

Федеральное государственное автономное образовательное учреждение
высшего образования «Пермский национальный исследовательский
политехнический университет»

Лабораторная работа №11
«Информационные динамические структуры»
Вариант №25

Выполнила:

студент первого курса

ЭТФ группы РИС-23-36

Акбашева Софья Руслановна

Проверила:

Доцент кафедры ИТАС О. А. Полякова

Пермь 2024

Информационные динамические структуры

Цель: Знакомство с динамическими информационными структурами на примере одно- и двунаправленных списков.

Постановка задачи: написать программу, в которой создаются динамические структуры и выполнить их обработку в соответствии со своим вариантом.

Задача: Записи в линейном списке содержат ключевое поле типа `*char`(строка символов). Сформировать двунаправленный список. Удалить элемент с заданным ключом. Добавить K элементов перед элементом с заданным номером.

Необходимо разработать следующие функции:

1. Создание списка.
2. Добавление элемента в список (в соответствии со своим вариантом).
3. Удаление элемента из списка (в соответствии со своим вариантом).
4. Печать списка.
5. Запись списка в файл.
6. Уничтожение списка.
7. Восстановление списка из файла.

Однонаправленные списки

Анализ задачи

- 1) Структура узел содержит два поля: `data` - данные узла, и `ptr_to_next_node` - указатель на следующий узел в списке.
- 2) Структура список содержит два поля: `head_node` - указатель на головной узел списка, и `tail_node` - указатель на хвостовой узел списка.
- 3) Функция `pushBack` добавляет новый элемент в конец списка. Если список пустой, новый узел становится головным и хвостовым узлом. Если список не пустой, новый узел связывается с хвостовым узлом и становится новым хвостовым узлом.

4) Функция `add_elements` добавляет новый элемент в список перед указанным элементом. Если вводить новый элемент перед первым элементом, новый узел становится головным узлом. В противном случае, новый узел связывается со следующим узлом после элемента, перед которым добавляется новый элемент.

5) Функция `print_list` выводит текущий список с помощью итерационного цикла. Если список пустой, выводится сообщение "Список пустой!".

6) Функция `pop_element` удаляет указанный элемент из списка. Если удалять первый элемент, новый головной элемент становится следующим за удаляемым. Если удалять НЕ первый элемент, узлы перед и после удаляемого элемента связываются напрямую.

7) Функция `del_all_list` удаляет все элементы из списка. Головной узел списка заменяется следующим узлом, и так до тех пор, пока все узлы не будут удалены. Память освобождается.

8) Для работы с файлами необходимо `#include <fstream>`.
Подключение файловых потоков ввода и вывода. Функция `writing_to_a_file` с помощью итерационного цикла записывает в файл содержимое списка.
Функция `recoveru` восстанавливает список из файла с помощью итерационного цикла.

Блок схема

начало

```
struct Node {  
    char data; //данные  
    Node* ptr_to_next_node = nullptr;  
};
```

```
struct List {  
    Node* head_node = nullptr;  
    Node* teil_node = nullptr;  
};
```

```
void add_elements(List& list, int befor_number) {  
    char element;  
    cout << "Введите новый элемент ";  
    cin >> element;  
  
    Node* new_node = new Node;  
    new_node->data = element;  
  
    if (befor_number == 1) {  
        new_node->ptr_to_next_node = list.head_node;  
        list.head_node = new_node;  
    }  
    else {  
        Node* pointer_node = list.head_node;  
        for (int i = 0; i < befor_number - 2; i++) {  
            pointer_node = pointer_node->ptr_to_next_node;  
        }  
        new_node->ptr_to_next_node = pointer_node->ptr_to_next_node;  
        pointer_node->ptr_to_next_node = new_node;  
    }  
}
```

```
void pop_element(List& list, char num_del_el) {  
    if (list.head_node != nullptr) {  
        Node* pointer_node = list.head_node;  
        while (pointer_node != nullptr) {  
            if (pointer_node->data == num_del_el) {  
                if (pointer_node == list.head_node) {  
                    Node* new_Head = list.head_node->ptr_to_next_node;  
                    delete list.head_node;  
                    list.head_node = new_Head;  
                    pointer_node = list.head_node;  
                }  
                else {  
                    Node* before_deletion = list.head_node;  
                    while (before_deletion->ptr_to_next_node != pointer_node) {  
                        before_deletion = before_deletion->ptr_to_next_node;  
                    }  
                    before_deletion->ptr_to_next_node = pointer_node->ptr_to_next_node;  
                    pointer_node = before_deletion;  
                }  
            }  
            else {  
                pointer_node = pointer_node->ptr_to_next_node;  
            }  
        }  
    }  
}
```

```
void recovery(List& list, ifstream& file) {  
    string all_str;  
    getline(file, all_str);  
    Node* new_node = new Node;  
    new_node->data = all_str[0];  
    list.head_node = new_node;  
    list.teil_node = new_node;  
    while (getline(file, all_str)) {  
        pushBack(list, all_str[0]);  
    }  
}
```

```
#include <iostream>  
#include <fstream>  
#include <string>  
using namespace std;
```

```
void pushBack(List& list, const int& data) {  
    Node* new_node = new Node;  
    new_node->data = data;  
    if (list.head_node == nullptr) {  
        list.head_node = new_node;  
        list.teil_node = new_node;  
    }  
    else { // если список не пустой  
        list.teil_node->ptr_to_next_node = new_node;  
        list.teil_node = new_node;  
    }  
}
```

```
void print_list(List& list) {  
    cout << endl << "Текущий список:" << endl;  
    Node* current_node = list.head_node;  
    if (current_node != nullptr) {  
        while (current_node != nullptr) {  
            cout << current_node->data << " ";  
            current_node = current_node->ptr_to_next_node;  
        }  
    }  
    else {  
        cout << "Список пустой!";  
    }  
    cout << endl << endl;  
    delete current_node;  
}
```

```
void del_all_list(List& list) {  
    Node* ptr_node = list.head_node;  
    if (ptr_node != nullptr) {  
        while (ptr_node != nullptr) {  
            Node* new_Head = list.head_node->ptr_to_next_node;  
            delete list.head_node;  
            list.head_node = new_Head;  
            ptr_node = list.head_node;  
        }  
    }  
}
```

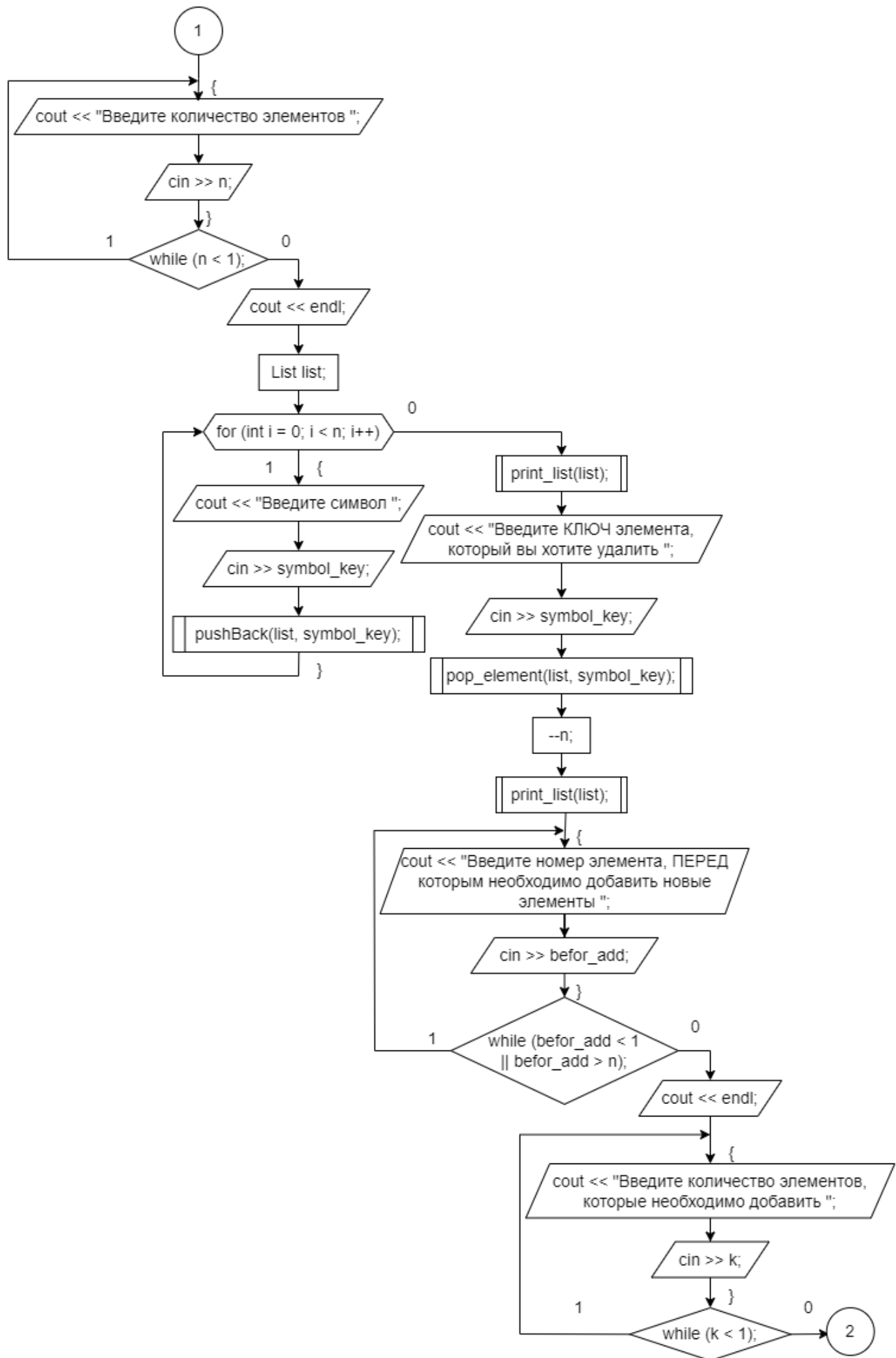
```
void writing_to_a_file(List& list, ofstream& file) {  
    if (list.head_node != nullptr) {  
        Node* pointer_q = list.head_node;  
        while (pointer_q != nullptr) {  
            file << pointer_q->data << endl;  
            pointer_q = pointer_q->ptr_to_next_node;  
        }  
    }  
}
```

