Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

направление подготовки: 09.03.01 – «Информатика и вычислительная техника»

**ОТЧЁТ**

**Лабораторная работа №11**

**«Информационные динамические структуры»**

**Вариант 8**

Выполнила работу:

студентка гр. ИВТ-24-2б

Малая Алина Александровна

Проверил:

Доцент кафедры ИТАС

Полякова Ольга Андреевна

(оценка) (подпись)

(дата)

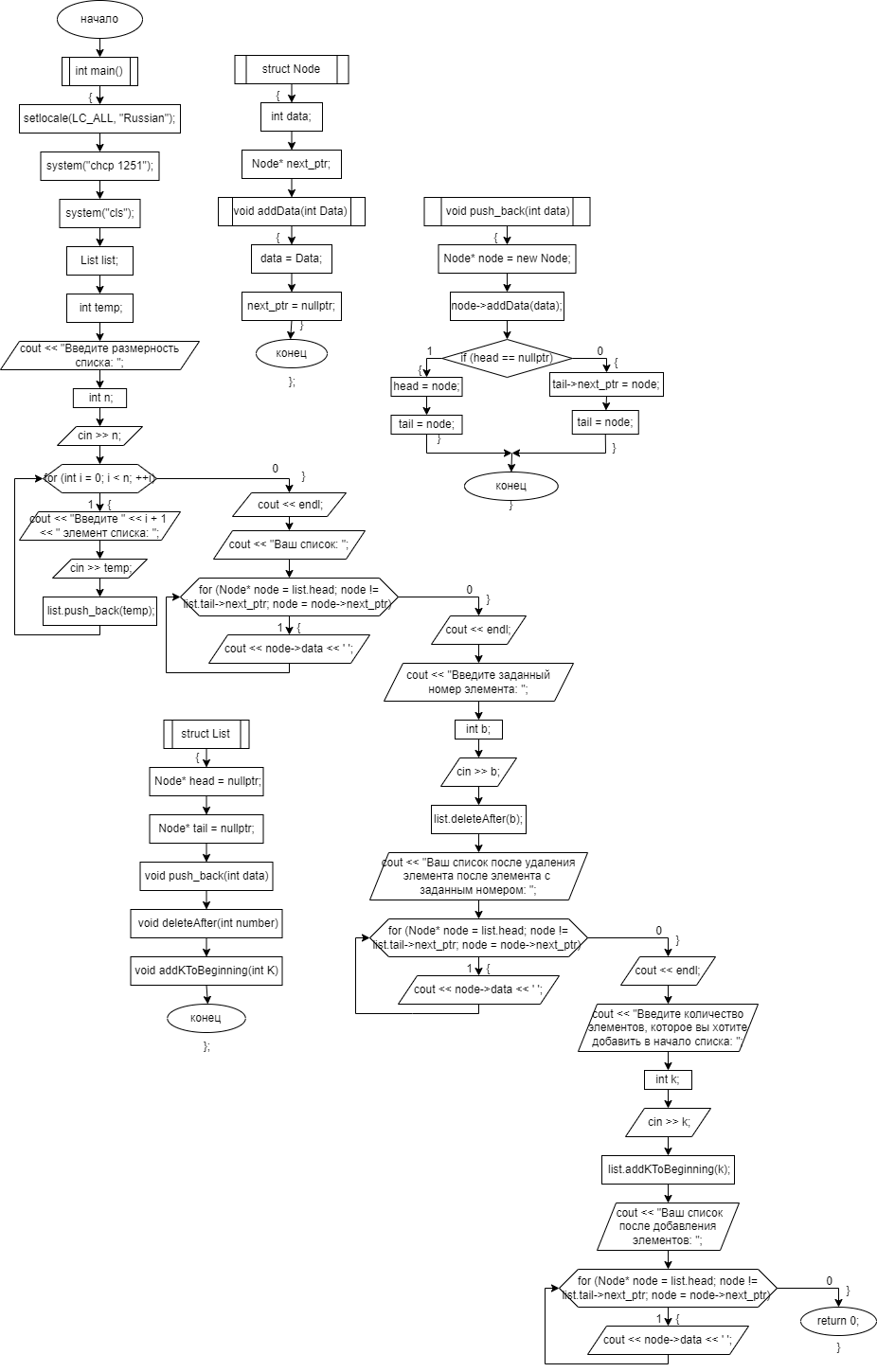
г. Пермь, 2025

**Постановка задачи**

Записи в линейном списке содержат ключевое поле типа int. Сформировать однонаправленный, двунаправленный список, стек и очередь. Удалить из них элемент после элемента с заданным номером, добавить K элементов в начало списка (однонаправленного и двунаправленного), стека и очереди.

