;

q.size--;

delete tmp;

}

template <typename T>

void delete\_key(Queue<T>& q, T k)

{

int i = 1;

while (i <=q.size)

{

if (q.head->data == k)

{

pop(q);

}

else

{

push(q, q.head->data);

pop(q);

i++;

}

}

}

template <typename T>

void insert(Queue<T>& q, int count, int number)

{

int i = 1;

while (i < number)

{

push(q, q.head->data);

pop(q);

i++;

}

T a;

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

{

cout << "ВВедите элемент для добавления: ";

cin >> a;

push(q, a);

}

for (i; i <+ q.size; i++)

{

push(q, q.head->data);

pop(q);

}

}

template <typename T>

void delete\_queue(Queue<T>& q)

{

while (q.head->next != nullptr)

{

pop(q);

}

Node<T>\* tmp = q.head;

q.head = nullptr;

q.size--;

delete tmp;

}

int main()

{

setlocale(0, "");

Queue<char> q;

do

{

cout << "Введите кол-во элементов в очереди: ";

cin >> n;

} while (n <= 0);

new\_queue(q, n);

cout << endl;

print\_queue(q);

cout << endl << "Введите ключ элемента для удаления: ";

cin >> key;

delete\_key(q, key);

cout << endl;

print\_queue(q);

int num;

do

{

cout << endl << "Введите номер элемента, перед которым нужно вставить элемент: ";

cin >> num;

} while (num <= 0 || num > q.size);

cout << endl << "Введите кол-во элементов для добавления: ";

int count;

cin >> count;

insert(q, count, num);

cout << endl;

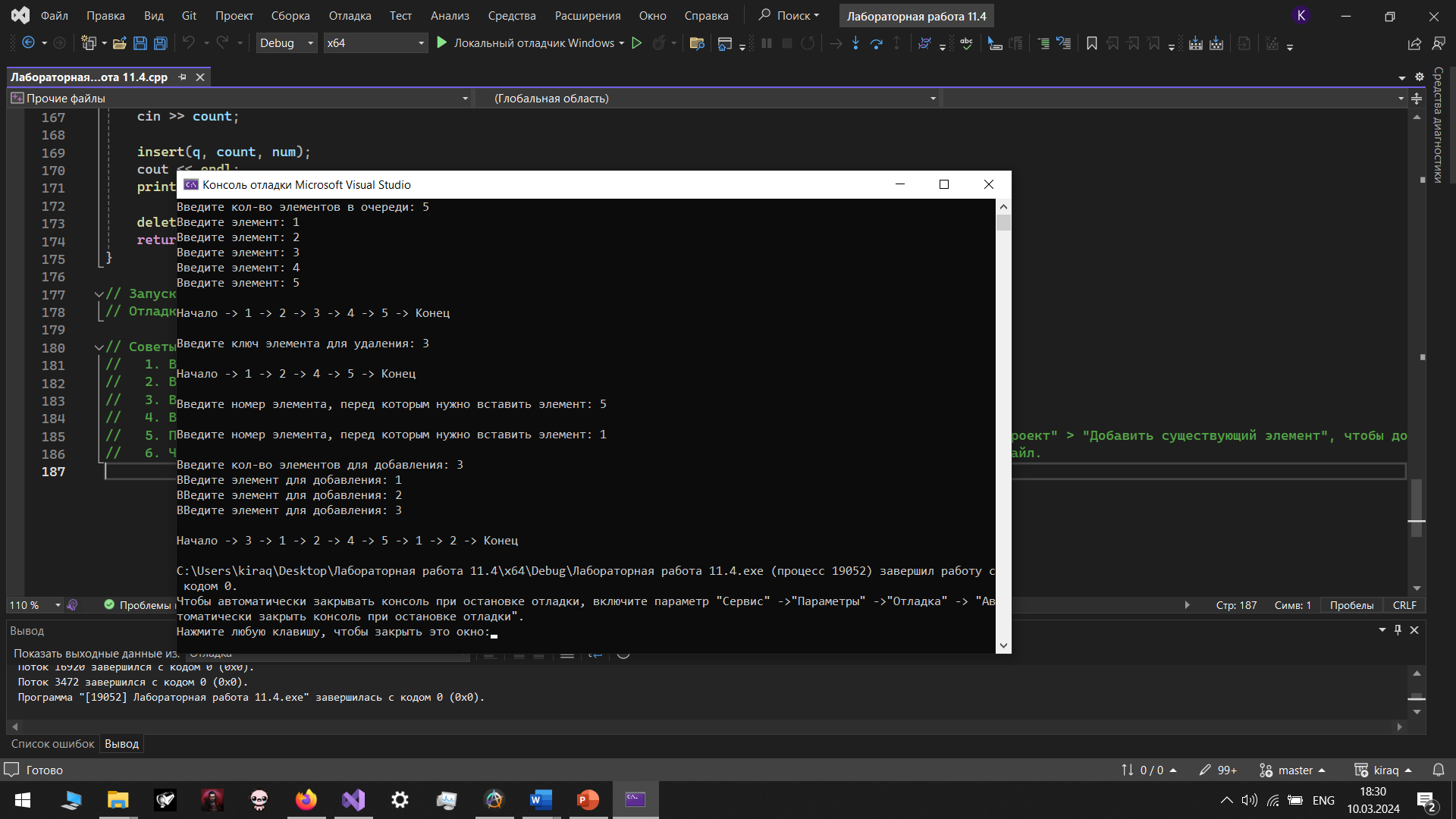
print\_queue(q);

delete\_queue(q);

return 0;

}

**Пример работы кода:**



**Вывод:**

Код работает как надо