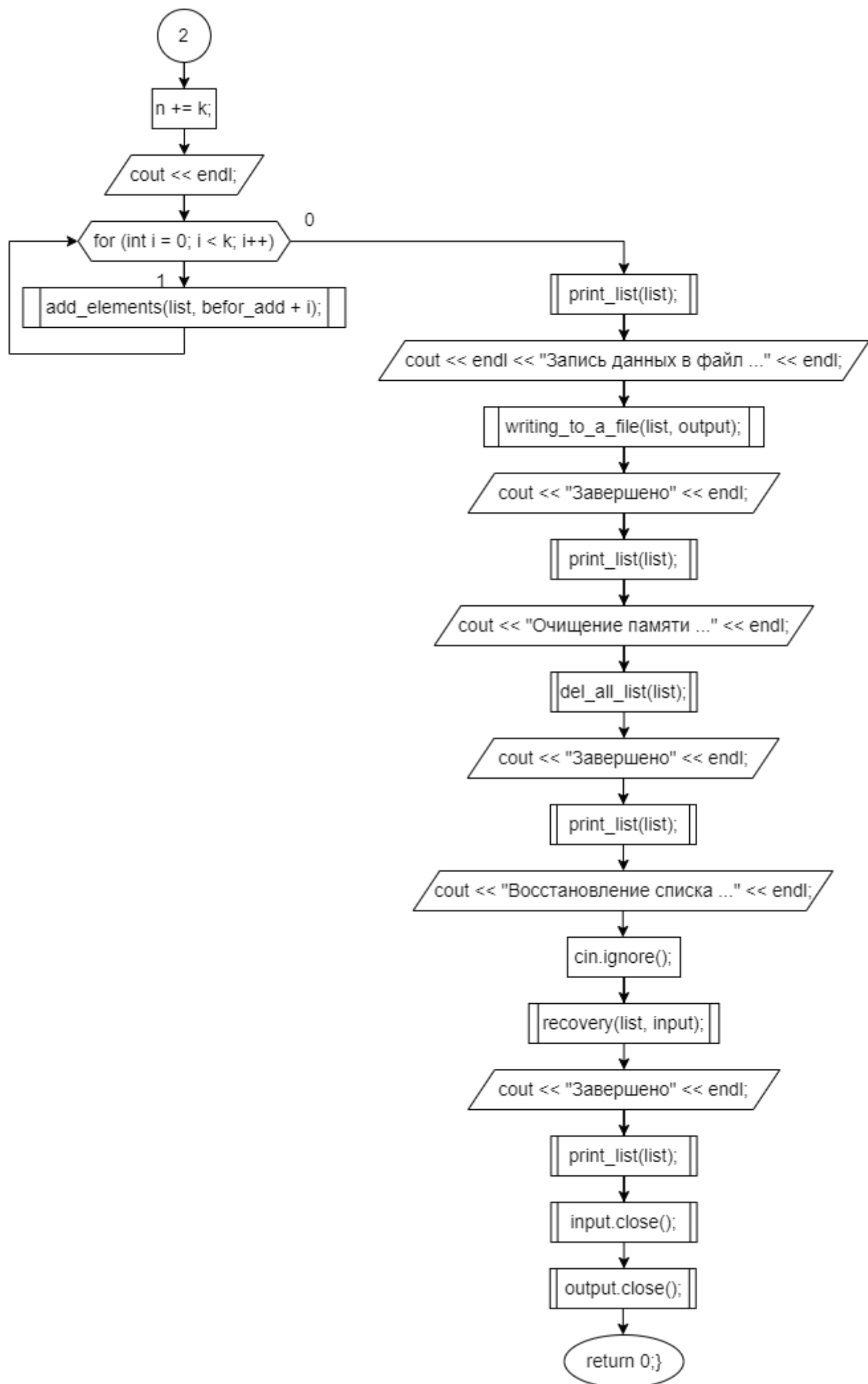
```
int main() {  
    setlocale(LC_ALL, "Russian");  
    system("chcp 1251");  
    system("cls");
```

```
    ifstream input("F11.txt");  
    ofstream output("F11.txt");
```

```
    int n, k, befor_add;  
    char symbol_key;
```

1





Код программы

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

struct Node {
    char data; //данные
    Node* ptr_to_next_node = nullptr; //указатель на следующий элемент
};

struct List {
    Node* head_node = nullptr;
    Node* teill_node = nullptr;
};

void pushBack(List& list, const int& data); //добавляю новый элемент в конец
списка
void add_elements(List& list, int befor_number); //добавляю новый элемент
void print_list(List& list); //вывод текущего списка
void pop_element(List& list, char num_del_el); //удаление элемента
void del_all_list(List& list); //освобождение памяти
void writing_to_a_file(List& list, ofstream& file); //запись данных в файл
void recovery(List& list, ifstream& file); //восстановление

int main() {
    setlocale(LC_ALL, "Russian"); //локализация
    system("chcp 1251");
    system("cls");

    ifstream input("F11.txt"); //входной файловый поток
    ofstream output("F11.txt"); //выходной файловый поток

    int n, k, befor_add;
    char symbol_key;

    do {
        cout << "Введите количество элементов ";
        cin >> n;
    } while (n < 1);
    cout << endl;

    List list; //инициализирую список

    for (int i = 0; i < n; i++) {
        cout << "Введите символ ";
        cin >> symbol_key;
        pushBack(list, symbol_key);
    }

    print_list(list); //вывожу текущий массив

    cout << "Введите КЛЮЧ элемента, который вы хотите удалить ";
    cin >> symbol_key;

    pop_element(list, symbol_key); //удаляю элемент
    --n; //изменяю длину списка
    print_list(list); //вывожу текущий массив

    do {
        cout << "Введите номер элемента, ПЕРЕД которым необходимо добавить новые
элементы ";
        cin >> befor_add; //НОМЕР элемента
    } while (befor_add < 1 || befor_add > n);
    cout << endl;
```

```

do {
    cout << "Введите количество элементов, которые необходимо добавить ";
    cin >> k; //количество элементов, которые надо добавить
} while (k < 1);
n += k; //изменяю длину списка
cout << endl;

for (int i = 0; i < k; i++) { //добавляю новые элементы
    add_elements(list, befor_add + i);
}
print_list(list); //вывожу текущий массив

cout << endl << "Запись данных в файл ..." << endl;
writing_to_a_file(list, output);
cout << "Завершено" << endl;

print_list(list); //вывожу текущий массив
cout << "Очищение памяти ..." << endl;
del_all_list(list);

cout << "Завершено" << endl;
print_list(list); //вывожу текущий массив

cout << "Восстановление списка ..." << endl;
cin.ignore();
recovery(list, input);
cout << "Завершено" << endl;

print_list(list); //вывожу текущую очередь

input.close(); //закрываю файл
output.close(); //закрываю файл

return 0;
}

void pushBack(List& list, const int& data) { //добавляю новый элемент в конец
списка
    Node* new_node = new Node; //создаю новый динамический узел
    new_node->data = data; //присваиваю полю узла данные
    if (list.head_node == nullptr) { //если список пустой
        list.head_node = new_node; //новый узел - головной узел списка
        list.teil_node = new_node; //новый узел - хвостовой узел списка
    }
    else { // если список не пустой
        list.teil_node->ptr_to_next_node = new_node; //связываю новый узел с
хвостовым
        list.teil_node = new_node; //меняю хвостовой узел на новый
    }
}

void add_elements(List& list, int befor_number) { //добавляю новый элемент
char element;
cout << "Введите новый элемент ";
cin >> element;

Node* new_node = new Node; //создаю новый узел
new_node->data = element; //присваиваю значение данных

if (befor_number == 1) { //если вводить новый элемент перед первым элементом
    new_node->ptr_to_next_node = list.head_node;
    list.head_node = new_node;
}
else {
    Node* pointer_node = list.head_node;
    for (int i = 0; i < befor_number - 2; i++) { //иду до элемента перед
которым над добавить новый

```



```

        pointer_node = pointer_node->ptr_to_next_node; //элемент перед
необходимым элементом
    }
    new_node->ptr_to_next_node = pointer_node->ptr_to_next_node; //связываю
новый узел со следующим
    pointer_node->ptr_to_next_node = new_node; //связываю новый узел с
предыдущим

    }
}

void print_list(List& list) { //вывод текущего списка
    cout << endl << "Текущий список:" << endl;
    Node* current_node = list.head_node;
    if (current_node != nullptr) {
        while (current_node != nullptr) { //пока не дойду до последнего элемента
            cout << current_node->data << ' '; //вывод данных текущего узла
            current_node = current_node->ptr_to_next_node; //переход к следующему
узлу
        }
    }
    else {
        cout << "Список пустой!";
    }
    cout << endl << endl;
    delete current_node; //очищаю память
}

void pop_element(List& list, char num_del_el) { //удаление элемента
    if (list.head_node != nullptr) { //если список НЕ пустой
        Node* pointer_node = list.head_node;
        while (pointer_node != nullptr) { //пока не дойду до конца
            if (pointer_node->data == num_del_el) {
                if (pointer_node == list.head_node) { //если надо удалить первый
элемент
                    Node* new_Head = list.head_node->ptr_to_next_node;
                    delete list.head_node; //удаляю текущий головной элемент
                    list.head_node = new_Head; //присваиваю головному элементу
новый элемент
                    pointer_node = list.head_node;
                }
                else { //если удалять НЕ первый элемент
                    Node* before_deletion = list.head_node;
                    while (before_deletion->ptr_to_next_node != pointer_node) {
                        before_deletion = before_deletion->ptr_to_next_node;
                    }
                    before_deletion->ptr_to_next_node = pointer_node->
ptr_to_next_node; //связываю узлы
                    pointer_node = before_deletion;
                }
            }
            else {
                pointer_node = pointer_node->ptr_to_next_node; //перехожу на
следующий элемент
            }
        }
    }
}

void del_all_list(List& list) { //освобождение памяти
    Node* ptr_node = list.head_node;
    if (ptr_node != nullptr) {
        while (ptr_node != nullptr) {
            Node* new_Head = list.head_node->ptr_to_next_node;
            delete list.head_node; //удаляю текущий головной элемент
            list.head_node = new_Head; //присваиваю головному элементу новый
элемент
        }
    }
}

```

```

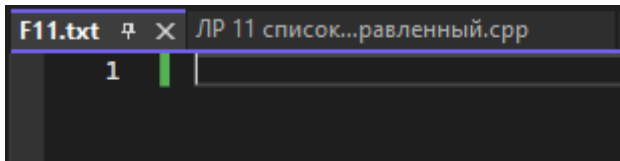
        ptr_node = list.head_node;
    }
}

void writing_to_a_file(List& list, ofstream& file) { //запись данных в файл
    if (list.head_node != nullptr) { //если список не пустой
        Node* pointer_q = list.head_node; //указатель на первый элемент
        while (pointer_q != nullptr) { //пока не дойду до конца
            file << pointer_q->data << endl;
            pointer_q = pointer_q->ptr_to_next_node; //перехожу на следующий узел
        }
    }
}

void recovery(List& list, ifstream& file) { //восстановление
    string all_str;
    getline(file, all_str); //считываю строку
    Node* new_node = new Node; //создаю новый динамический узел
    new_node->data = all_str[0]; //присваиваю полю узла данные
    list.head_node = new_node; //новый узел - головной узел списка
    list.tail_node = new_node; //новый узел - хвостовой узел списка
    while (getline(file, all_str)) { //пока не пройду весь файл
        pushBack(list, all_str[0]); //добавляю
    }
}

```

Результат работы программы



```

Введите количество элементов 6
Введите символ q
Введите символ H
Введите символ f
Введите символ t
Введите символ t
Введите символ M
Текущий список:
q H f t t M

Введите КЛЮЧ элемента, который вы хотите удалить t
Текущий список:
q H f M

Введите номер элемента, ПЕРЕД которым необходимо добавить новые элементы 2
Введите количество элементов, которые необходимо добавить 3
Введите новый элемент 1
Введите новый элемент 2
Введите новый элемент 3
Текущий список:
q 1 2 3 H f M

Запись данных в файл ...
Завершено
Текущий список:
q 1 2 3 H f M

Очищение памяти ...
Завершено
Текущий список:
Список пустой!