**Однонаправленный список**

**Блок – схема**

****

**Код программы**

#include <iostream>

#include <clocale>

using namespace std;

struct Node // Cписок типа int

{

int data;

Node\* next\_ptr;

void addData(int Data)

{

data = Data;

next\_ptr = nullptr;

}

};

struct List

{

Node\* head = nullptr;

Node\* tail = nullptr;

void push\_back(int data) // Заполнение списка

{

Node\* node = new Node;

node->addData(data);

if (head == nullptr) {

head = node;

tail = node;

}

else

{

tail->next\_ptr = node;

tail = node;

}

}

void deleteAfter(int number) // Функция для удаления элемента после элемента с заданным номером

{

if (head == nullptr) return; // Проверяем если список пуст

Node\* current = head;

int count = 1;

while (current && count < number) // Поиск элемента с нужным индексом

{

current = current->next\_ptr;

count++;

}

if (!current || !current->next\_ptr) return; // Элемент не найден или нет следующего

Node\* toDelete = current->next\_ptr;

current->next\_ptr = toDelete->next\_ptr; // Перенаправляем указатель

// Обновляем tail, если удаляем последний элемент

if (toDelete == tail)

{

tail = current; // Tail становится элементом перед удаленным

}

}

void addKToBeginning(int K) // Функция для добавления K элементов в начало списка

{

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

{

int a;

cout << "Введите значение для нового элемента (" << i + 1 << "/" << K << "): ";

cin >> a;

Node\* newNode = new Node; // Создаем новый узел

newNode->addData(a); // Устанавливаем значение

newNode->next\_ptr = head; // Новый узел указывает на старую голову

head = newNode; // Обновляем голову списка

if (tail == nullptr) // Если список пуст

{

tail = newNode; // Обновляем tail, чтобы он указывал на новый узел, который является и головой, и хвостом

}

}

}

};

int main()

{

setlocale(LC\_ALL, "Russian");

system("chcp 1251");

system("cls");

List list;

int temp;

cout << "Введите размерность списка: ";

int n;

cin >> n;

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

{

cout << "Введите " << i + 1 << " элемент списка: ";

cin >> temp;

list.push\_back(temp);

}

cout << endl;

// Выводим наш список

cout << "Ваш список: ";

for (Node\* node = list.head; node != list.tail->next\_ptr; node = node->next\_ptr)

{

cout << node->data << ' ';

}

cout << endl;

cout << "Введите значение элемента после которого удалится элемент : ";

int b;

cin >> b;

list.deleteAfter(b);

cout << "Ваш список после удаления элемента: ";

for (Node\* node = list.head; node != list.tail->next\_ptr; node = node->next\_ptr)

{

cout << node->data << ' ';

}

cout << endl;

cout << "Введите количество элементов, которое вы хотите добавить в начало списка: ";

int k;

cin >> k;

list.addKToBeginning(k);

cout << "Ваш список после добавления элементов: ";

for (Node\* node = list.head; node != list.tail->next\_ptr; node = node->next\_ptr)

{

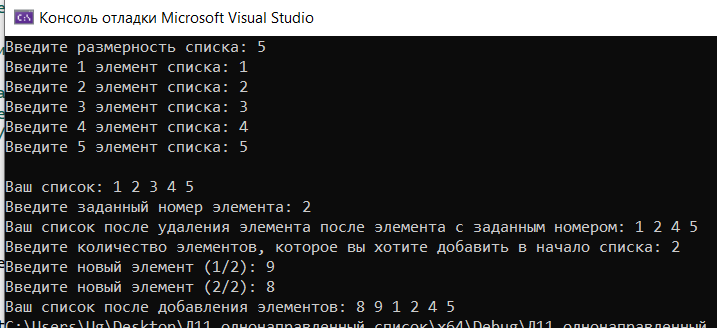
cout << node->data << ' ';

}

return 0;

