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

Лабораторная работа №11

«Информационные динамические структуры»

Вариант №25

Выполнил:

студент первого курса

ЭТФ группы РИС-23-3б

Коротаев Александр Дмитриевич

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.

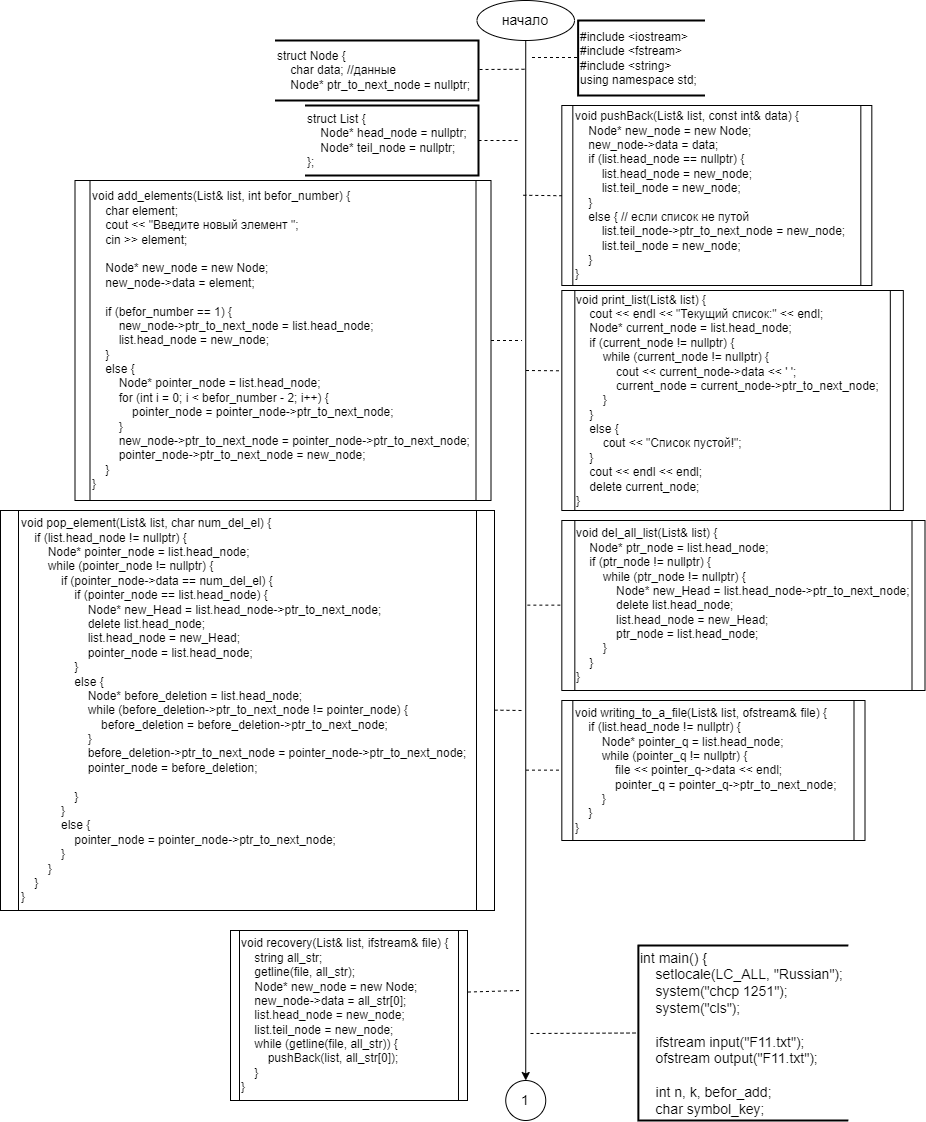
6. Уничтожение списка.

