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-23-3б

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: однонаправленный список, двунаправленный список, стек и очередь. 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.

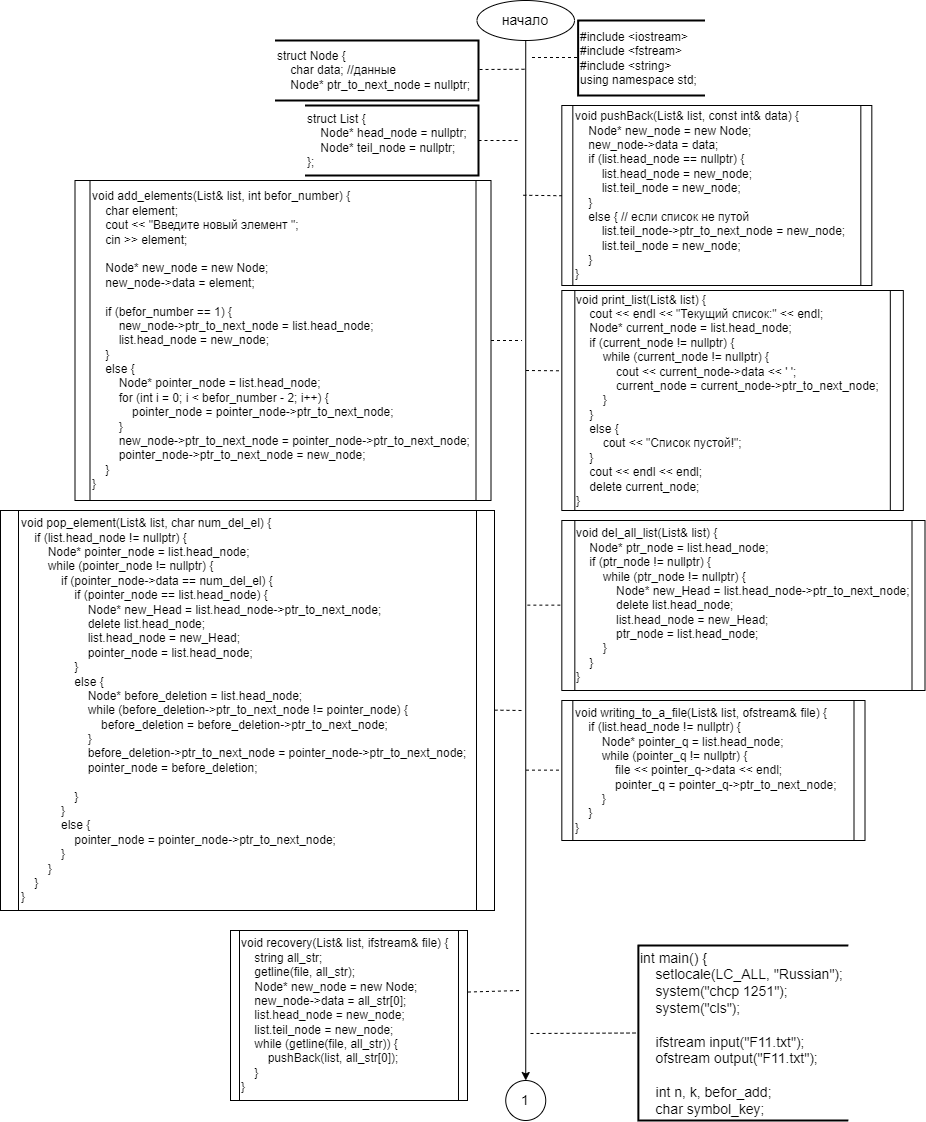
7. Восстановление списка из файла.