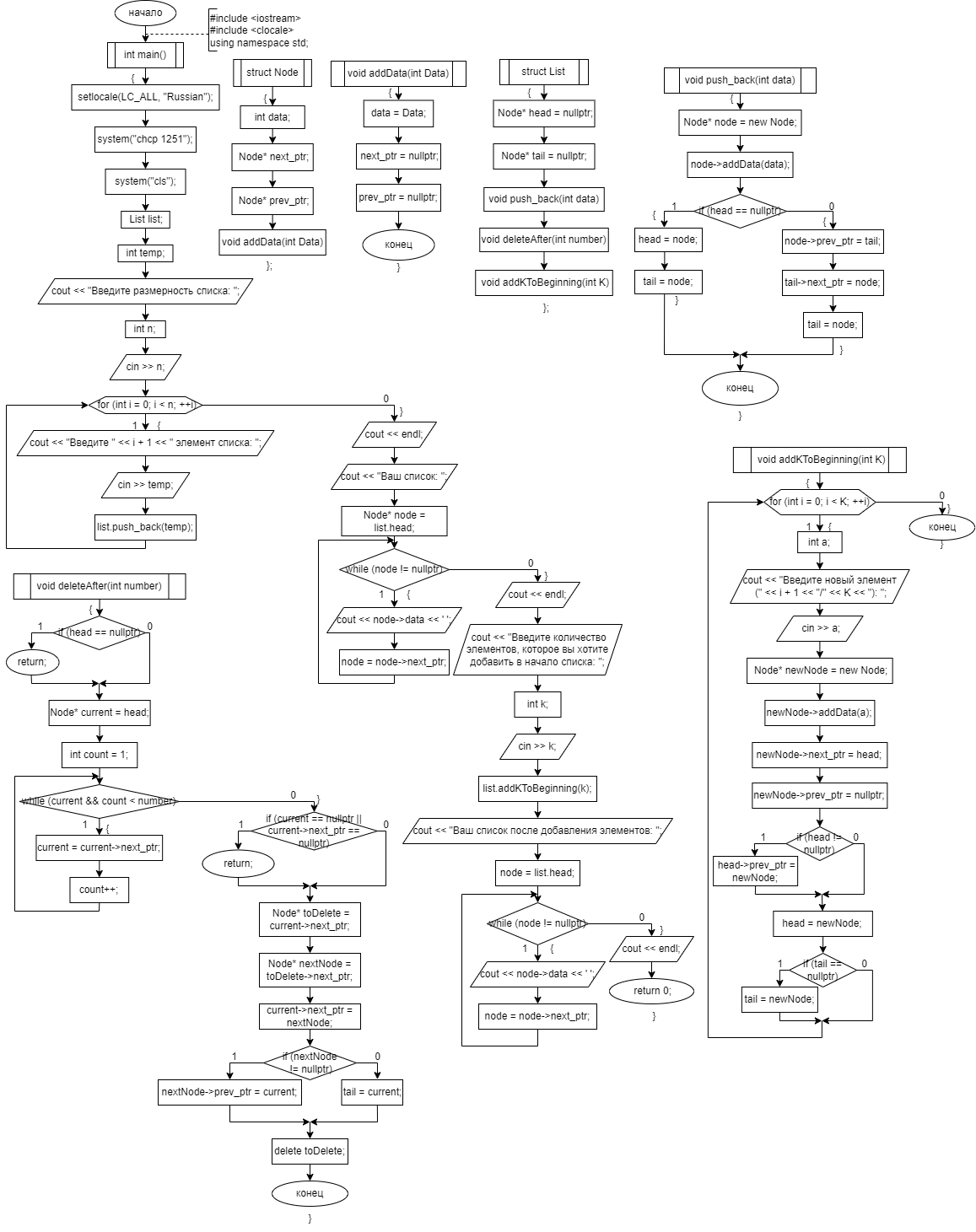
}

**Вывод**

****

**Двунаправленный список**

**Блок – схема**

****

**Код программы**

#include <iostream>

#include <clocale>

using namespace std;

struct Node // Объявляем структуру Node, которая представляет собой узел списка.

{

int data;

Node\* next\_ptr;

Node\* prev\_ptr;

void addData(int Data) // Объявляем функцию-член addData, которая принимает целое число Data и устанавливает его в качестве значения для data.

{

data = Data; // Присваиваем значение Data члену data структуры Node.

next\_ptr = nullptr; // Устанавливаем next\_ptr в nullptr (нулевой указатель), так как при создании нового узла он пока не указывает ни на что.

prev\_ptr = nullptr; // Устанавливаем prev\_ptr в nullptr (нулевой указатель), так как при создании нового узла он пока не указывает ни на что.