Восстановление списка ...
Завершено
Текущий список:
q 1 2 3 H f M

```

F11.txt		ЛР 11 список...равленный.cpp
1		q
2		1
3		2
4		3
5		H
6		f
7		M

Двунаправленные списки

Анализ задачи

1) Структура узел содержит данные и два указателя на предыдущий и следующий узлы.

2) Структура список содержит указатели на головной и хвостовой узлы списка.

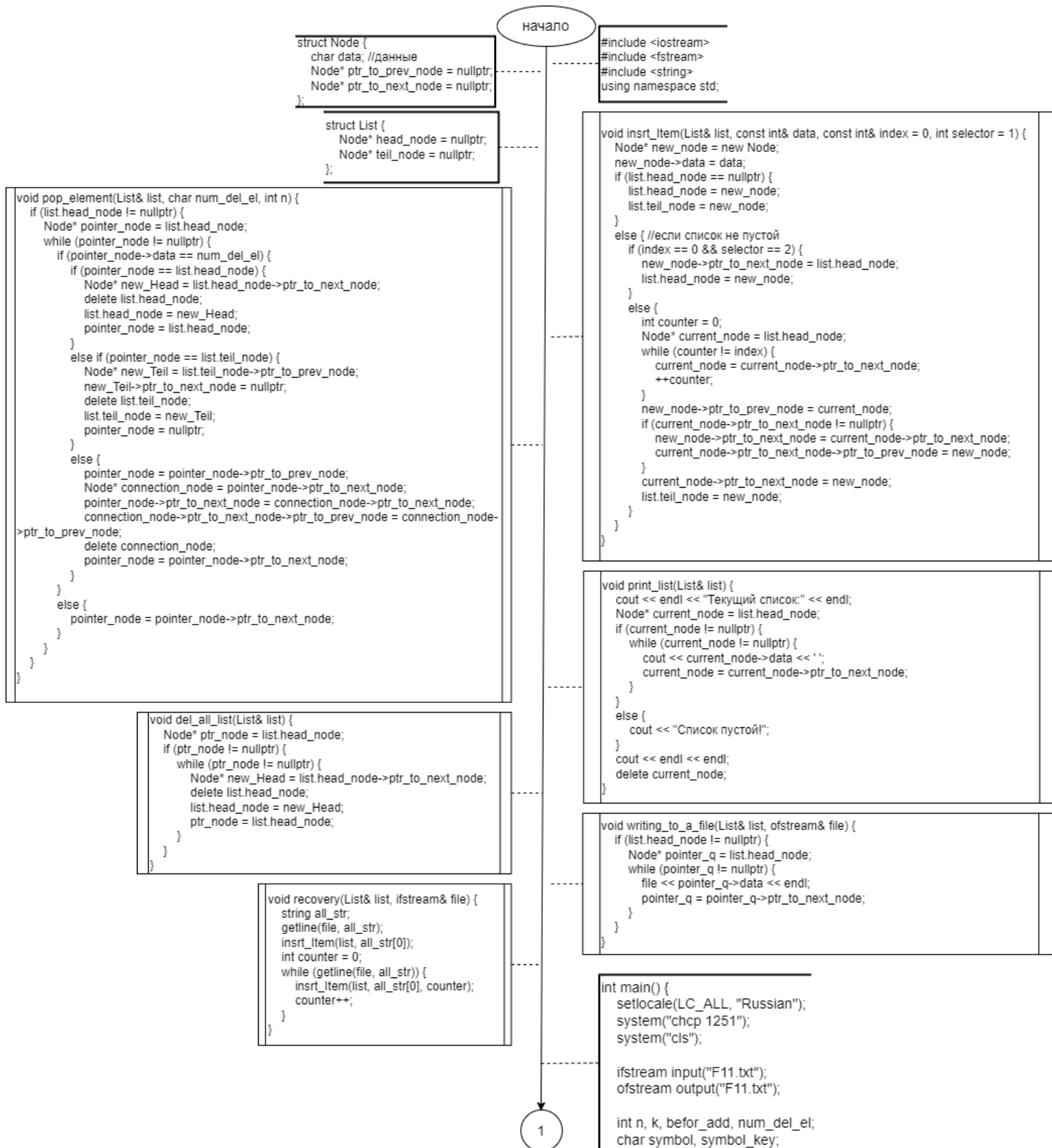
3) Функция `insrt_Item` добавляет новый элемент в конец списка. Если список пуст, новый узел становится головным и хвостовым узлом. Если список не пуст, новый узел связывается с головным узлом или вставляется в нужное место в списке.

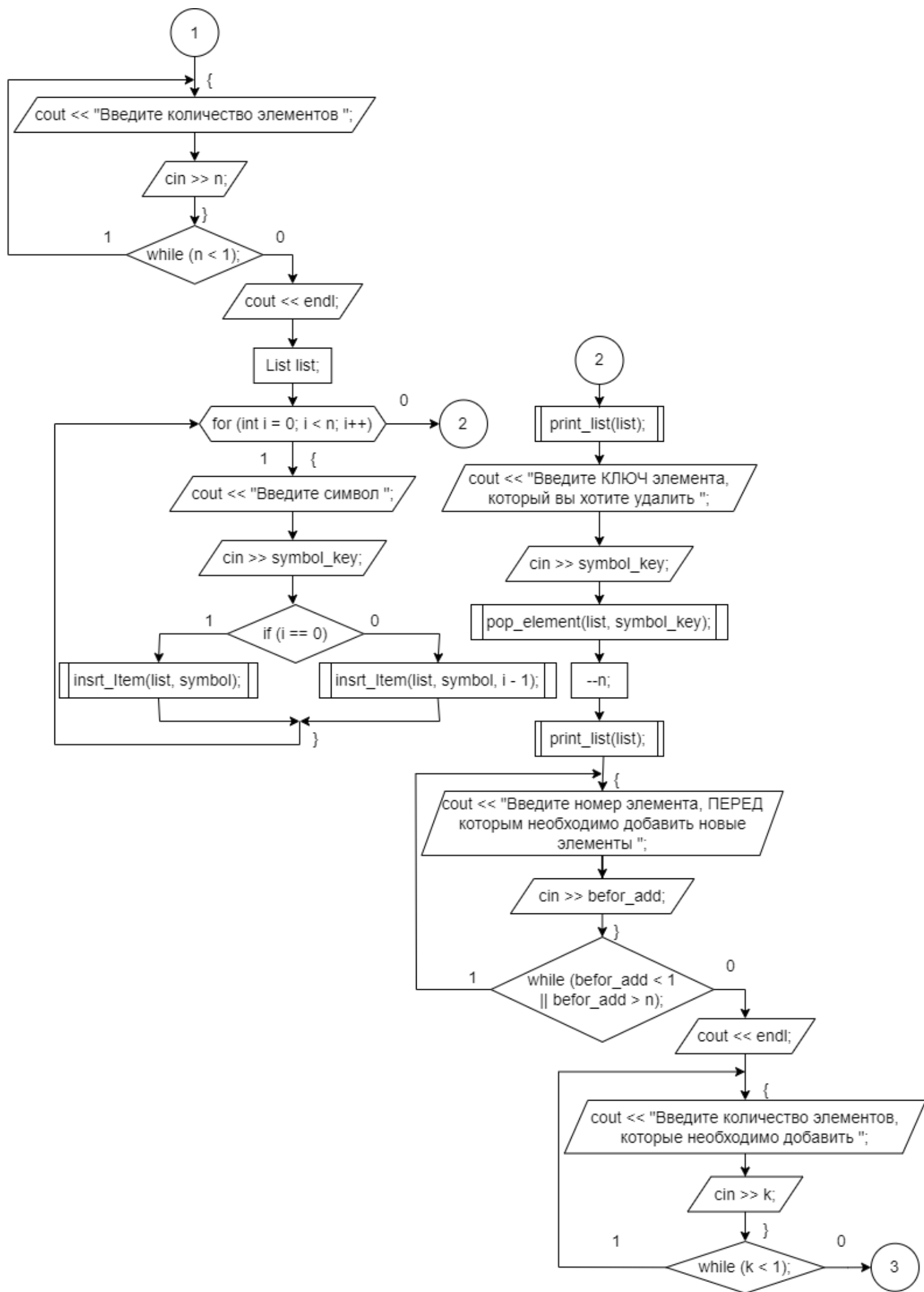
4) Функция `pop_element` удаляет элемент из списка. Если удаляемый элемент является головным или хвостовым узлом, код удаляет его и обновляет соответствующие указатели. Если удаляемый элемент находится в середине списка, код удаляет его и обновляет связи между узлами.

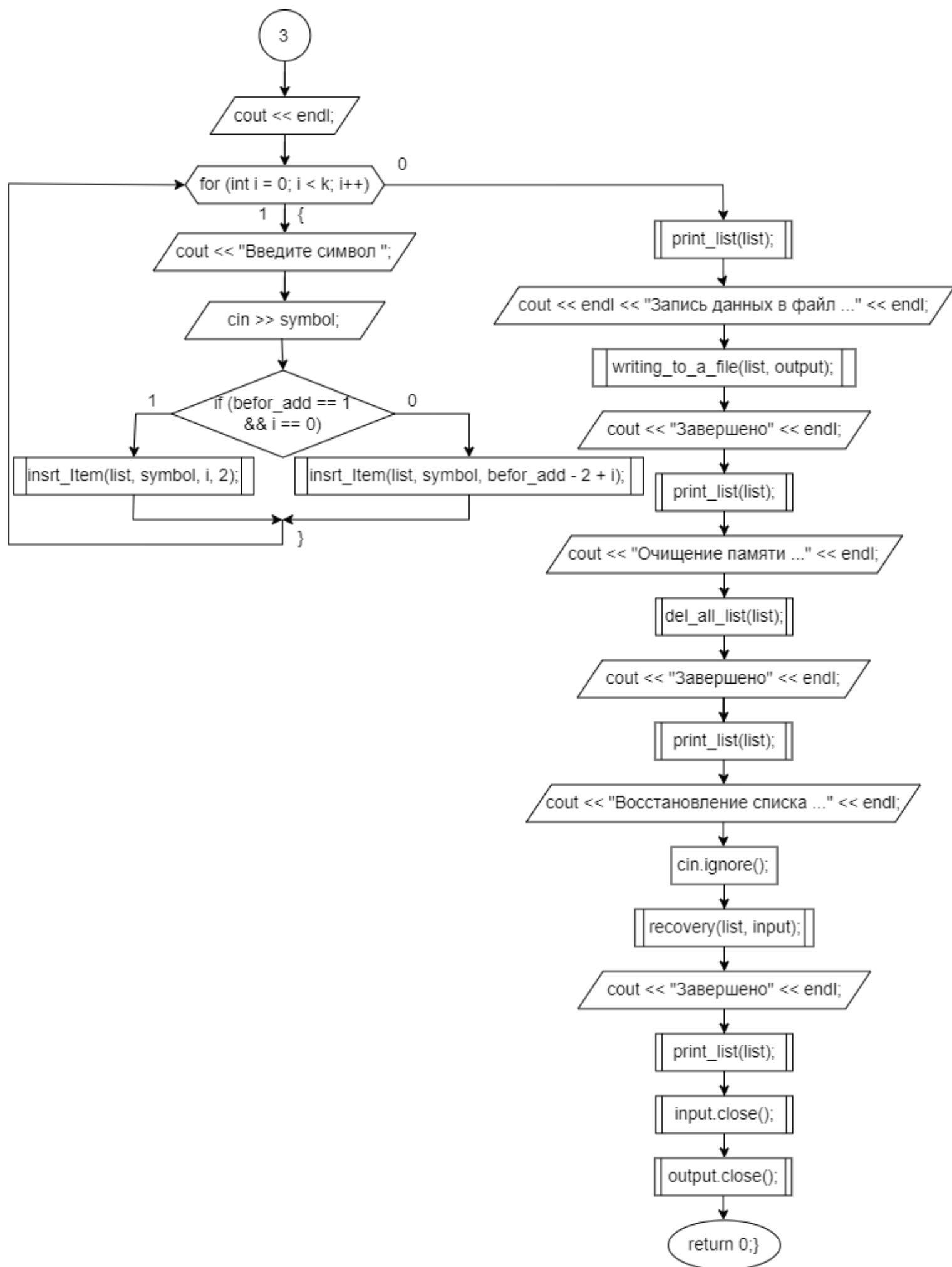
5) Функция `del_all_list` удаляет все элементы из списка. Головной узел списка заменяется следующим узлом, и так до тех пор, пока все узлы не будут удалены. Память освобождается.

6) Для работы с файлами необходимо `#include <fstream>`.
Подключение файловых потоков ввода и вывода. Функция `writing_to_a_file` с помощью итерационного цикла записывает в файл содержимое списка.
Функция `recoveru` восстанавливает список из файла с помощью итерационного цикла.

Блок схема







Код программы

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

struct Node {
    char data; //данные
    Node* ptr_to_prev_node = nullptr; //указатель на предыдущий элемент
    Node* ptr_to_next_node = nullptr; //указатель на следующий элемент
};

struct List {
    Node* head_node = nullptr;
    Node* teil_node = nullptr;
};

void insrt_Item(List& list, const int& data, const int& index = 0, int selector = 1); //добавляю новый элемент в конец списка
void pop_element(List& list, char num_del_el, int n); //удаление элемента
void print_list(List& list); //вывод текущего списка
void del_all_list(List& list); //очистение списка
void writing_to_a_file(List& list, ofstream& file); //запись данных в файл
void recovery(List& list, ifstream& file); //восстановление

int main() {
    setlocale(LC_ALL, "Russian"); //локализация
    system("chcp 1251");
    system("cls");

    ifstream input("F11.txt"); //входной файловый поток
    ofstream output("F11.txt"); //выходной файловый поток

    int n, k, befor_add;
    char symbol, symbol_key;

    do {
        cout << "Введите количество элементов ";
        cin >> n; //количество элементов в списке
    } while (n < 1);
    cout << endl;

    List list; //инициализирую список

    for (int i = 0; i < n; i++) {
        cout << "Введите символ ";
        cin >> symbol;
        if (i == 0) insrt_Item(list, symbol); //добавляю новый элемент
        else insrt_Item(list, symbol, i - 1);
    }

    print_list(list); //вывожу текущий массив

    cout << "Введите КЛЮЧ элемента, который вы хотите удалить ";
    cin >> symbol_key;

    pop_element(list, symbol_key, n); //удаляю элемент
    cout << "Выполнено удаление элемента(-ов) с заданным ключом" << endl;

    --n; //изменяю длину списка
    print_list(list); //вывожу текущий массив

    do {
        cout << "Введите номер элемента, ПЕРЕД которым необходимо добавить новые элементы ";
        cin >> befor_add; //НОМЕР элемента
```



```

} while (befor_add < 1 || befor_add > n);
cout << endl;

do {
    cout << "Введите количество элементов, которые необходимо добавить ";
    cin >> k; //количество элементов, которые надо добавить
} while (k < 1);
cout << endl;

for (int i = 0; i < k; i++) {
    cout << "Введите символ ";
    cin >> symbol;
    if (befor_add == 1 && i == 0) {
        insrt_Item(list, symbol, i, 2);
    }
    else insrt_Item(list, symbol, befor_add - 2 + i);
}

print_list(list); //вывожу текущий массив

cout << "Запись данных в файл ..." << endl;
writing_to_a_file(list, output);
cout << "Завершено" << endl << endl;

cout << "Очищение памяти ..." << endl;
del_all_list(list);
cout << "Завершено" << endl;
print_list(list); //вывожу текущий массив

cout << "Восстановление списка ..." << endl;
cin.ignore();
recovery(list, input);
cout << "Завершено" << endl;

print_list(list); //вывожу текущую очередь

input.close(); //закрываю файл
output.close(); //закрываю файл

return 0;
}

void insrt_Item(List& list, const int& data, const int& index, int selector) {
//добавляю новый элемент в конец списка
Node* new_node = new Node; //создаю новый динамический узел
new_node->data = data; //присваиваю полю узла данные

if (list.head_node == nullptr) { //если список пустой
    list.head_node = new_node; //новый узел - головной узел списка
    list.teil_node = new_node; //новый узел - хвостовой узел списка
}
else { //если список не пустой
    if (index == 0 && selector == 2) {
        new_node->ptr_to_next_node = list.head_node; //связываю новый узел с
        ГОЛОВНЫМ
        list.head_node = new_node; //новый головной узел
    }
    else {
        int counter = 0;
        Node* current_node = list.head_node;
        while (counter != index) { //иду ДО элемента, который надо удалять
            current_node = current_node->ptr_to_next_node;
            ++counter;
        }
        new_node->ptr_to_prev_node = current_node;
        if (current_node->ptr_to_next_node != nullptr) {

