Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет»**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы» направление подготовки: 09.03.04 – «Программная инженерия»

**Лабораторная работа №11.**

**«Информационные динамические структуры.**

**Двухсвязные списки»**

Выполнил студент гр. РИС-24-1б

Чижов Денис Николаевич

Проверил:

Доц. Каф. ИТАС 

Ольга Андреевна Полякова

(оценка) (подпись)

(дата)

г. Пермь, 2025

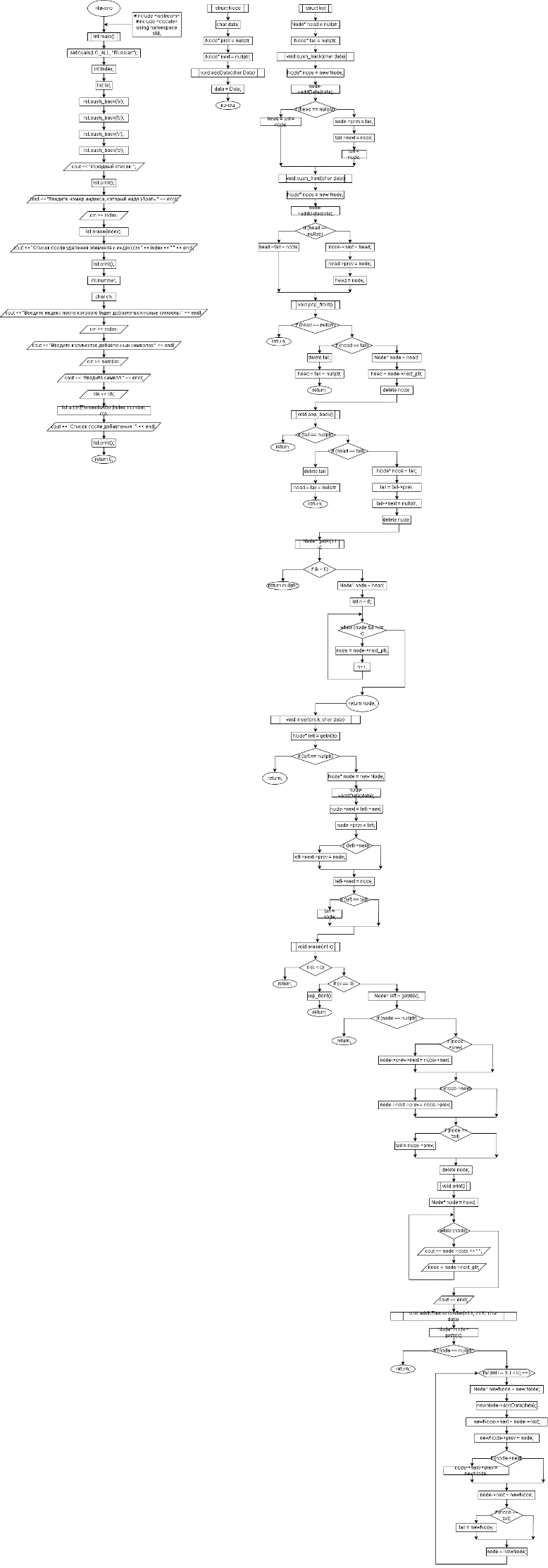
**Задача:**

23.Записи в линейном списке содержат ключевое поле типа \*char(строка символов). Сформировать двухсвязный список. Удалить элемент с заданным ключом. Добавить K элементов после элемента с заданным ключом.

**Анализ и визуализация:**

Мы формируем двухсвязный список, в котором записаны наши элементы типа char и затем удаляем элемент с заданным индексом и добавляем K элементов после элемента с выбранным индексом.

**Блок-схема:**



**Код:**

#include <iostream>

#include <clocale>

using namespace std;

struct Node {

char data;

Node\* prev = nullptr; // Указатель на предыдущий элемент

Node\* next = nullptr; // Указатель на следующий элемент

void addData(char Data) {

data = Data;

}

};

struct list {

Node\* head = nullptr;

Node\* tail = nullptr;