}

};

struct List // Объявляем структуру List, которая представляет собой список.

{

Node\* head = nullptr; // Объявляем член структуры List с именем head типа Node\*, который будет хранить указатель на первый узел в списке. Инициализируем его в nullptr.

Node\* tail = nullptr; // Объявляем член структуры List с именем tail типа Node\*, который будет хранить указатель на последний узел в списке. Инициализируем его в nullptr.

void push\_back(int data) // Объявляем функцию-член push\_back, которая принимает целое число data и добавляет новый узел с этим значением в конец списка.

{

Node\* node = new Node; // Создаем новый узел с помощью оператора new.

node->addData(data); // Устанавливаем данные для нового узла.

if (head == nullptr) // Если список пуст (head равен nullptr):

{

head = node; // Устанавливаем head на новый узел.

tail = node; // Устанавливаем tail на новый узел (так как это единственный элемент).

}

else // Если список не пуст

{

node->prev\_ptr = tail; // Устанавливаем указатель prev\_ptr нового узла на текущий хвостовой узел (tail).

tail->next\_ptr = node; // Устанавливаем next\_ptr текущего последнего узла (tail) на новый узел.

tail = node; // Устанавливаем tail на новый узел (так как он теперь последний).

}

}

void deleteAfter(int number) // Объявляем функцию-член deleteAfter, которая принимает целое число number и удаляет узел, следующий после узла с указанным номером.

{

if (head == nullptr) return; // Если список пуст (head равен nullptr), выходим из функции.

Node\* current = head; // Устанавливаем указатель current на начало списка (head).

int count = 1; // Инициализируем счетчик count значением 1 (так как отсчет начинается с 1, а не с 0).

while (current && count < number) // Пока current не равен nullptr (не достигли конца списка) и count меньше number:

{

current = current->next\_ptr; // Перемещаем current на следующий узел.

count++; // Увеличиваем счетчик count.

}

if (current == nullptr || current->next\_ptr == nullptr) return; // Если current равен nullptr (узел с заданным номером не найден) или current->next\_ptr равен nullptr (после узла с заданным номером нет узла), выходим из функции.

Node\* toDelete = current->next\_ptr; // Устанавливаем указатель toDelete на узел, который нужно удалить (узел, следующий после current).

Node\* nextNode = toDelete->next\_ptr; // Сохраняем указатель на узел, идущий после удаляемого.

current->next\_ptr = nextNode; // Перенаправляем указатель next\_ptr текущего узла (current) на узел, следующий после удаляемого узла (toDelete), исключая удаляемый узел из списка.

if (nextNode != nullptr) // Если удаляем не последний элемент

{

nextNode->prev\_ptr = current; // Устанавливаем указатель prev\_ptr следующего узла (nextNode) на текущий узел (current).

}

else // Если удаляем последний элемент, обновляем tail

{

tail = current; // Устанавливаем tail на текущий узел (так как он теперь последний).

}

delete toDelete; // Освобождаем память, занимаемую удаленным узлом.

}

void addKToBeginning(int K) // Объявляем функцию-член addKToBeginning, которая принимает целое число K и добавляет K новых узлов в начало списка.

{

for (int i = 0; i < K; ++i) // Цикл, который выполняется K раз.

{

int a; // Объявляем переменную a типа int для хранения значения нового узла.

cout << "Введите новый элемент (" << i + 1 << "/" << K << "): "; // Выводим сообщение с просьбой ввести значение для нового элемента.

cin >> a; // Считываем значение с клавиатуры и сохраняем его в переменной a.

Node\* newNode = new Node; // Создаем новый узел с помощью оператора new.

newNode->addData(a); // Устанавливаем данные для нового узла.

newNode->next\_ptr = head; // Устанавливаем next\_ptr нового узла на текущую голову списка (head).

newNode->prev\_ptr = nullptr; // Устанавливаем prev\_ptr нового узла на nullptr, так как он будет новым началом списка.

if (head != nullptr) // Если список не пуст

{

head->prev\_ptr = newNode; // Устанавливаем prev\_ptr старой головы списка (head) на новый узел.

}

head = newNode; // Устанавливаем head на новый узел (так как он теперь первый).

if (tail == nullptr) // Если список был пуст

{

tail = newNode; // Устанавливаем tail на новый узел (так как он теперь единственный элемент).