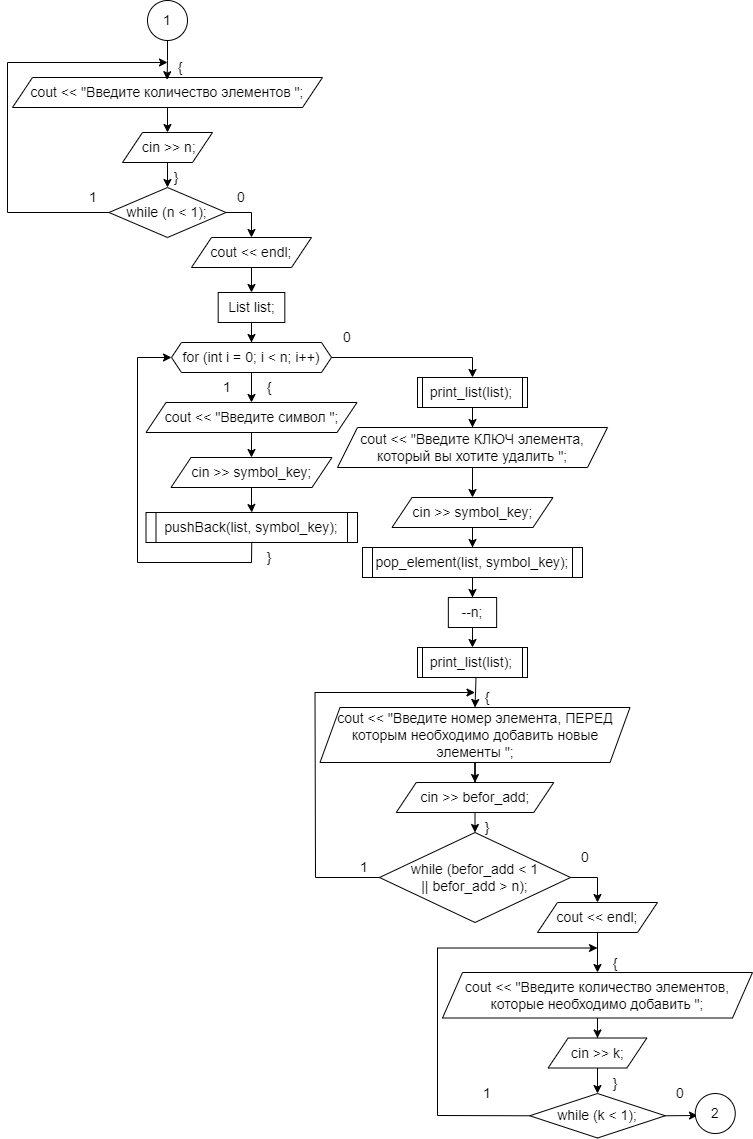
# **Однонаправленные списки**

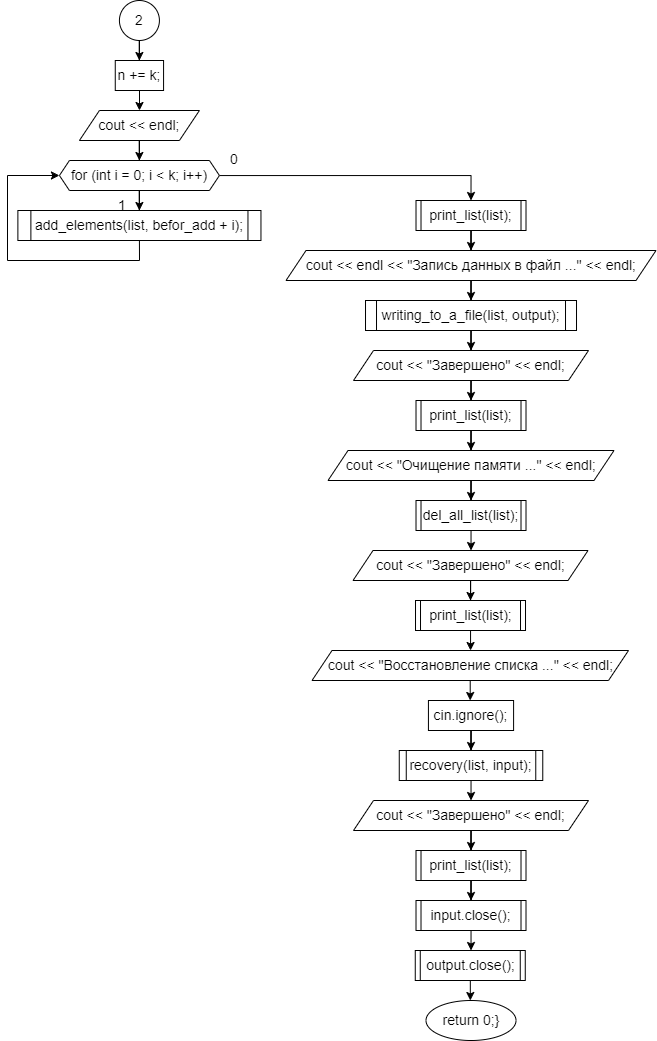
## Анализ задачи

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 teil\_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 recovery 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\_next\_node = nullptr; //указатель на следующий элемент

};

struct List {

Node\* head\_node = nullptr;

Node\* teil\_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 путой

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 ввовдить 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) { //освобожденеи 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.teil\_node = new\_node; //новый узел - хвостовой узел списка

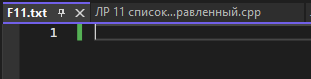
while (getline(file, all\_str)) { //пока не пройду весь файл