```

```

        new_node->ptr_to_next_node = current_node->ptr_to_next_node;
//связываю узлы
        current_node->ptr_to_next_node->ptr_to_prev_node = new_node;
    }
    current_node->ptr_to_next_node = new_node;
    list.teil_node = new_node;
}
}

void pop_element(List& list, char num_del_el, int n) { //удаление элемента
    if (list.head_node != nullptr) { //если список пустой
        Node* pointer_node = list.head_node;
        while (pointer_node != nullptr) { //пока не дойду до конца
            if (pointer_node->data == num_del_el) { //если найден ключевой элемент
                if (pointer_node == list.head_node) { //если удалять головной
элемент
                    Node* new_Head = list.head_node->ptr_to_next_node;
                    delete list.head_node; //удаляю текущий головной элемент
                    list.head_node = new_Head; //новая голова
                    pointer_node = list.head_node; //текущий элемент остается
головным
                }
                else if (pointer_node == list.teil_node) { //если удалять
хвостовой элемент
                    Node* new_Teil = list.teil_node->ptr_to_prev_node;
                    new_Teil->ptr_to_next_node = nullptr;
                    delete list.teil_node; //удаляю текущий хвостовой элемент
                    list.teil_node = new_Teil; //новый хвост
                    pointer_node = nullptr;
                }
                else {
                    pointer_node = pointer_node->ptr_to_prev_node;
                    Node* connection_node = pointer_node->ptr_to_next_node;
//связываю узлы
                    pointer_node->ptr_to_next_node = connection_node->
ptr_to_next_node;
                    connection_node->ptr_to_next_node->ptr_to_prev_node =
connection_node->ptr_to_prev_node;
                    delete connection_node; //освобождаю память
                    pointer_node = pointer_node->ptr_to_next_node; //перехожу на
следующий элемент
                }
            }
            else {
                pointer_node = pointer_node->ptr_to_next_node; //перехожу на
следующий элемент
            }
        }
    }
}

void print_list(List& list) { //вывод текущего списка
    cout << endl << "Текущий список:" << endl;
    Node* current_node = list.head_node;
    if (current_node != nullptr) { //если список не пустой
        while (current_node != nullptr) { //пока не дойду до последнего элемента
            cout << current_node->data << ' '; //вывод данных текущего узла
            current_node = current_node->ptr_to_next_node; //переход к следующему
узлу
        }
    }
    else {
        cout << "Список пустой!";
    }
    cout << endl << endl;
    delete current_node; //очищаю память
}

```

```

}

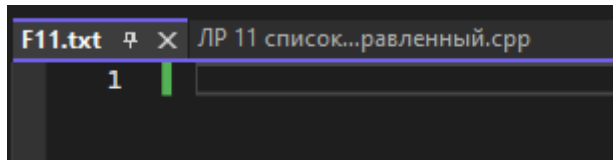
void del_all_list(List& list) {
    Node* ptr_node = list.head_node;
    if (ptr_node != nullptr) {
        while (ptr_node != nullptr) {
            Node* new_Head = list.head_node->ptr_to_next_node;
            delete list.head_node; //удаляю текущий головной элемент
            list.head_node = new_Head; //присваиваю головному элементу новый
элемент
            ptr_node = list.head_node;
        }
    }
}

void writing_to_a_file(List& list, ofstream& file) { //запись данных в файл
    if (list.head_node != nullptr) { //если список не пустой
        Node* pointer_q = list.head_node; //указатель на первый элемент
        while (pointer_q != nullptr) { //пока не дойду до конца
            file << pointer_q->data << endl;
            pointer_q = pointer_q->ptr_to_next_node; //перехожу на следующий узел
        }
    }
}

void recovery(List& list, ifstream& file) { //восстановление
    string all_str;
    getline(file, all_str); //считываю строку
    insrt_Item(list, all_str[0]);
    int counter = 0;
    while (getline(file, all_str)) { //пока не пройду весь файл
        insrt_Item(list, all_str[0], counter); //добавляю
        counter++;
    }
}

```

Результат работы программы



Введите количество элементов 6

Введите символ к
Введите символ в
Введите символ а
Введите символ к
Введите символ в
Введите символ а

Текущий список:
к в а к в а

Введите КЛЮЧ элемента, который вы хотите удалить к
Выполнено удаление элемента(-ов) с заданным ключом

Текущий список:
в а в а

Введите номер элемента, ПЕРЕД которым необходимо добавить новые элементы 4

Введите количество элементов, которые необходимо добавить 5

Введите символ ы
Введите символ Ы
Введите символ ф
Введите символ Ф
Введите символ Й

Текущий список:
в а в ы Ы ф Ф Й а

Запись данных в файл ...
Завершено

Очищение памяти ...
Завершено

Текущий список:
Список пустой!

Восстановление списка ...
Завершено

Текущий список:
в а в ы Ы ф Ф Й а

F11.txt	✕	ЛР 11 список...равленный.cpp
1		в
2		а
3		в
4		ы
5		Ы
6		ф
7		Ф
8		Й
9		а
10		