}

}

}

};

int main()

{

setlocale(LC\_ALL, "Russian");

system("chcp 1251");

system("cls");

List list;

int temp;

cout << "Введите размерность списка: ";

int n;

cin >> n;

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

{

cout << "Введите " << i + 1 << " элемент списка: ";

cin >> temp;

list.push\_back(temp);

}

cout << endl;

cout << "Ваш список: ";

Node\* node = list.head;

while (node != nullptr) // Пока не достигли конца списка

{

cout << node->data << ' '; // Выводим значение данных текущего узла.

node = node->next\_ptr; // Перемещаем node на следующий узел.

}

cout << endl;

cout << "Введите значение элемента после которого удалится элемент: ";

int b;

cin >> b;

list.deleteAfter(b);

cout << "Ваш список после удаления элемента: ";

node = list.head;

while (node != nullptr)

{

cout << node->data << ' ';

node = node->next\_ptr;

}

cout << endl;

cout << "Введите количество элементов, которое вы хотите добавить в начало списка: ";

int k;

cin >> k;

list.addKToBeginning(k);

cout << "Ваш список после добавления элементов: ";

node = list.head;

while (node != nullptr)

{

cout << node->data << ' ';

node = node->next\_ptr;

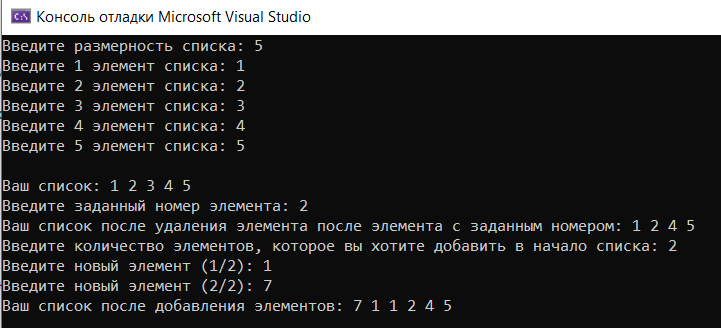
}

cout << endl;

return 0;

}

**Вывод**

****

**Стек**

**Блок – схема**

**Код программы**

#include <iostream>

#include <clocale>

using namespace std;

struct Node

{

int data;

Node\* next\_ptr;

};

struct Stack

{

Node\* head = nullptr;

int size;

void push\_back(int Data) // функция добавления элемента

{

Node\* node = new Node;

node->data = Data;

node->next\_ptr = head;

head = node;

}

void deleteAfter(int Data) // функция удаления элемента после заданного

{

Node\* node = head;

while (node != nullptr)

{

if (node->data == Data)

{

if (node->next\_ptr != nullptr)

{

Node\* deleteElement = node->next\_ptr;

node->next\_ptr = deleteElement->next\_ptr;

delete deleteElement;

size--;

return;

}

else

{

cout << "После элемента " << Data << " нет других элементов" << endl;

return;

}

}

node = node->next\_ptr;

}

cout << "Элемент " << Data << " не найден в стеке" << endl;

}

void addKElements(int K) // функция добавления К элментов в начало стека

{

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

{

int data;

cout << "Введите значение для нового элемента (" << i + 1 << "/" << K << "): ";

cin >> data;

push\_back(data);

}

}

void Show() // функция вывода стека на экран

{

Node\* temp = head;

while (temp != 0)

{

cout << temp->data << " ";

temp = temp->next\_ptr;

}

cout << endl;

}

};

int main()

{

setlocale(LC\_ALL, "Russian");

system("chcp 1251");

system("cls");

Stack MyStack;

int temp;

cout << "Введите размерность стека: ";

int n;

cin >> n;

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

{

cout << "Введите " << i + 1 << " элемент стека: ";

cin >> temp;

MyStack.push\_back(temp);

}

cout << "Ваш стек: ";

MyStack.Show();

cout << "Введите значение элмента после которого удалится элемент: ";

int b;

cin >> b;

MyStack.deleteAfter(b);

cout << "Ваш стек после удаления элемента: ";

MyStack.Show();

cout << "Введите количество элементов, которое вы хотите добавить в начало стека: ";

int k;

cin >> k;