// Добавление элемента в конец списка

void push\_back(char data) {

Node\* node = new Node;

node->addData(data);

if (head == nullptr) {

head = tail = node; // Если список пуст, head и tail указывают на новый элемент

}

else {

node->prev = tail; // Новый элемент указывает на текущий tail

tail->next = node; // Текущий tail указывает на новый элемент

tail = node; // Новый элемент становится tail

}

}

// Добавление элемента в начало списка

void push\_front(char data) {

Node\* node = new Node;

node->addData(data);

if (head == nullptr) {

head = tail = node; // Если список пуст, head и tail указывают на новый элемент

}

else {

node->next = head; // Новый элемент указывает на текущий head

head->prev = node; // Текущий head указывает на новый элемент

head = node; // Новый элемент становится head

}

}

// Удаление элемента из начала списка

void pop\_front() {

if (head == nullptr) return; // Если список пуст, ничего не делаем

if (head == tail) { // Если в списке только один элемент

delete head;

head = tail = nullptr;

return;

}

Node\* node = head; // Запоминаем текущий head

head = head->next; // Перемещаем head на следующий элемент

head->prev = nullptr; // Новый head не имеет предыдущего элемента

delete node; // Удаляем старый head

}

// Удаление элемента из конца списка

void pop\_back() {

if (tail == nullptr) return; // Если список пуст, ничего не делаем

if (head == tail) { // Если в списке только один элемент

delete tail;

head = tail = nullptr;

return;

}

Node\* node = tail; // Запоминаем текущий tail

tail = tail->prev; // Перемещаем tail на предыдущий элемент

tail->next = nullptr; // Новый tail не имеет следующего элемента

delete node; // Удаляем старый tail

}

// Получение элемента по индексу

Node\* getAt(int k) {

if (k < 0) return nullptr;

Node\* node = head;

int n = 0;

while (node && n != k) {

node = node->next;

n++;

}

return node;

}

// Вставка элемента после элемента с индексом k

void insert(int k, char data) {

Node\* left = getAt(k);

if (left == nullptr) return;

Node\* node = new Node;

node->addData(data);

node->next = left->next; // Новый элемент указывает на следующий элемент left

node->prev = left; // Новый элемент указывает на left

if (left->next) {

left->next->prev = node; // Следующий элемент left указывает на новый элемент

}

left->next = node; // Left указывает на новый элемент

if (left == tail) tail = node; // Если left был tail, новый элемент становится tail

}

// Удаление элемента с индексом k

void erase(int k) {

if (k < 0) return;

if (k == 0) {

pop\_front();

return;

}

Node\* node = getAt(k);

if (node == nullptr) return;

if (node->prev) {

node->prev->next = node->next; // Предыдущий элемент указывает на следующий

}

if (node->next) {

node->next->prev = node->prev; // Следующий элемент указывает на предыдущий

}

if (node == tail) tail = node->prev; // Если удаляемый элемент был tail, обновляем tail

delete node;

}

// Вывод списка

void print() {

Node\* node = head;

while (node) {

cout << node->data << " ";

node = node->next;

}

cout << endl;

}

// Добавление K элементов после элемента с индексом k

void addKElementsAfter(int k, int K, char data) {

Node\* node = getAt(k);

if (node == nullptr) return;

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

Node\* newNode = new Node;

newNode->addData(data);

newNode->next = node->next; // Новый элемент указывает на следующий элемент node

newNode->prev = node; // Новый элемент указывает на node

if (node->next) {

node->next->prev = newNode; // Следующий элемент node указывает на новый элемент

}

node->next = newNode; // Node указывает на новый элемент

if (node == tail) tail = newNode; // Если node был tail, новый элемент становится tail

node = newNode; // Перемещаем node на новый элемент для следующей итерации

}

}

};

int main() {

setlocale(LC\_ALL, "Russian");

int index;

list lst;

lst.push\_back('a');

lst.push\_back('b');

lst.push\_back('c');

lst.push\_back('d');

cout << "Исходный список: ";

lst.print();

// Удаляем элемент с индексом 2

cout << "Введите номер индекса, который надо убрать:" << endl;

cin >> index;

lst.erase(index);

cout << "Список после удаления элемента с индексом " << index << ":" << endl;

lst.print();

// Добавляем K элементов после элемента с индексом k

int number;

char ch;

cout << "Введите индекс после которого будет добавляться новые символы:" << endl;

cin >> index;

cout << "Введите количество добавленных символов:" << endl;

cin >> number;

cout << "Введите символ:" << endl;

cin >> ch;

lst.addKElementsAfter(index, number, ch);

cout << "Список после добавления: " << endl;

lst.print();

return 0;

}

**Вывод:**

Программа работает. Задача решена.