Стеки

Анализ задачи

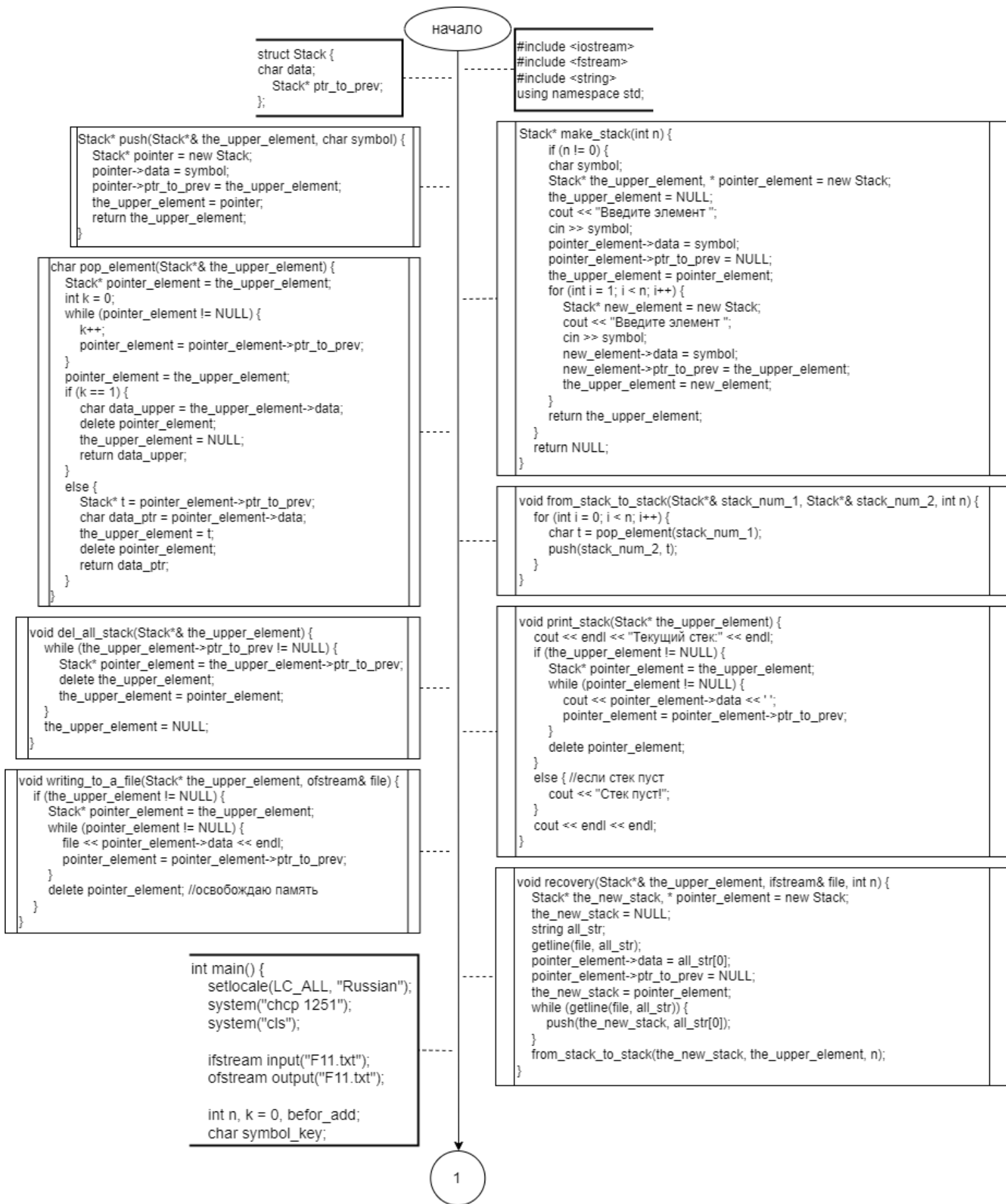
- 1) Нужно создать структуру, которая будет содержать поля (`char data;`) для данных и (`Stack* ptr_to_prev;`) для адреса следующего элемента.
- 2) Создам функцию для создания стека и его заполнения. В начале проверим равно ли количество элементов стека нулю. Если да, то возвращаем `NULL`. Иначе выделим память под 1 элемент, пользователь записывает данные и адрес на `NULL` в элемент. Затем ставим указатель на верхний элемент стека. После этого с помощью арифметического цикла введём оставшиеся элементы. В конце возвращаем указатель на верхний элемент.
- 3) Создам функцию для вывода элементов стека. В начале проверим указывает ли верхний элемент стека на `NULL`. Если да, то выводим на экран сообщение - “стек пуст”. Иначе с помощью итерационного цикла выводим элемент и переходим к следующему пока указатель не будет равен `NULL`.
- 4) Создам функцию для возвращения первого элемента и его удаления. Для начала считаем количество элементов в стеке с помощью итерационного цикла. Потом проверяем равно ли `k` единице. Если да, то обнуляем указатель и возвращаем элемент. Иначе сохраняем значение последнего элемента, делаем второй элемент первым, удаляем последний элемент и возвращаем первый элемент.
- 5) Создам функцию для добавления элемента в стек. Сначала выделим память под новый элемент. Потом присваиваем значение, которое вводит пользователь, для нового элемента, делаем указатель на нижний элемент и делаем новый элемент первым элементом стека.
- 6) Сформировать стек. Так как не сказано сколько элементов содержит стек, то пользователь должен ввести количество элементов. Потом вызвать ранее написанные функцию для создания стека и его заполнения; и функцию для вывода стека.
- 7) Чтобы удалить элемент с заданным ключом создадим второй стек, в который будут переноситься нужные элементы главного стека. Потом, так

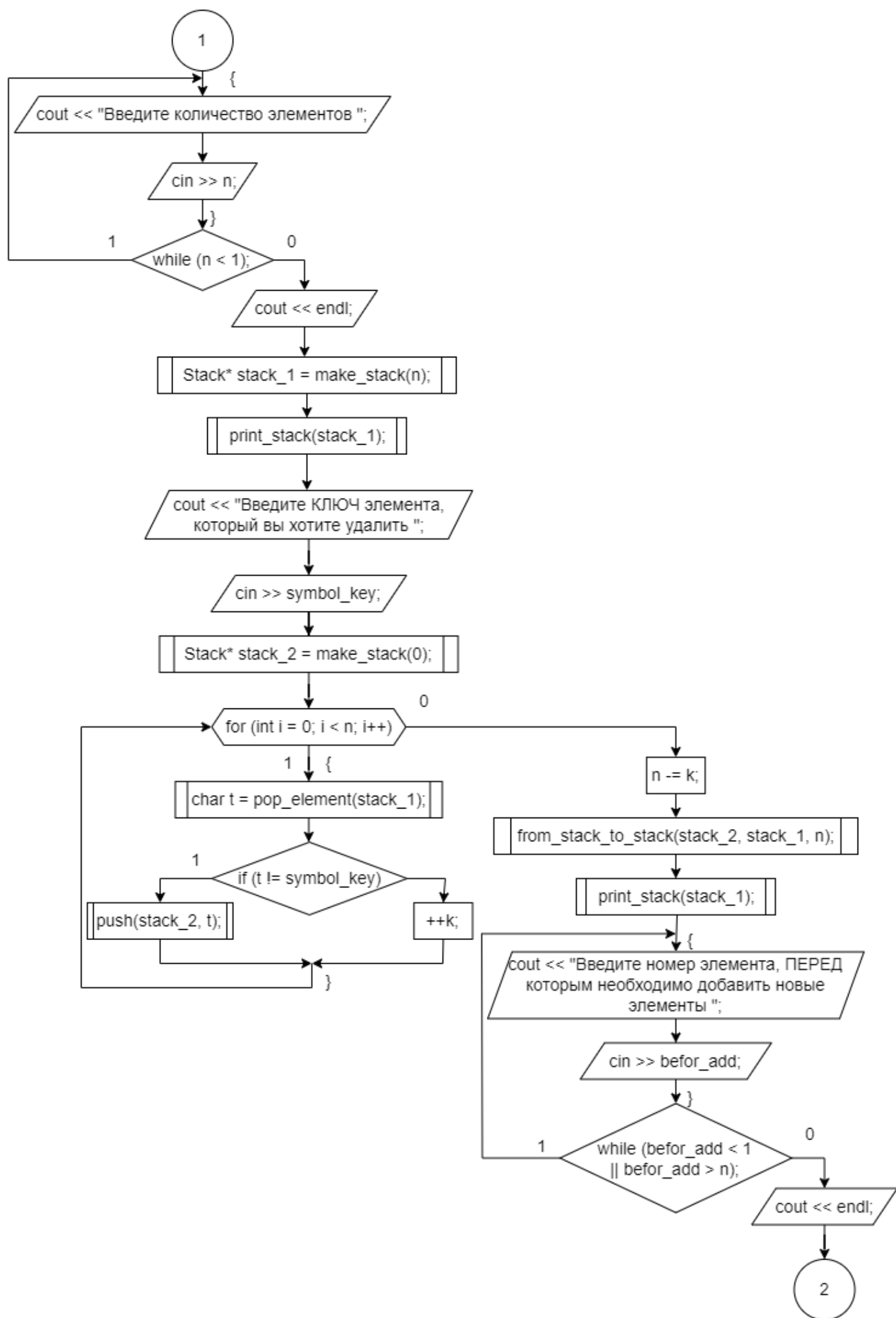
как ключ не указан, пользователь должен ввести ключ для удаления. После чего с помощью цикла и оператора выбора перенесём все нужные элементы во второй стек и посчитаем количество элементов равных ключу. Затем с помощью арифметического цикла перенесём элементы из второго стека в исходный. После чего с помощью ранее написанной функции выведем элементы стека на экран.

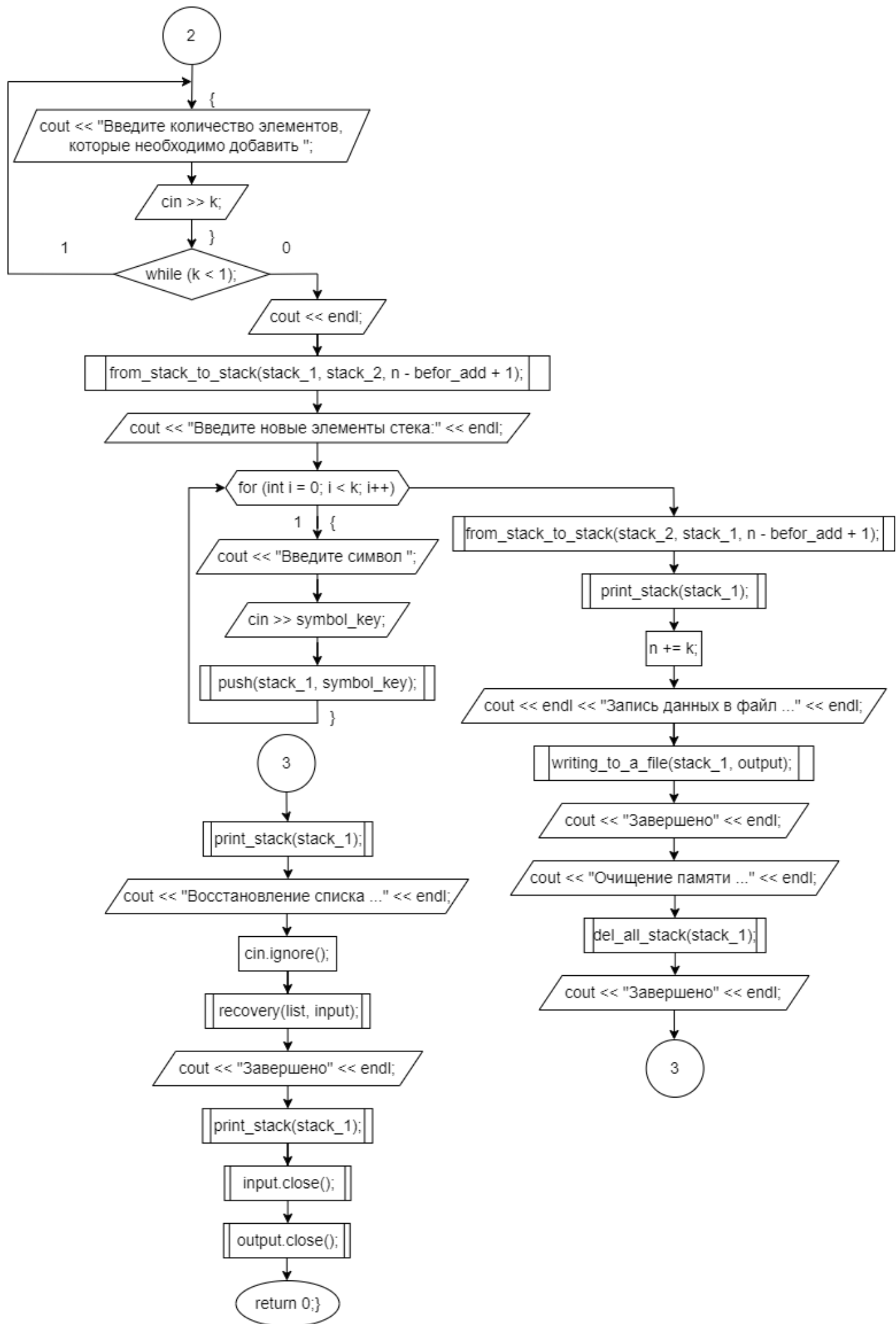
8) Добавить K элементов перед элементом с заданным номером. Так как количество элементов для добавления и номер элемента, перед которым добавляют элементы, не указаны, пользователь должен ввести их. После этого перенесём с помощью арифметического цикла нужные элементы во второй стек. Затем с помощью арифметического цикла добавим k элементов в исходный стек. После чего с помощью арифметического цикла перенесём элементы из второго стека в исходный и с помощью ранее написанной функции выведем элементы стека на экран.