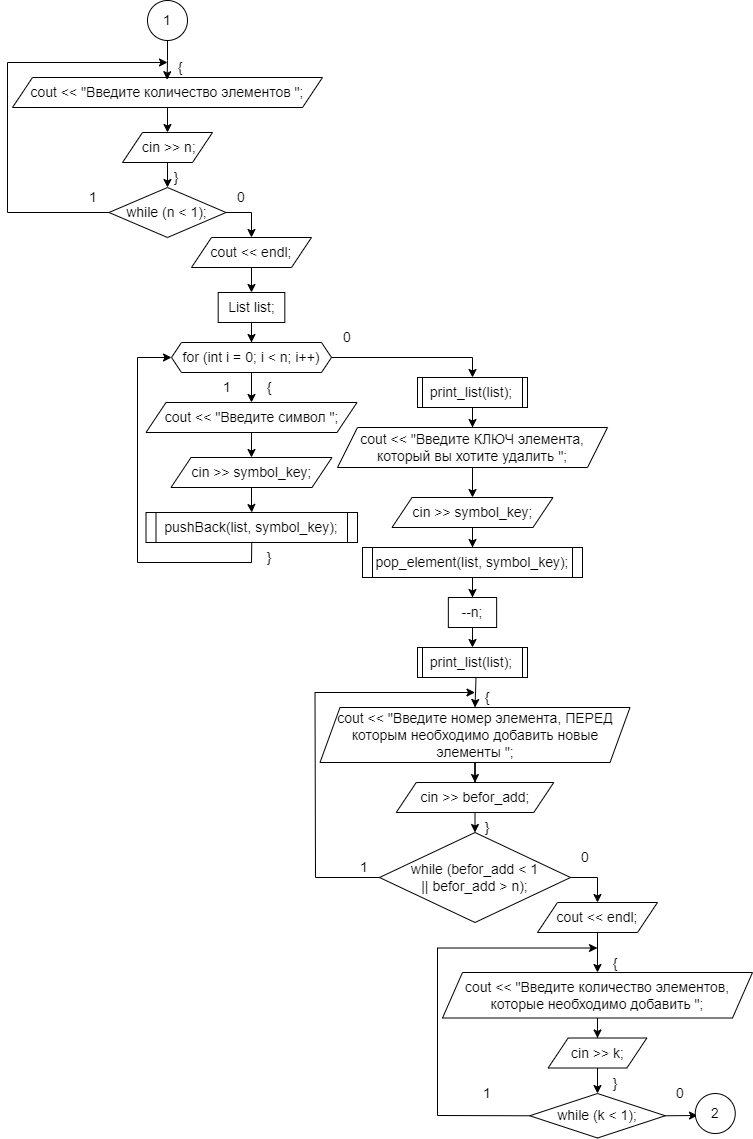
7. Восстановление списка из файла.