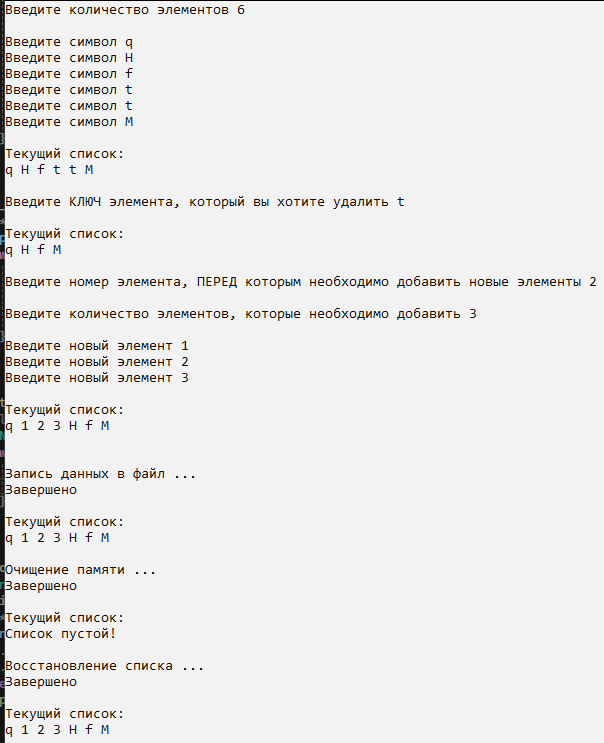
pushBack(list, all\_str[0]); //добавляю

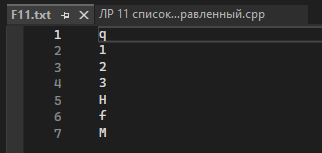
}

}

## Результат работы программы





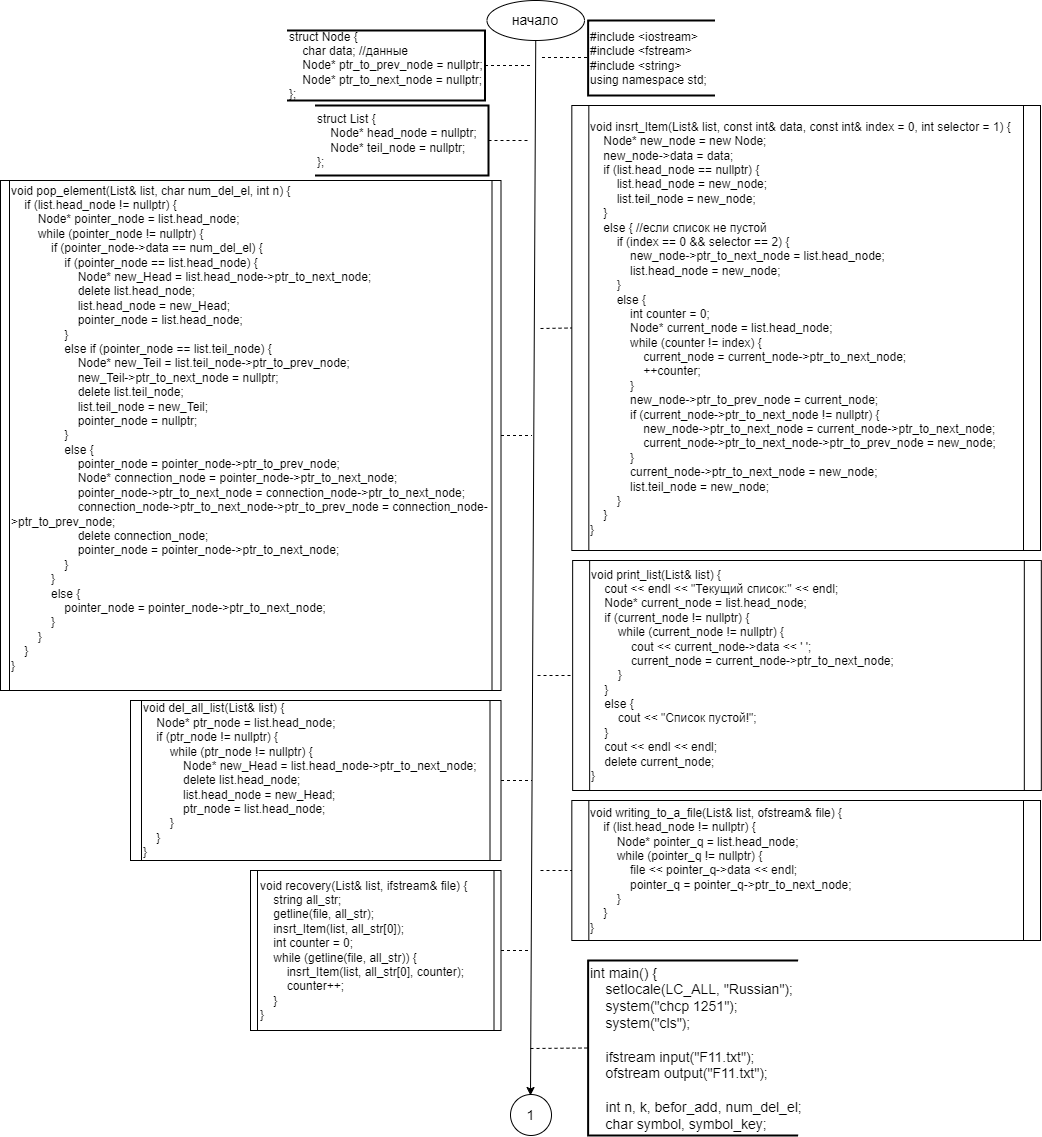


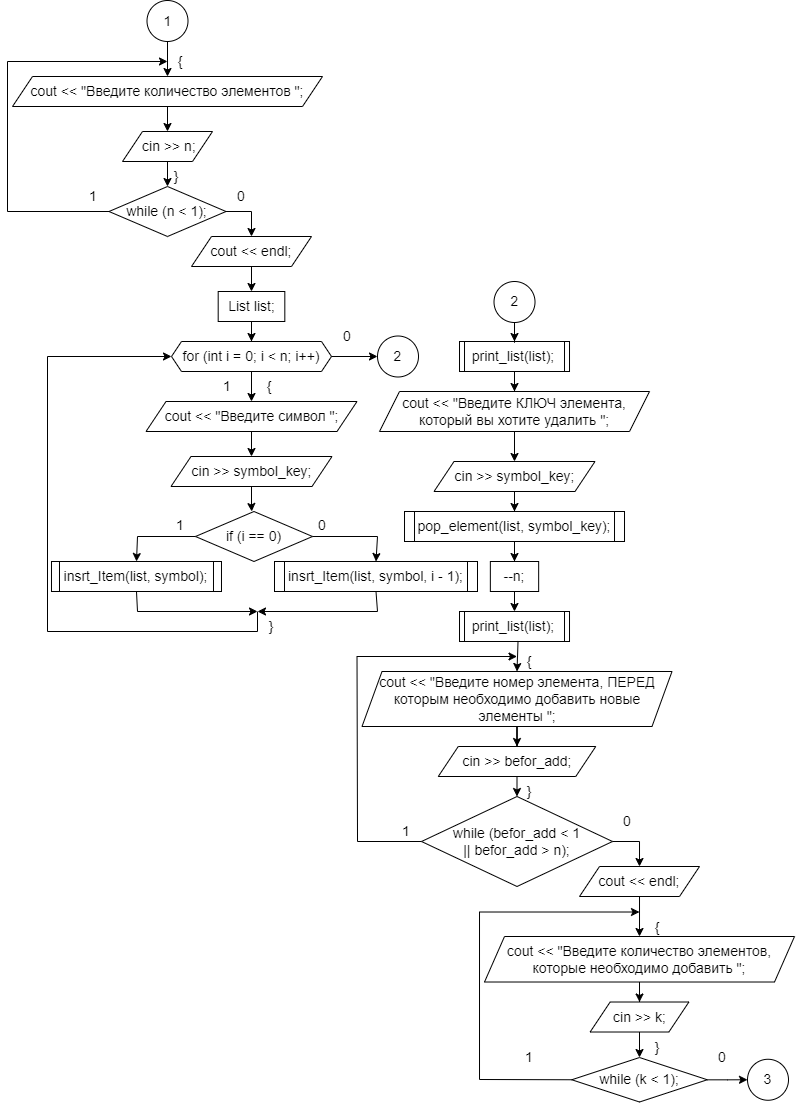
# **Двунаправленные списки**

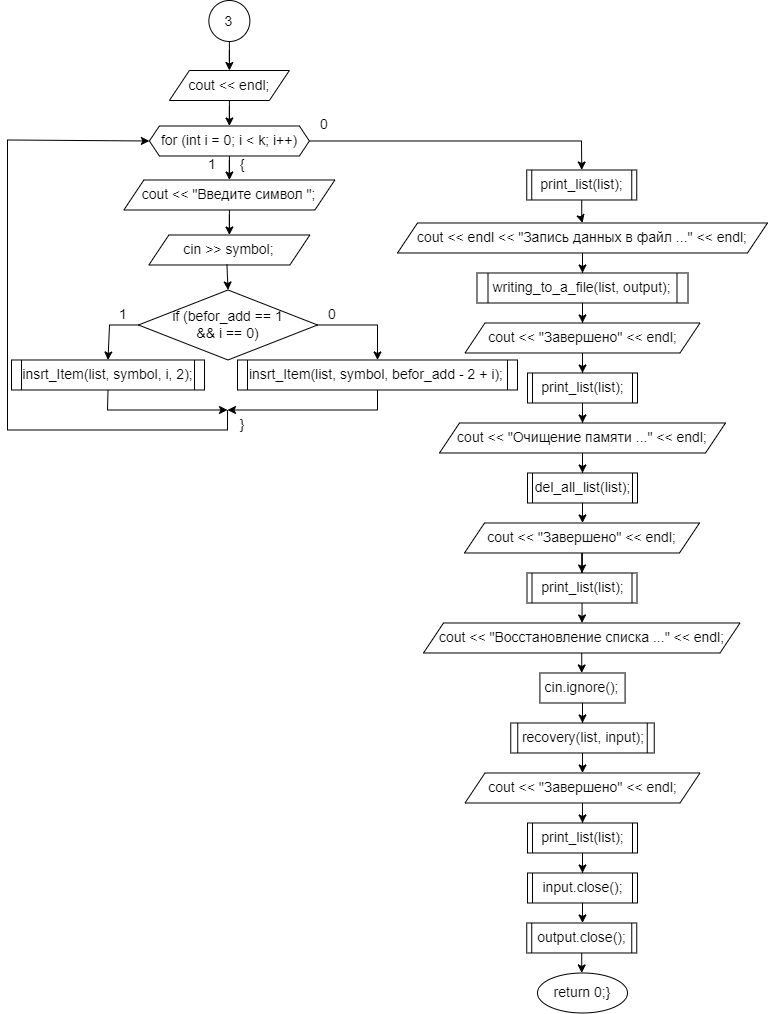
## Анализ задачи

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 recovery 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); //очищение 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

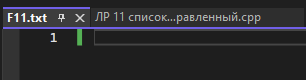
insrt\_Item(list, all\_str[0], counter); //добавляю