9) Для работы с файлами необходимо `#include <fstream>`.
Подключение файловых потоков ввода и вывода. Функция `writing_to_a_file` с помощью итерационного цикла записывает в файл содержимое стека.
Функция `recovery` восстанавливает стек из файла с помощью итерационного цикла.

Блок схема







Код программы

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

struct Stack { //стек
    char data; //данные
    Stack* ptr_to_prev; //адрес на предыдущий элемент
};

Stack* make_stack(int n);
Stack* push(Stack*& the_upper_element, char symbol);
char pop_element(Stack*& the_upper_element); //нахожу верхний элемент
void from_stack_to_stack(Stack*& stack_num_1, Stack*& stack_num_2, int n);
void print_stack(Stack* the_upper_element); //вывод стека
void del_all_stack(Stack*& the_upper_element); //удаление всех элементов стека
void writing_to_a_file(Stack* the_upper_element, ofstream& file); //запись данных
в файл
void recovery(Stack*& the_upper_element, ifstream& file, int n); //восстановление
стека

int main() {
    setlocale(LC_ALL, "Russian"); //локализация
    system("chcp 1251");
    system("cls");

    ifstream input("Fl1.txt"); //входной файловый поток
    ofstream output("Fl1.txt"); //выходной файловый поток

    int n, k = 0, befor_add;
    char symbol_key;

    do {
        cout << "Введите количество элементов ";
        cin >> n; //количество элементов в списке
    } while (n < 1);
    cout << endl;

    Stack* stack_1 = make_stack(n); //создаю стек
    print_stack(stack_1); //вывожу текущий стек

    cout << "Введите КЛЮЧ элемента, который вы хотите удалить ";
    cin >> symbol_key;

    Stack* stack_2 = make_stack(0); //создаю пустой стек
    for (int i = 0; i < n; i++) {
        //перенос всех элементов кроме элемента с КЛЮЧОМ во второй стек
        char t = pop_element(stack_1); //извлекаю верхний элемент
        if (t != symbol_key) { //если символ не равен ключевому
            push(stack_2, t); //переношу во второй стек
        }
        else { //если символ равен ключевому
            ++k; //увеличиваю количество ключевых символов
        }
    }
    n -= k; //изменяю количество элементов в стеке

    from_stack_to_stack(stack_2, stack_1, n);
    print_stack(stack_1); //вывожу текущий стек

    do {
        cout << "Введите количество элементов, которые необходимо ДОБАВИТЬ ";
        cin >> k; //количество элементов, которые надо добавить
    } while (k < 1);
    cout << endl;
```

```

do {
    cout << "Введите номер элемента, ПЕРЕД которым необходимо добавить
новые элементы ";
    cin >> befor_add; //НОМЕР элемента
} while (befor_add < 1 || befor_add > n);
cout << endl;

from_stack_to_stack(stack_1, stack_2, n - befor_add + 1);

cout << "Введите новые элементы стека:" << endl;
for (int i = 0; i < k; i++) {
    cout << "Введите символ ";
    cin >> symbol_key;
    push(stack_1, symbol_key); //добавляю символ в стек 1
}

from_stack_to_stack(stack_2, stack_1, n - befor_add + 1);
print_stack(stack_1); //вывожу текущий стек

n += k; //изменяю количество элементов в стеке

cout << "Запись данных в файл ..." << endl;
writing_to_a_file(stack_1, output);
cout << "Завершено" << endl << endl;

cout << "Очищение памяти ..." << endl;
del_all_stack(stack_1); //очищаю весь стек
cout << "Завершено" << endl;
print_stack(stack_1); //вывожу текущий стек

cout << "Восстановление стека ..." << endl;
cin.ignore();
recovery(stack_1, input, n);
cout << "Завершено" << endl;

print_stack(stack_1); //вывожу текущую очередь

input.close(); //закрываю файл
output.close(); //закрываю файл
return 0;
}

Stack* make_stack(int n) {
    if (n != 0) {
        char symbol;
        Stack* the_upper_element, * pointer_element = new Stack;
        the_upper_element = NULL;

        cout << "Введите элемент ";
        cin >> symbol;

        pointer_element->data = symbol; //присваиваю значение новому элементу
        pointer_element->ptr_to_prev = NULL; //адрес на предыдущий элемент
        the_upper_element = pointer_element; //изменяю верхний элемент в
стеке

        for (int i = 1; i < n; i++) { //добавление новых элемнтов
            Stack* new_element = new Stack;
            cout << "Введите элемент ";
            cin >> symbol;

            new_element->data = symbol;
            new_element->ptr_to_prev = the_upper_element;
            the_upper_element = new_element;
        }
        return the_upper_element;
    }
}

```

```

    }
    return NULL;
}

Stack* push(Stack*& the_upper_element, char symbol) { //добавление элемента
    Stack* pointer = new Stack; //выделяю память для нового стека
    pointer->data = symbol; //присваиваю данные новому элементу
    pointer->ptr_to_prev = the_upper_element; //адрес предыдущего элемента
    the_upper_element = pointer; //новый верхний элемент
    return the_upper_element; //возвращаю новый верхний элемент
}

char pop_element(Stack*& the_upper_element) { //нахожу верхний элемент
    Stack* pointer_element = the_upper_element;
    int k = 0;
    while (pointer_element != NULL) {
        k++;
        pointer_element = pointer_element->ptr_to_prev;
    }
    pointer_element = the_upper_element;

    if (k == 1) {
        char data_upper = the_upper_element->data; //данные верхнего элемента
        delete pointer_element;
        the_upper_element = NULL;
        return data_upper;
    }
    else {
        Stack* t = pointer_element->ptr_to_prev;
        char data_ptr = pointer_element->data; //данные необходимого элемента
        the_upper_element = t; //меняю верхний элемент
        delete pointer_element;
        return data_ptr;
    }
    //the_upper_element - верхний элемент
}

void from_stack_to_stack(Stack*& stack_num_1, Stack*& stack_num_2, int n) {
    //перенос элементов из стека в стек
    for (int i = 0; i < n; i++) { //переношу элементы из стека 1 в стек 2
        char t = pop_element(stack_num_1); //извлекаю верхний элемент из
        первого
        push(stack_num_2, t); //переношу элементы во второй стек
    }
}

void print_stack(Stack* the_upper_element) { //вывод стека
    cout << endl << "Текущий стек:" << endl;
    if (the_upper_element != NULL) { //если стек не пустой
        Stack* pointer_element = the_upper_element; //указатель на первый
        элемент
        while (pointer_element != NULL) { //пока не дойду до нижнего элемента
            cout << pointer_element->data << ' ';
            pointer_element = pointer_element->ptr_to_prev;
        }
        delete pointer_element; //освобождаю память
    }
    else { //если стек пуст
        cout << "Стек пуст!";
    }
    cout << endl << endl;
}

void del_all_stack(Stack*& the_upper_element) { //удаление всех элементов стека
    while (the_upper_element->ptr_to_prev != NULL) {
        Stack* pointer_element = the_upper_element->ptr_to_prev;
        delete the_upper_element; //освобождение памяти верхнего элемента
    }
}

```

```

        the_upper_element = pointer_element;
    }
    the_upper_element = NULL;
}

void writing_to_a_file(Stack* the_upper_element, ofstream& file) { //запись данных
в файл
    if (the_upper_element != NULL) { //если стек не пустой
        Stack* pointer_element = the_upper_element; //указатель на первый
элемент
        while (pointer_element != NULL) { //пока не дойду до нижнего элемента
            file << pointer_element->data << endl;
            pointer_element = pointer_element->ptr_to_prev;
        }
        delete pointer_element; //освобождаю память
    }
}

void recovery(Stack*& the_upper_element, ifstream& file, int n) { //восстановление
стека
    Stack* the_new_stack, * pointer_element = new Stack;
    the_new_stack = NULL;

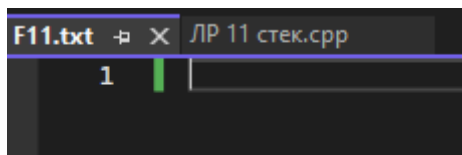
    string all_str;
    getline(file, all_str); //считываю строку

    pointer_element->data = all_str[0]; //присваиваю значение новому элементу
    pointer_element->ptr_to_prev = NULL; //адрес на предыдущий элемент
    the_new_stack = pointer_element; //изменяю верхний элемент в стеке

    while (getline(file, all_str)) { //пока не пройду весь файл
        push(the_new_stack, all_str[0]); //переношу элементы во временный
стек
    }
    from_stack_to_stack(the_new_stack, the_upper_element, n); //переношу
элементы в главный стек
}

```

Результат работы программы



```

Введите количество элементов 8

Введите элемент п
Введите элемент у
Введите элемент п
Введите элемент у
Введите элемент п
Введите элемент у
Введите элемент п
Введите элемент у

Текущий стек:
у п у п у п у п

Введите КЛЮЧ элемента, который вы хотите удалить у

Текущий стек:
п п п п

Введите количество элементов, которые необходимо ДОБАВИТЬ 1

Введите номер элемента, ПЕРЕД которым необходимо добавить новые элементы 3

Введите новые элементы стека:
Введите символ +

Текущий стек:
п п + п п

Запись данных в файл ...
Завершено

Очищение памяти ...
Завершено

Текущий стек:
Стек пуст!

Восстановление стека ...
Завершено

Текущий стек:
п п + п п

```

F11.txt		✕	ЛР 11 стек.cpp
1			п
2			п
3			+
4			п
5			п
6			

Очереди

Анализ задачи

- 1) Структура данных "очередь" представлена структурой Queue, которая содержит указатели на начало и конец очереди, а также размер очереди. Элементы очереди представлены структурой Node, которая содержит данные и указатель на следующий элемент.
- 2) Функция `init_queue` инициализирует очередь, создавая новый элемент и устанавливая его как начало и конец очереди.
- 3) Функция `push_element` добавляет новый элемент в конец очереди, увеличивает размер очереди и обновляет указатели на начало и конец очереди.
- 4) Функция `new_queue` формирует очередь, запрашивая у пользователя ввод элементов и вызывая функцию `push_element` для каждого элемента.
- 5) Функция `print_queue` выводит очередь. В начале проверяет указывает ли верхний элемент стека на `nullptr`. Если да, то выводим на экран сообщение - "стек пуст". Иначе с помощью итерационного цикла выводим элемент и переходим к следующему пока указатель не будет равен `nullptr`.
- 6) Функция `pop_element` удаляет головной элемент из очереди, обновляет указатели на начало и конец очереди и освобождает память.
- 7) Функция `delete_key` удаляет все элементы, равные заданному ключу, из очереди. Функция перебирает элементы через итерационный цикл.
- 8) Функция `insert` вставляет элементы в заданное место в очереди. Функция содержит итерационный цикл для поиска элемента перед которым надо вставить элементы; и два арифметических цикла для перестановки существующих элементов и добавления новых.
- 9) Функция `delete_all_queue` удаляет все элементы из очереди – освобождает память.
- 10) В основной функции создается очередь, затем вызываются функции `new_queue`, `print_queue` и `delete_all_queue`.
- 11) Для работы с файлами необходимо `#include <fstream>`.
Подключение файловых потоков ввода и вывода. Функция `writing_to_a_file` с

помощью итерационного цикла записывает в файл содержимое очереди.
Функция `recoveru` восстанавливает очередь из файла с помощью итерационного цикла.

Блок схема

начало

```
struct Node {  
    char data;  
    Node* ptr_to_next_node;  
};
```

```
struct Queue {  
    int size;  
    Node* head_node = nullptr;  
    Node* teil_node = nullptr;  
};
```