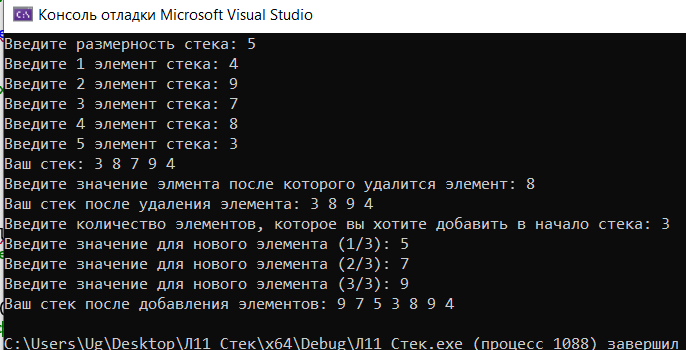
MyStack.addKElements(k);

cout << "Ваш стек после добавления элементов: ";

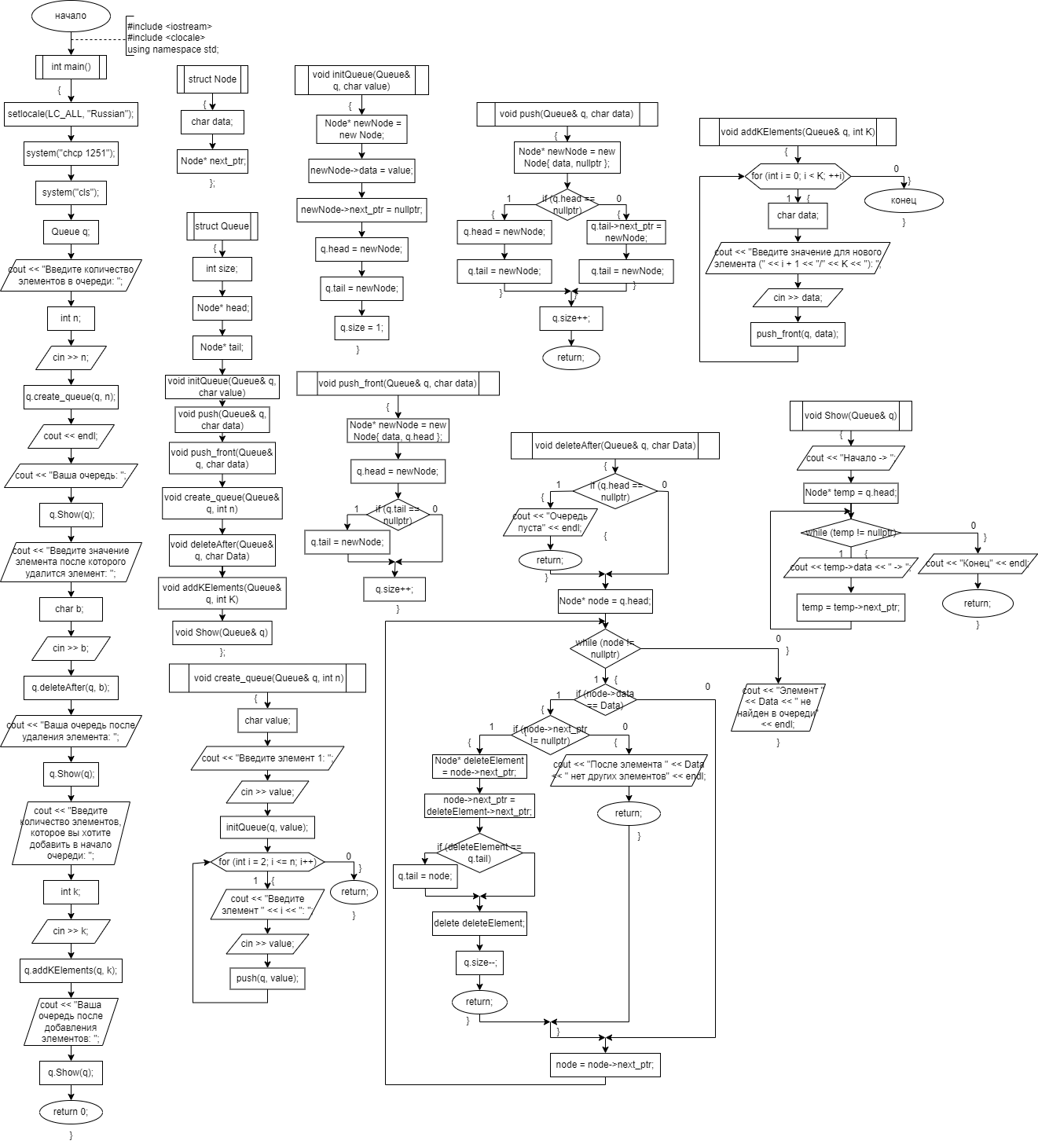
MyStack.Show();

return 0;