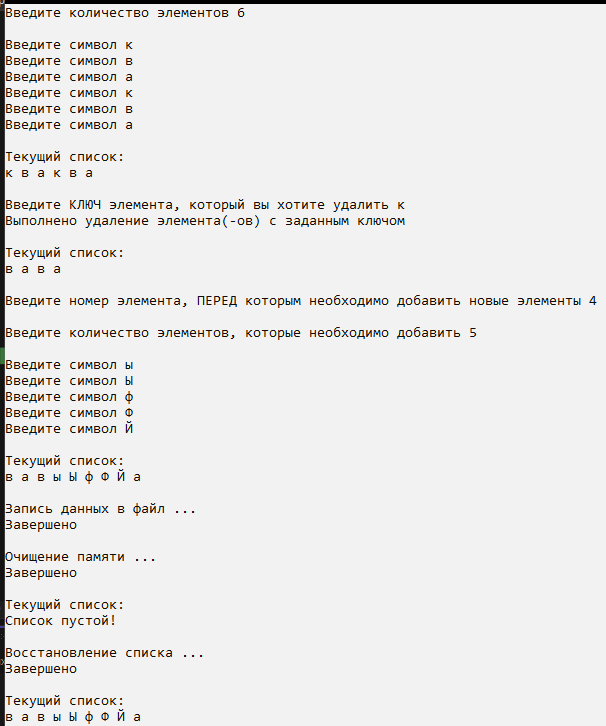
counter++;