```
void push_element(Queue& line, const char& symbol) {  
    Node* new_node = new Node;  
    line.size++;  
    new_node->data = symbol;  
    new_node->ptr_to_next_node = nullptr;  
    line.teil_node->ptr_to_next_node = new_node; и  
    line.teil_node = new_node;  
}
```

```
void print_queue(Queue& line) {  
    cout << endl << "Текущая очередь: " << endl;  
    if (line.head_node != nullptr) {  
        Node* pointer_q = line.head_node;  
        cout << "start -> ";  
        while (pointer_q != nullptr) {  
            cout << pointer_q->data << " ";  
            pointer_q = pointer_q->ptr_to_next_node;  
        }  
        cout << "-> end" << endl << endl;  
    }  
    else {  
        cout << "Очередь пуста!" << endl << endl;  
    }  
}
```

```
void delete_key(Queue& line, char symbol) {  
    int counter = 0;  
    while (counter != line.size) {  
        if (line.head_node->data == symbol) {  
            pop_element(line);  
        }  
        else {  
            push_element(line, line.head_node->data);  
            pop_element(line);  
            ++counter;  
        }  
    }  
}
```

```
void delete_all_queue(Queue& line) {  
    while (line.head_node->ptr_to_next_node != nullptr) {  
        pop_element(line);  
    }  
    Node* pointer_q = line.head_node;  
    line.head_node = nullptr;  
    --line.size; //уменьшаю размер  
    delete pointer_q;  
}
```

```
void recovery(Queue& line, ifstream& file) {  
    string all_str;  
    getline(file, all_str);  
    init_queue(line, all_str[0]);  
    while (getline(file, all_str)) {  
        push_element(line, all_str[0]);  
    }  
}
```

```
#include <iostream>  
#include <fstream>  
#include <string>  
using namespace std;
```

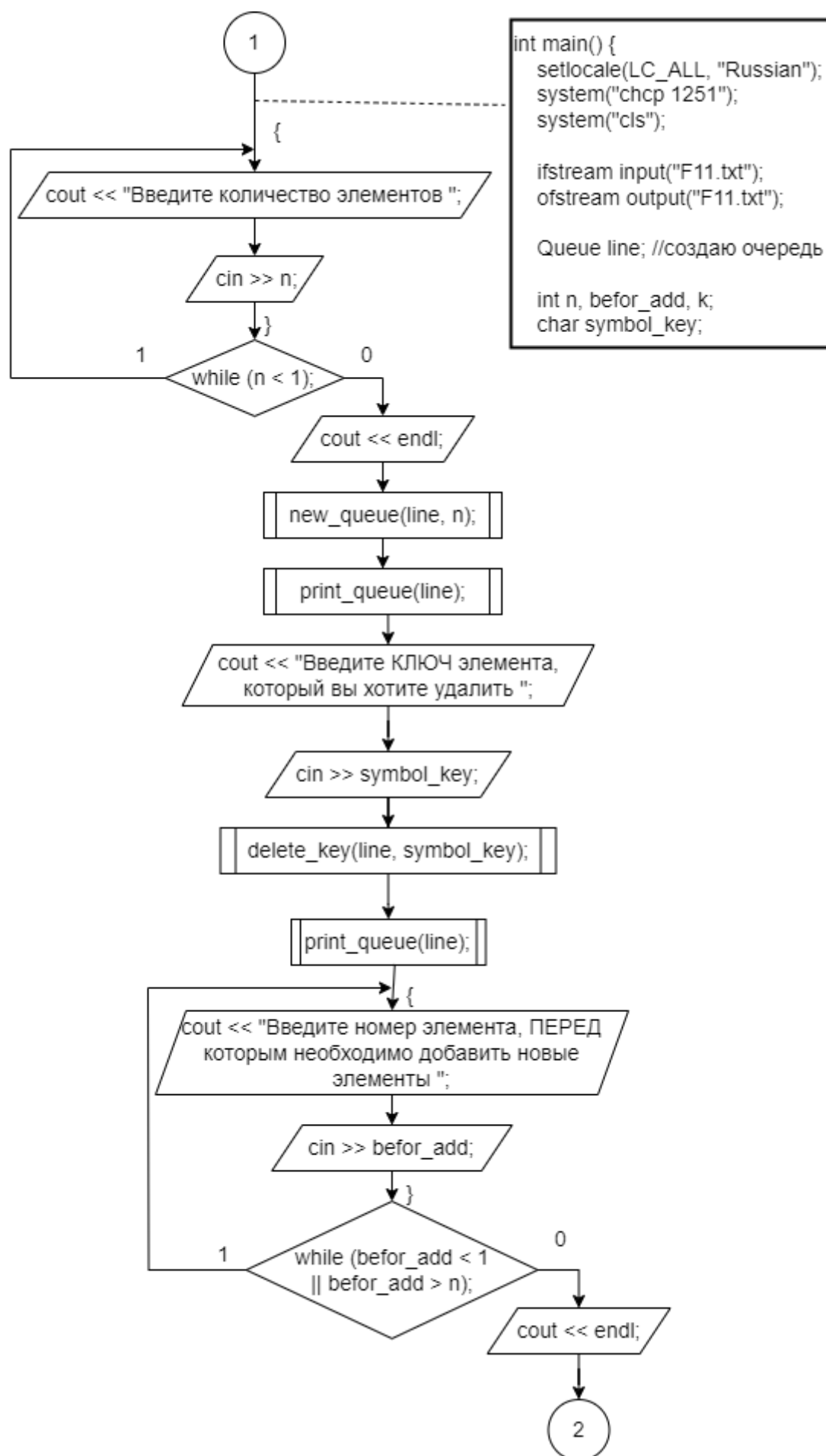
```
void init_queue(Queue& line, const char& symbol) {  
    Node* new_node = new Node;  
    new_node->data = symbol;  
    line.head_node = new_node;  
    line.teil_node = new_node;  
    line.size = 1;  
}
```

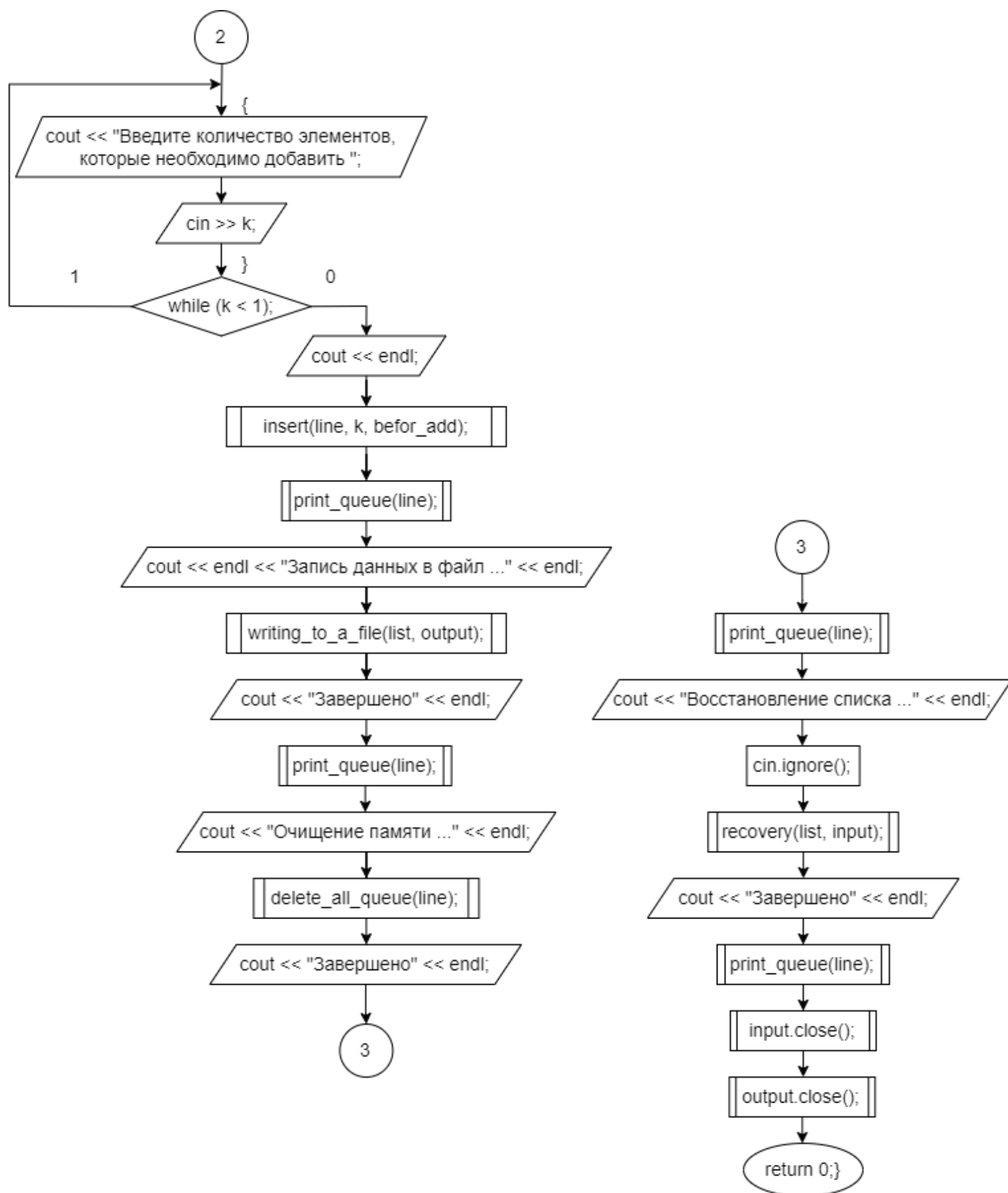
```
void new_queue(Queue& line, int n) {  
    char symbol;  
    cout << "Введите элемент ";  
    cin >> symbol;  
    init_queue(line, symbol);  
    for (int i = 1; i < n; i++) {  
        cout << "Введите элемент ";  
        cin >> symbol;  
        push_element(line, symbol);  
    }  
}
```

```
void pop_element(Queue& line) {  
    Node* pointer_q = line.head_node;  
    line.head_node = line.head_node->ptr_to_next_node;  
    --line.size;  
    delete pointer_q;  
}
```

```
void insert(Queue& line, int quantity, int number) { /  
    int counter = 1;  
    char symbol;  
    while (counter != number) {  
        push_element(line, line.head_node->data);  
        pop_element(line);  
        ++counter;  
    }  
    for (int i = 0; i < quantity; i++) {  
        cout << "Введите символ ";  
        cin >> symbol;  
        push_element(line, symbol);  
    }  
    for (int i = 0; i <= line.size - quantity - number; i++) {  
        push_element(line, line.head_node->data);  
        pop_element(line);  
    }  
}
```

```
void writing_to_a_file(Queue& line, ofstream& file) {  
    if (line.head_node != nullptr) {  
        Node* pointer_q = line.head_node;  
        while (pointer_q != nullptr) {  
            file << pointer_q->data << endl;  
            pointer_q = pointer_q->ptr_to_next_node;  
        }  
    }  
}
```





Код программы