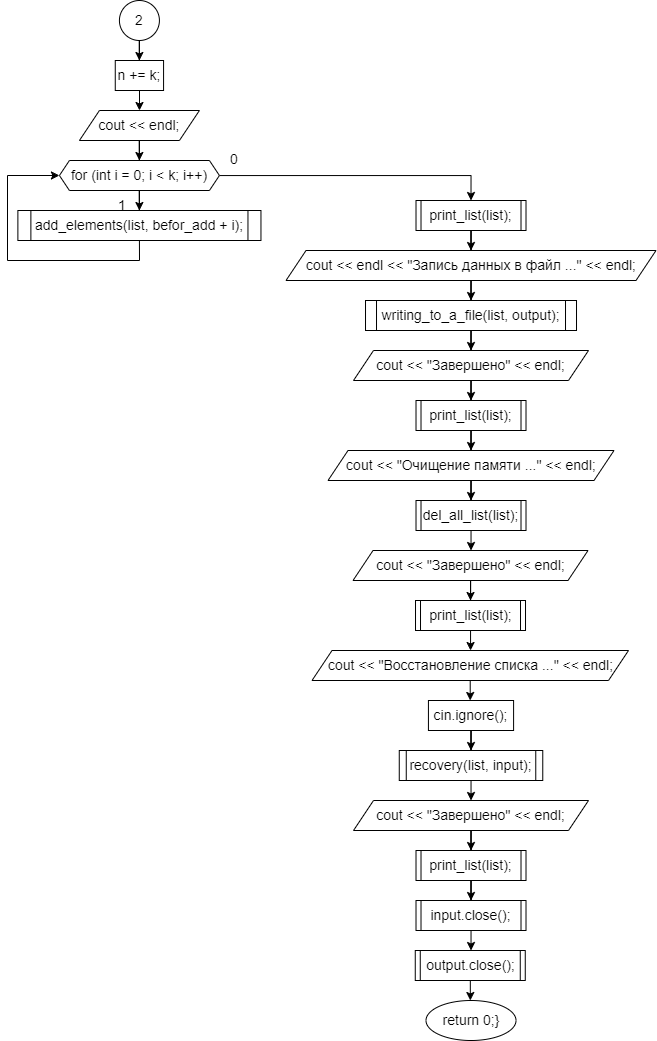
# **Однонаправленные списки**

## 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.







## Код программы

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

struct Node {

char data; //данные

Node\* ptr\_to\_next\_node = nullptr;

};

struct List {

Node\* head\_node = nullptr;

Node\* teil\_node = nullptr;

};

void pushBack(List& list, const int& data);

void add\_elements(List& list, int befor\_number);

void print\_list(List& list);

void pop\_element(List& list, char num\_del\_el);

void del\_all\_list(List& list);

void writing\_to\_a\_file(List& list, ofstream& file);

void recovery(List& list, ifstream& file);

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;

do {

cout << "Введите количество элементов ";

cin >> n;

} while (n < 1);

cout << endl;

List list;

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

cout << "Введите символ ";

cin >> symbol\_key;

pushBack(list, symbol\_key);

}

print\_list(list);

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

cin >> symbol\_key;

pop\_element(list, symbol\_key);

--n;

print\_list(list);

do {

cout << "Введите номер элемента, ПЕРЕД которым необходимо добавить новые элементы ";

cin >> befor\_add;

} while (befor\_add < 1 || befor\_add > n);

cout << endl;

do {

cout << "Введите количество элементов, которые необходимо добавить ";

cin >> k;

} while (k < 1);

n += k;

cout << endl;

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

add\_elements(list, befor\_add + i);

}

print\_list(list);

cout << endl << "Запись данных в файл ..." << endl;

writing\_to\_a\_file(list, output);

cout << "Завершено" << endl;

print\_list(list);

cout << "Очищение памяти ..." << endl;

del\_all\_list(list);

cout << "Завершено" << endl;

print\_list(list);

cout << "Восстановление списка ..." << endl;

cin.ignore();

recovery(list, input);

cout << "Завершено" << endl;

print\_list(list);

input.close();

output.close();

return 0;

}

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 {

list.teil\_node->ptr\_to\_next\_node = new\_node;

list.teil\_node = new\_node;

}

}

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 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 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 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;

}

}

}

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;