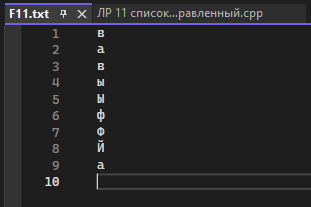
}

}

## Результат работы программы





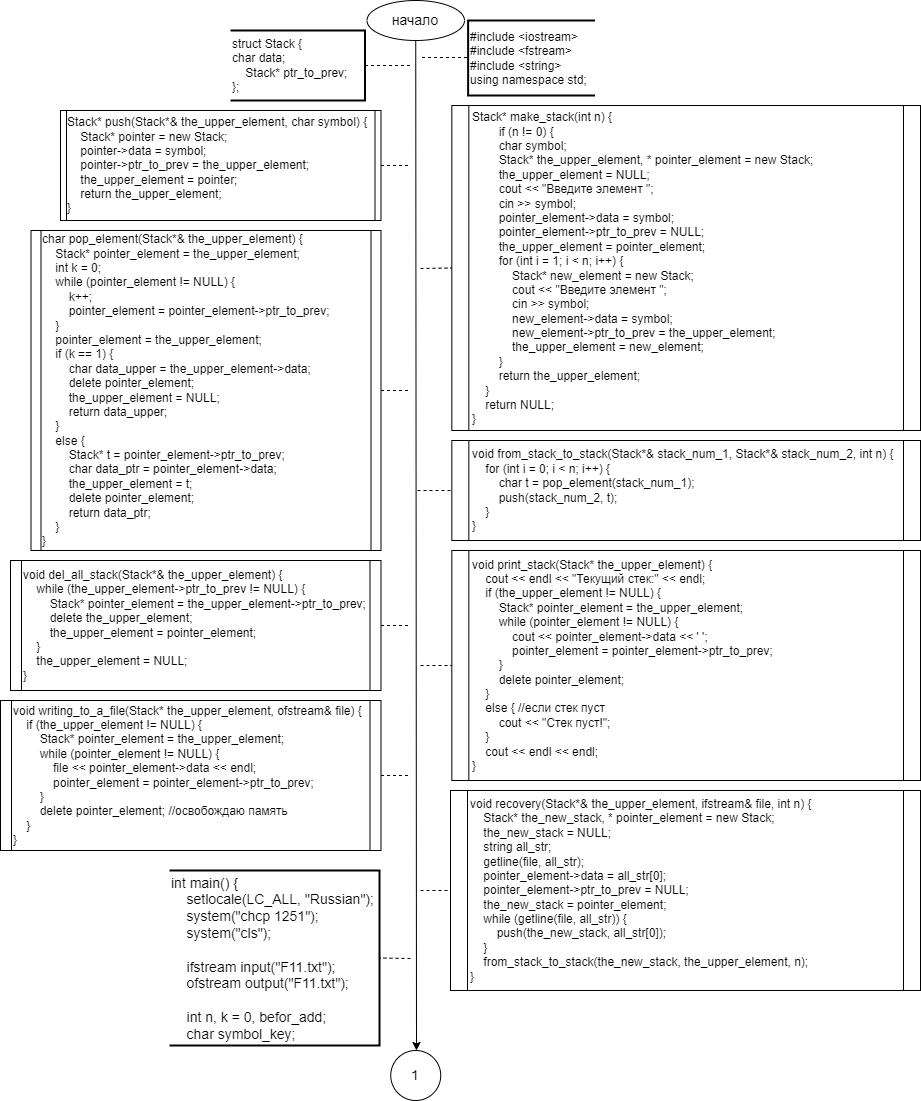


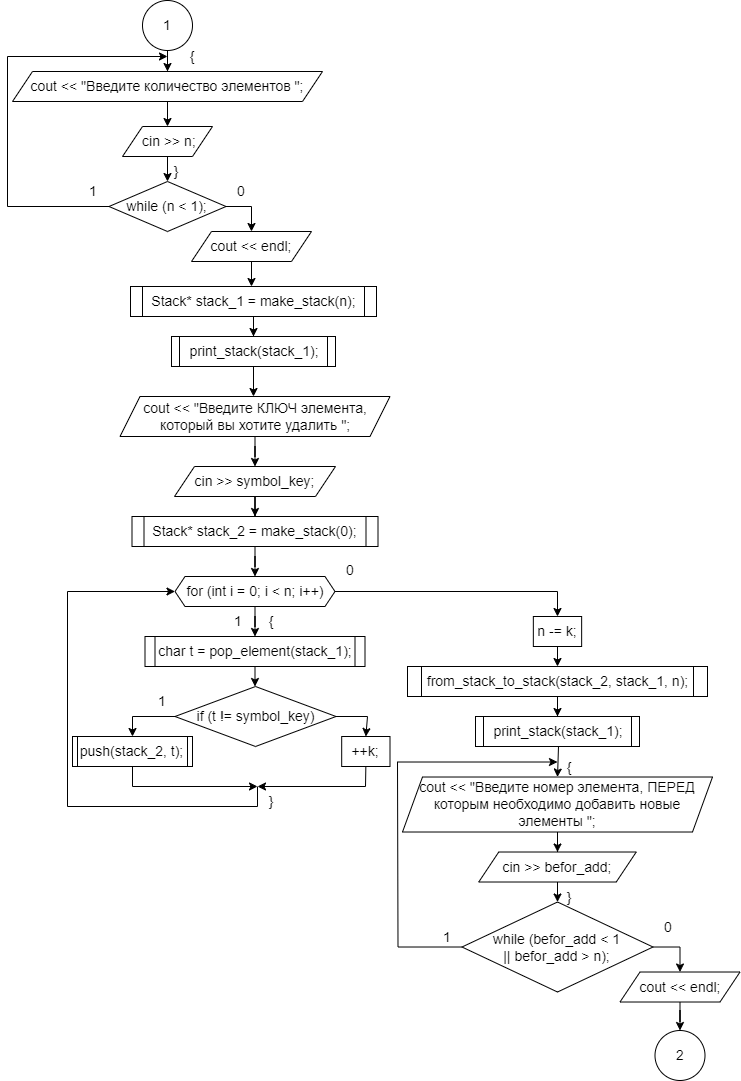
# **Стеки**

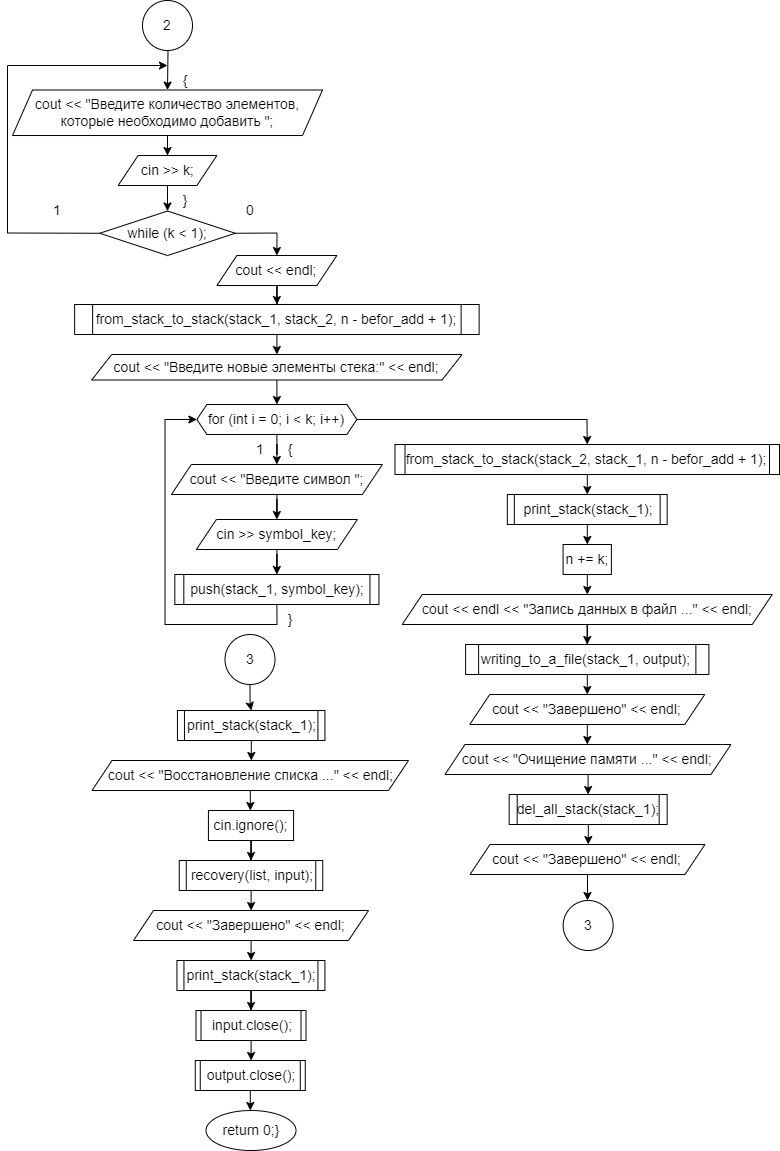
## Анализ задачи

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 К 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("F11.txt"); //входной файловый поток

ofstream output("F11.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 знаяение 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 знаяение 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

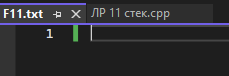
push(the\_new\_stack, all\_str[0]); //переношу элементы во временный стек