}

**Вывод**

**Очередь**

**Блок – схема**

**Код программы**

#include <iostream>

#include <clocale>

using namespace std;

struct Node

{

char data;

Node\* next\_ptr;

};

struct Queue

{

int size;

Node\* head;

Node\* tail;

void initQueue(Queue& q, char value) // функция инициализации очереди

{

Node\* newNode = new Node;

newNode->data = value;

newNode->next\_ptr = nullptr;

q.head = newNode;

q.tail = newNode;

q.size = 1;

}

void push(Queue& q, char data) // функция добавления элемента в очередь (в конец)

{

Node\* newNode = new Node{ data, nullptr };

if (q.head == nullptr)

{

q.head = newNode;

q.tail = newNode;

}

else

{

q.tail->next\_ptr = newNode;

q.tail = newNode;

}

q.size++;

return;

}

void push\_front(Queue& q, char data) // функция добавления элемента в начало очереди

{

Node\* newNode = new Node{ data, q.head };

q.head = newNode;

if (q.tail == nullptr)

{

q.tail = newNode;

}

q.size++;

}

void create\_queue(Queue& q, int n)

{

char value;

cout << "Введите элемент 1: ";

cin >> value;

initQueue(q, value);

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

{

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

cin >> value;

push(q, value);

}

return;

}

void deleteAfter(Queue& q, char Data) // функция удаления элемента после заданного

{

if (q.head == nullptr)

{

cout << "Очередь пуста" << endl;

return;

}

Node\* node = q.head;

while (node != nullptr)

{

if (node->data == Data)

{

if (node->next\_ptr != nullptr)

{

Node\* deleteElement = node->next\_ptr;

node->next\_ptr = deleteElement->next\_ptr;

if (deleteElement == q.tail)

{

q.tail = node;

}

delete deleteElement;

q.size--;

return;

}

else

{

cout << "После элемента " << Data << " нет других элементов" << endl;

return;

}

}

node = node->next\_ptr;

}

cout << "Элемент " << Data << " не найден в очереди" << endl;

}

void addKElements(Queue& q, int K) // функция добавления К элементов в начало очереди

{

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

{

char data;

cout << "Введите значение для нового элемента (" << i + 1 << "/" << K << "): ";

cin >> data;

push\_front(q, data);

}

}

void Show(Queue& q) // функция вывода очереди на экран

{

cout << "Начало -> ";

Node\* temp = q.head;

while (temp != nullptr)

{

cout << temp->data << " -> ";

temp = temp->next\_ptr;

}

cout << "Конец" << endl;

return;

}

};

int main()

{

setlocale(LC\_ALL, "Russian");

system("chcp 1251");

system("cls");

Queue q;

cout << "Введите количество элементов в очереди: ";

int n;

cin >> n;

q.create\_queue(q, n);

cout << endl;

cout << "Ваша очередь: ";

q.Show(q);

cout << "Введите значение элмента после которого удалится элемент: ";

char b;

cin >> b;

q.deleteAfter(q, b);

cout << "Ваша очередь после удаления элемента: ";

q.Show(q);

cout << "Введите количество элементов, которое вы хотите добавить в начало очереди: ";

int k;

cin >> k;

q.addKElements(q, k);

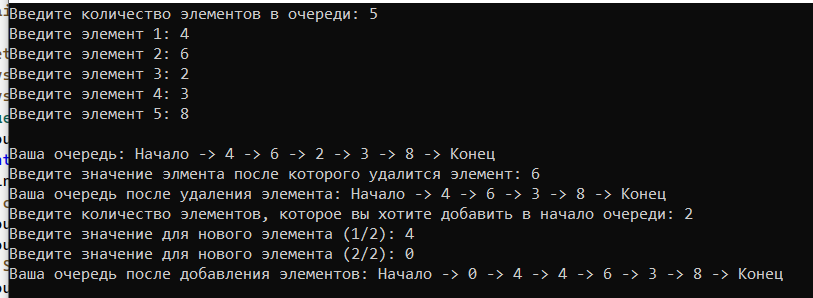
cout << "Ваша очередь после добавления элементов: ";

q.Show(q);

return 0;

}

**Вывод**

****

**GitHub**

<https://github.com/amalayaa>