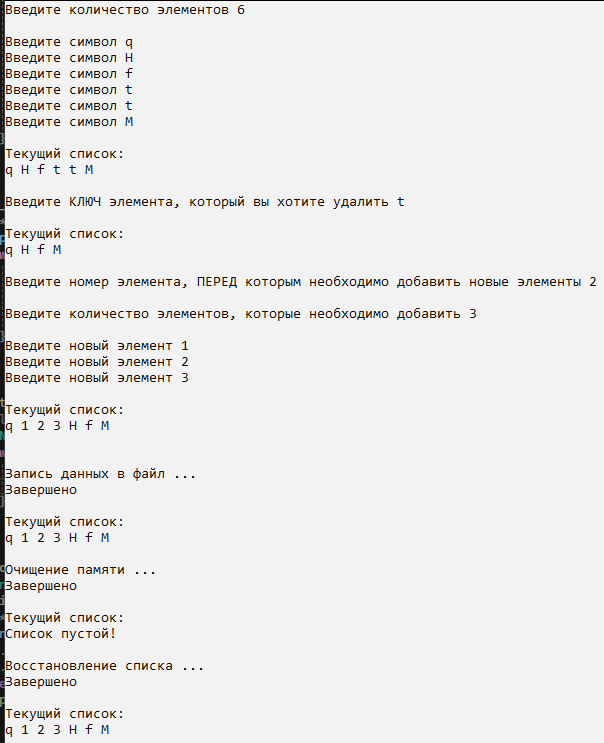
while (getline(file, all\_str)) {

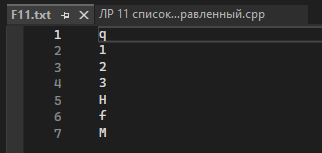
pushBack(list, all\_str[0]);