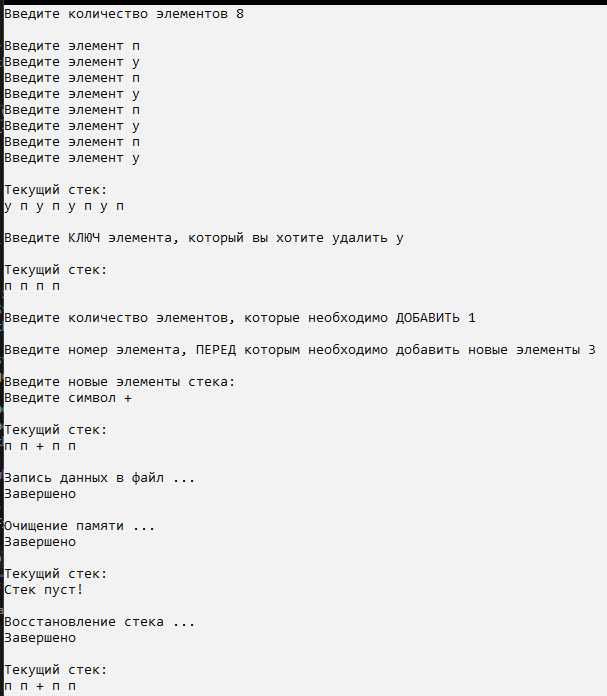
}

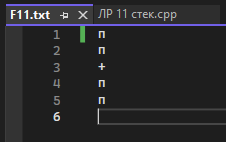
from\_stack\_to\_stack(the\_new\_stack, the\_upper\_element, n); //переношу элементы в главный стек