```

#include <iostream>
#include <fstream>

```

```

#include <string>
using namespace std;

struct Node {
    char data; //данные
    Node* ptr_to_next_node; //указатель на следующий элемент
};

struct Queue { //очередь
    int size; //размер
    Node* head_node = nullptr;
    Node* teill_node = nullptr;
};

void init_queue(Queue& line, const char& symbol); //инициализирую очередь
void push_element(Queue& line, const char& symbol); //добавляю элемент в конец
очереди
void new_queue(Queue& line, int n); //формирую очередь
void print_queue(Queue& line); //вывод очереди
void pop_element(Queue& line); //удаляю головной элемент из очереди
void delete_key(Queue& line, char symbol); //удаляю элемент по ключу
void insert(Queue& line, int quantity, int number); //вставляю элементы в нужное
место
void delete_all_queue(Queue& line);
void writing_to_a_file(Queue& line, ofstream& file); //запись данных в файл
void recovery(Queue& line, ifstream& file); //восстановление

int main() {
    setlocale(LC_ALL, "Russian"); //локализация
    system("chcp 1251");
    system("cls");

    ifstream input("F11.txt"); //входной файловый поток
    ofstream output("F11.txt"); //выходной файловый поток

    Queue line; //создаю очередь

    int n, befor_add, k;
    char symbol_key;

    do {
        cout << "Введите количество элементов ";
        cin >> n; //количество элементов в списке
    } while (n < 1);
    cout << endl;

    new_queue(line, n); //формирую новую очередь
    print_queue(line); //вывожу текущую очередь

    cout << "Введите КЛЮЧ элемента, который вы хотите удалить ";
    cin >> symbol_key;

    delete_key(line, symbol_key); //удаляю ключ
    print_queue(line); //вывожу текущую очередь

    do {
        cout << "Введите номер элемента, ПЕРЕД которым необходимо добавить
новые элементы ";
        cin >> befor_add; //НОМЕР элемента
    } while (befor_add < 1 || befor_add > line.size);
    cout << endl;

    do {
        cout << "Введите количество элементов, которые необходимо добавить ";
        cin >> k; //количество элементов, которые надо добавить
    } while (k < 1);

```

```

    cout << endl;

    insert(line, k, befor_add); //вставляю новые элементы
    print_queue(line); //вывожу текущую очередь

    cout << "Запись данных в файл ..." << endl;
    writing_to_a_file(line, output);
    cout << "Завершено" << endl << endl;

    cout << "Очищение памяти ..." << endl;
    delete_all_queue(line); //очищаю всю очередь
    cout << "Завершено" << endl;

    print_queue(line); //вывожу текущую очередь

    cout << "Восстановление очереди ..." << endl;
    cin.ignore();
    recovery(line, input);
    cout << "Завершено" << endl;

    print_queue(line); //вывожу текущую очередь

    input.close(); //закрываю файл
    output.close(); //закрываю файл
    return 0;
}

void init_queue(Queue& line, const char& symbol) { //инициализирую очередь
    Node* new_node = new Node; //резервирую память под новый элемент
    new_node->data = symbol; //присваиваю данные
    line.head_node = new_node; //головной элемент
    line.teil_node = new_node; //хвостовой элемент
    line.size = 1; //размер очереди, т.к. есть только один элемент
}

void push_element(Queue& line, const char& symbol) { //добавляю элемент в конец
очереди
    Node* new_node = new Node; //резервирую память под новый элемент
    line.size++; //увеличиваю текущий размер очереди на один
    new_node->data = symbol; //присваиваю данные
    new_node->ptr_to_next_node = nullptr; //последний элемент не указывает на
ч-л
    line.teil_node->ptr_to_next_node = new_node; //прошлый последний элемент
указывает на новый последний
    line.teil_node = new_node; //новый хвостовой элемент
}

void new_queue(Queue& line, int n) { //формирую очередь
    char symbol;
    cout << "Введите элемент ";
    cin >> symbol;
    init_queue(line, symbol); //инициализирую первым элементом
    for (int i = 1; i < n; i++) {
        cout << "Введите элемент ";
        cin >> symbol; //ввод элемента
        push_element(line, symbol); //ставлю новый элемент в конец очереди
    }
}

void print_queue(Queue& line) { //вывод очереди
    cout << endl << "Текущая очередь: " << endl;
    if (line.head_node != nullptr) {
        Node* pointer_q = line.head_node; //указатель на первый элемент
        cout << "start -> ";
        while (pointer_q != nullptr) { //пока не дойду до конца
            cout << pointer_q->data << ' '; //вывожу значение текущего
элемента

```

```

        pointer_q = pointer_q->ptr_to_next_node; //перехожу на
следующий узел
    }
    cout << "-> end" << endl << endl;
}
else {
    cout << "Очередь пуста!" << endl << endl;
}
}

void pop_element(Queue& line) { //удаляю головной элемент из очереди
    Node* pointer_q = line.head_node; //указатель на первый элемент
    line.head_node = line.head_node->ptr_to_next_node; //голова - следующий
элемент
    --line.size; //уменьшаю длину очереди
    delete pointer_q; //освобождаю память
}

void delete_key(Queue& line, char symbol) { //удаляю элемент по ключу
    int counter = 0; //счетчик
    while (counter != line.size) { //пока не обойду всю очередь
        if (line.head_node->data == symbol) { //если первый элемент - ключ
            pop_element(line); //удаляю голову
        }
        else { //если первый элемент - НЕ ключ
            push_element(line, line.head_node->data); //переставляю
головной элемент в конец очереди
            pop_element(line); //удаляю головной элемент
            ++counter;
        }
    }
}

void insert(Queue& line, int quantity, int number) { //вставляю элементы в нужное
место
    int counter = 1;
    char symbol;
    while (counter != number) { //пока не дойду до number
        push_element(line, line.head_node->data); //переставляю головной
элемент в конец очереди
        pop_element(line); //удаляю головной элемент
        ++counter;
    }
    for (int i = 0; i < quantity; i++) { //добавляю новые элементы
        cout << "Введите символ ";
        cin >> symbol; //ввожу новый символ
        push_element(line, symbol); //добавляю новый элемент в конец очереди
    }
    for (int i = 0; i <= line.size - quantity - number; i++) { //переставляю
оставшиеся элементы
        push_element(line, line.head_node->data); //переставляю головной
элемент в конец очереди
        pop_element(line); //удаляю головной элемент
    }
}

void delete_all_queue(Queue& line) {
    while (line.head_node->ptr_to_next_node != nullptr) { //пока не дойду до
последнего элемента
        pop_element(line); //удаляю головной элемент
    }
    Node* pointer_q = line.head_node; //указатель на первый элемент
    line.head_node = nullptr;
    --line.size; //уменьшаю размер
    delete pointer_q; //освобождаю память от последнего элемента
}

```

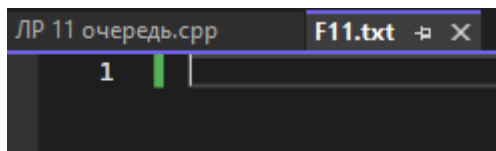
```

void writing_to_a_file(Queue& line, ofstream& file) { //запись данных в файл
    if (line.head_node != nullptr) {
        Node* pointer_q = line.head_node; //указатель на первый элемент
        while (pointer_q != nullptr) { //пока не дойду до конца
            file << pointer_q->data << endl;
            pointer_q = pointer_q->ptr_to_next_node; //перехожу на
следующий узел
        }
    }
}

void recovery(Queue& line, ifstream& file) { //восстановление
    string all_str;
    getline(file, all_str); //считываю строку
    init_queue(line, all_str[0]); //добавляю в очередь
    while (getline(file, all_str)) { //пока не пройду весь файл
        push_element(line, all_str[0]); //добавляю в очередь
    }
}

```

Результат работы программы



Введите количество элементов 8

Введите элемент K

Введите элемент y

Введите элемент k

Введите элемент A

Введите элемент p

Введите элемент e

Введите элемент K

Введите элемент y

Текущая очередь:

start -> K y k A p e K y -> end

Введите КЛЮЧ элемента, который вы хотите удалить K

Текущая очередь:

start -> y k A p e y -> end

Введите номер элемента, ПЕРЕД которым необходимо добавить новые элементы 5

Введите количество элементов, которые необходимо добавить 5

Введите символ q

Введите символ w

Введите символ e

Введите символ r

Введите символ t

Текущая очередь:

start -> y k A p q w e r t e y -> end

Запись данных в файл ...

Завершено

Очищение памяти ...

Завершено

Текущая очередь:

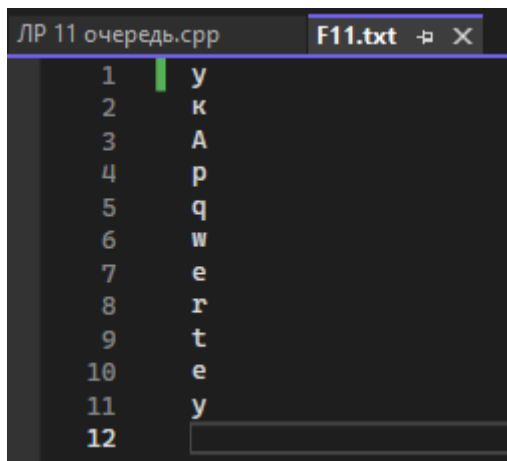
Очередь пуста!

Восстановление очереди ...

Завершено

Текущая очередь:

start -> y k A p q w e r t e y -> end

A screenshot of a code editor window titled 'ЛР 11 очередь.cpp'. The editor shows a list of 12 lines of code, each containing a number followed by a Cyrillic letter. The letters are: 1 у, 2 к, 3 А, 4 р, 5 q, 6 w, 7 е, 8 г, 9 т, 10 е, 11 у, and 12 followed by an empty input field. The window has a tab titled 'F11.txt' and standard window controls (minimize, maximize, close).

```
1 у
2 к
3 А
4 р
5 q
6 w
7 е
8 г
9 т
10 е
11 у
12 
```

Вывод

В ходе работы я применила знания о работе со списками, стеками и очередями. Также были созданы соответствующие функции для обработки необходимой структуры, например: создание структуры, добавление и удаление элементов, а также функция освобождения памяти. В результате работы мне удалось реализовать поставленную задачу используя: однонаправленный список, двунаправленный список, стек и очередь.

GitHub

Ссылка: <https://github.com/SonyAkb/laboratory-work-11.git>

main 1 Branch 0 Tags

Go to file

Add file

Code

SonyAkb Create README.md

94ede85 - 2 hours ago

32 Commits

F11.txt	Работа с файлом	6 hours ago
README.md	Create README.md	2 hours ago
ЛР 11 очередь.cpp	Работа с файлом	6 hours ago
ЛР 11 список 1-направленный.cpp	Работа с файлом	6 hours ago
ЛР 11 список 2-направленный.cpp	Работа с файлом	6 hours ago
ЛР 11 стек.cpp	Работа с файлом	6 hours ago
блок схема 1-ый список.drawio.png	Обновление блок схема 1-ый список.drawio.png	4 hours ago
блок схема 2-ый список.drawio.png	Обновление блок схема 2-ый список.drawio.png	4 hours ago
блок схема стек.drawio.png	Обновление блок схема стек.drawio.png	3 hours ago
блок схема очередь.drawio.png	Обновление блок схема очередь.drawio.png	3 hours ago

README

**Добораторна работа 11 - Информационные динамические структуры - вариант №25**

Цель: Знакомство с динамическими информационными структурами на примере одно- и двунаправленных списков.

Постановка задачи: написать программу, в которой создаются динамические структуры и выполнить их обработку в соответствии со своим вариантом.

Задача: Записи в линейном списке содержат ключевое поле типа `*char`(строка символов). Сформировать двунаправленный список. Удалить элемент с заданным ключом. Добавить К элементов перед элементом с заданным номером.

Необходимо разработать следующие функции:

1. Создание списка.
2. Добавление элемента в список (в соответствии со своим вариантом).
3. Удаление элемента из списка (в соответствии со своим вариантом).
4. Печать списка.
5. Запись списка в файл.
6. Уничтожение списка.
7. Восстановление списка из файла.