}

## Результат работы программы





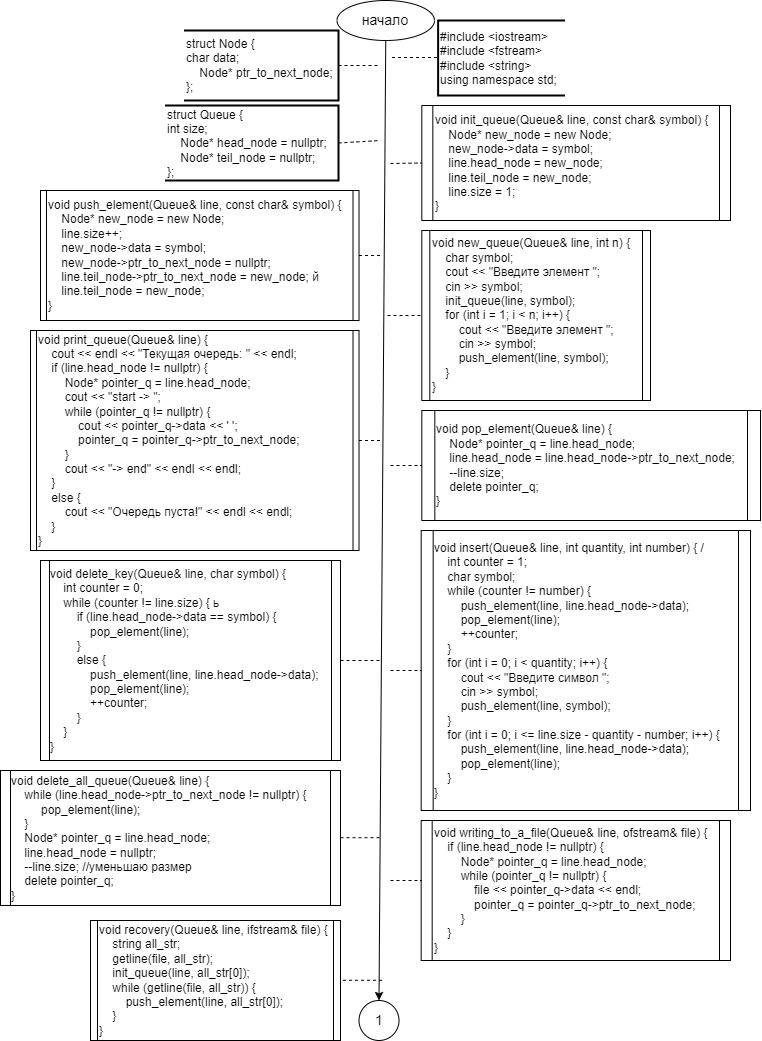


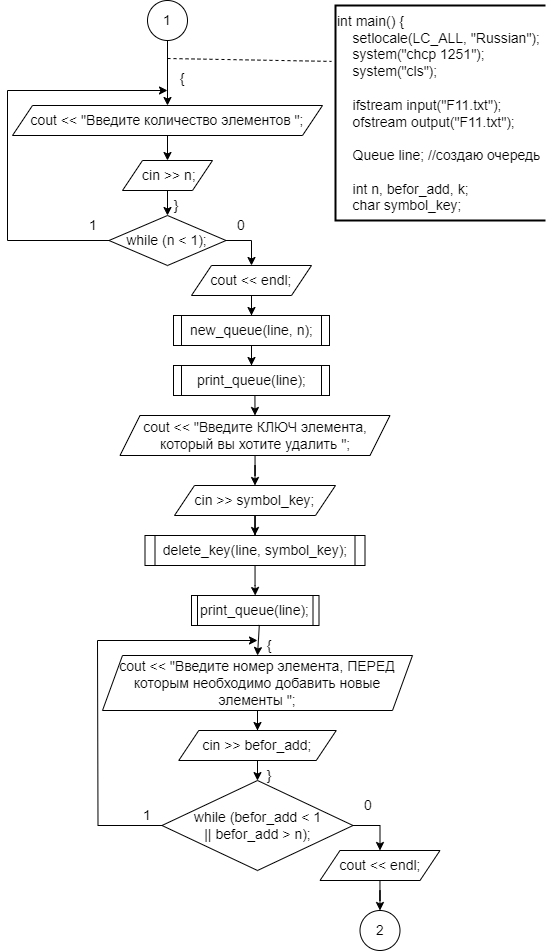
# **Очереди**

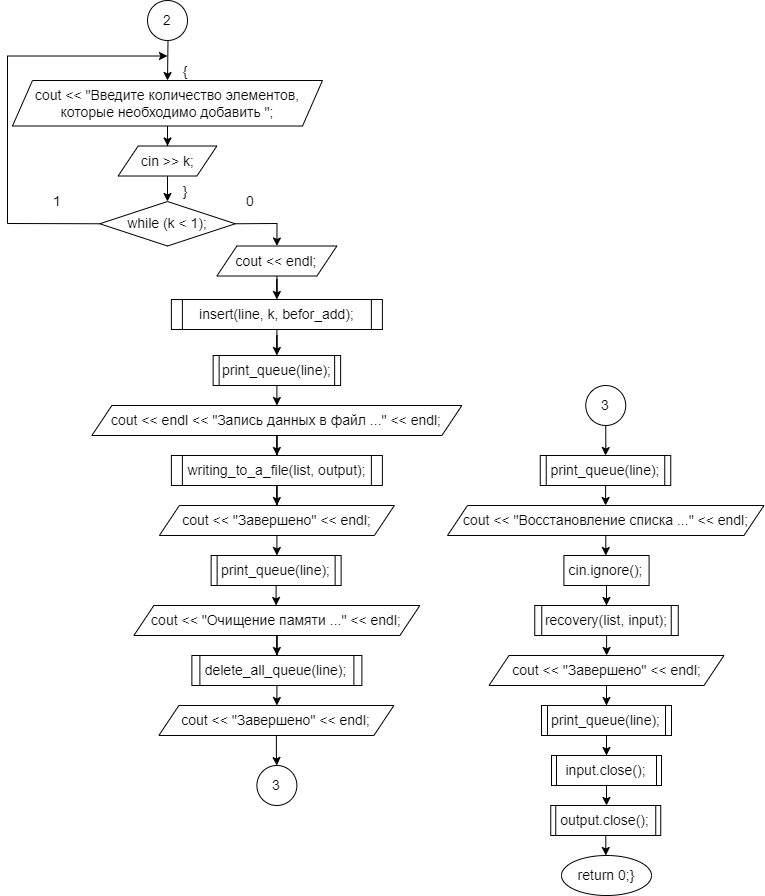
## Анализ задачи

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 recovery восстанавливает очередь 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\_next\_node; //указатель на следующий элемент

};

struct Queue { //очередь

int size; //размер

Node\* head\_node = nullptr;

Node\* teil\_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); //всавляю 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

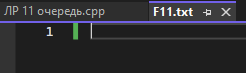
while (getline(file, all\_str)) { //пока не пройду фесь файл

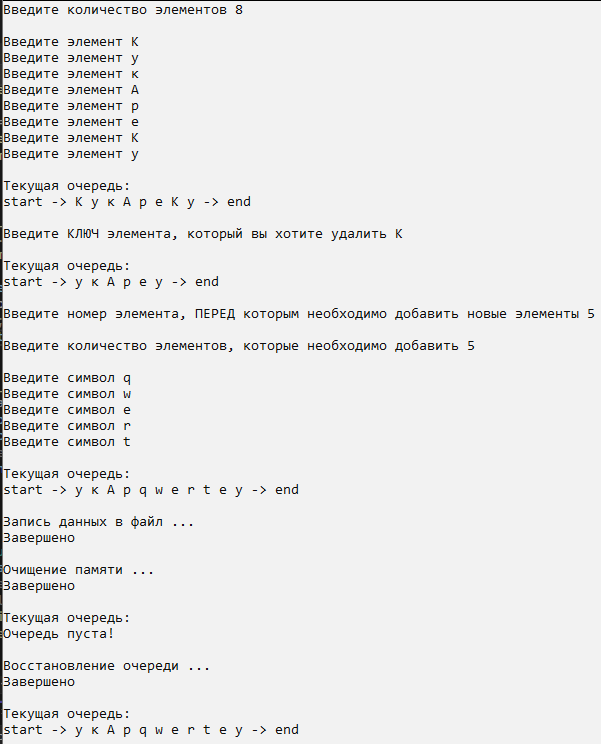
push\_element(line, all\_str[0]); //добавляю в очередь

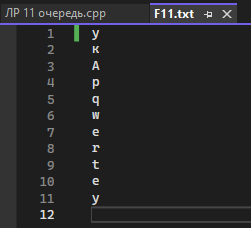
}

}

## Результат работы программы







# **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 структкры, 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>