}

}

## Результат работы программы

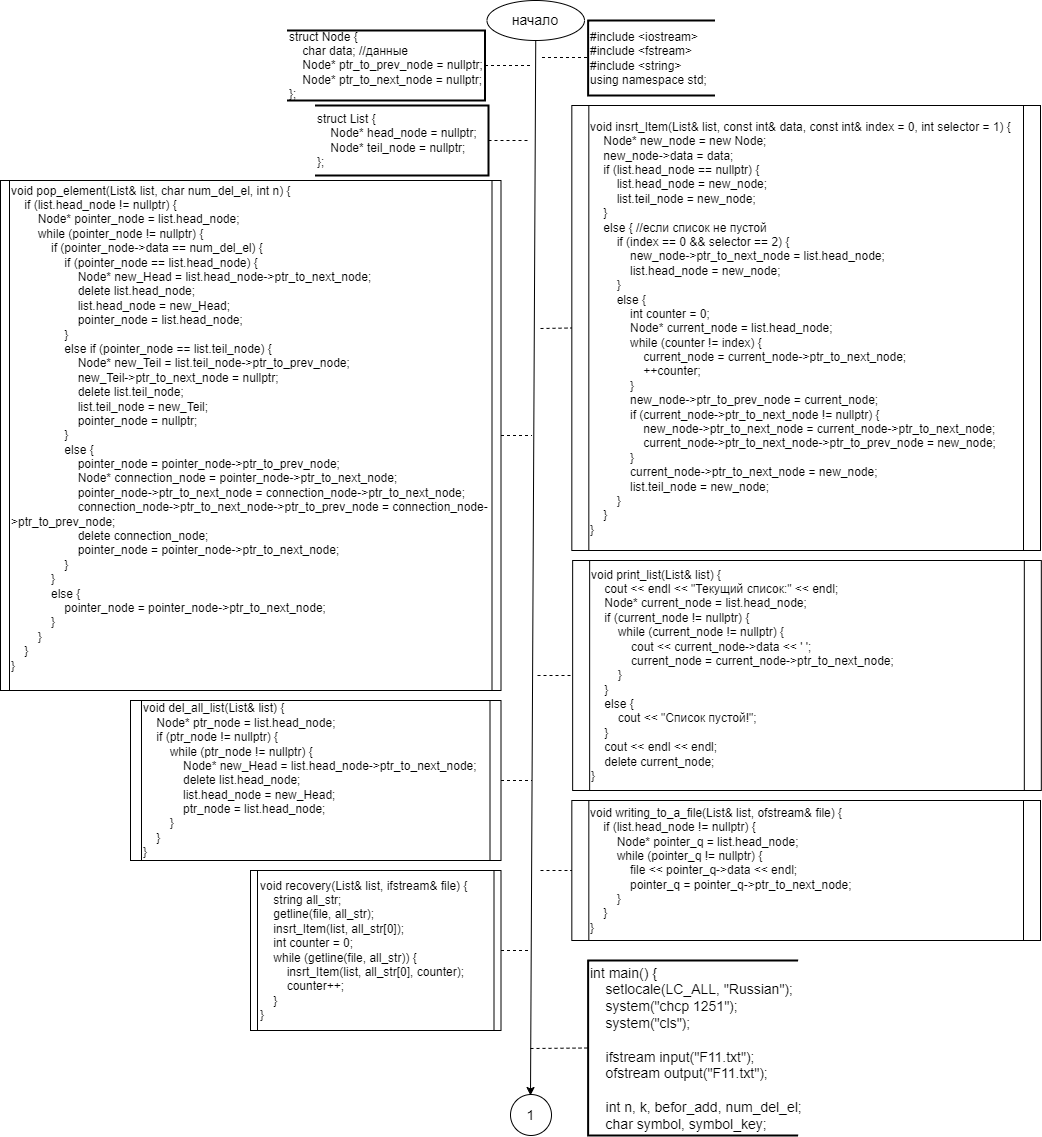


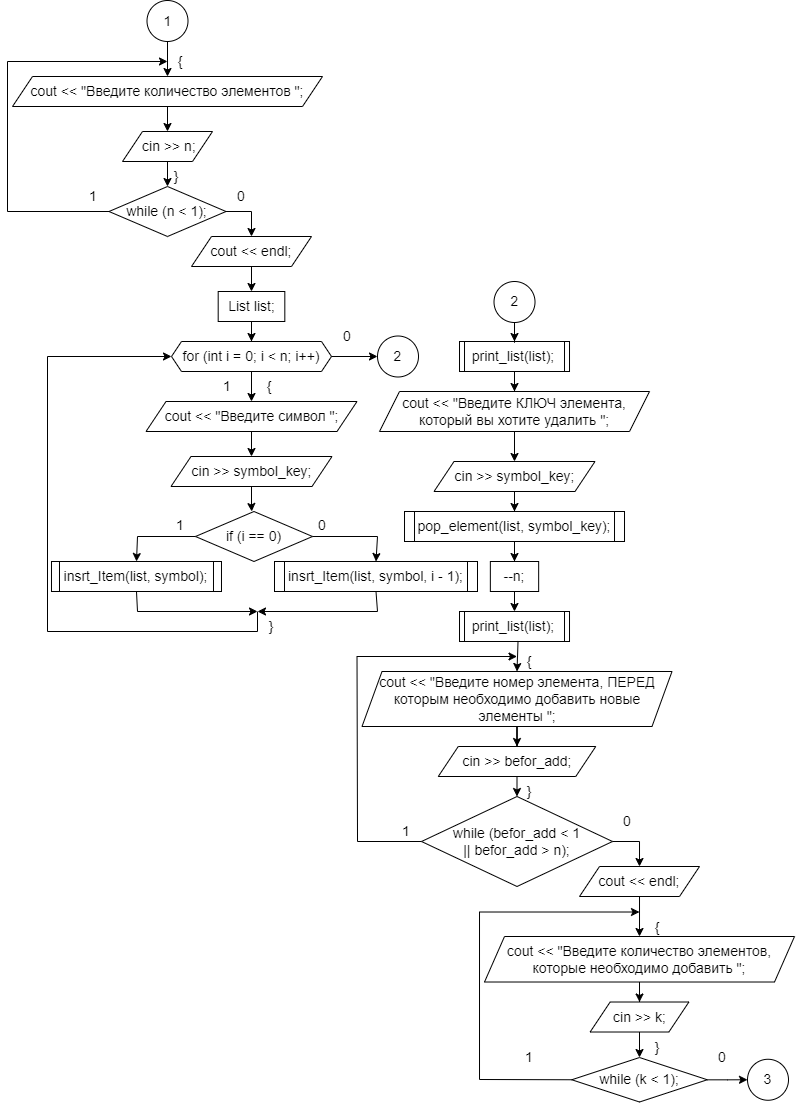


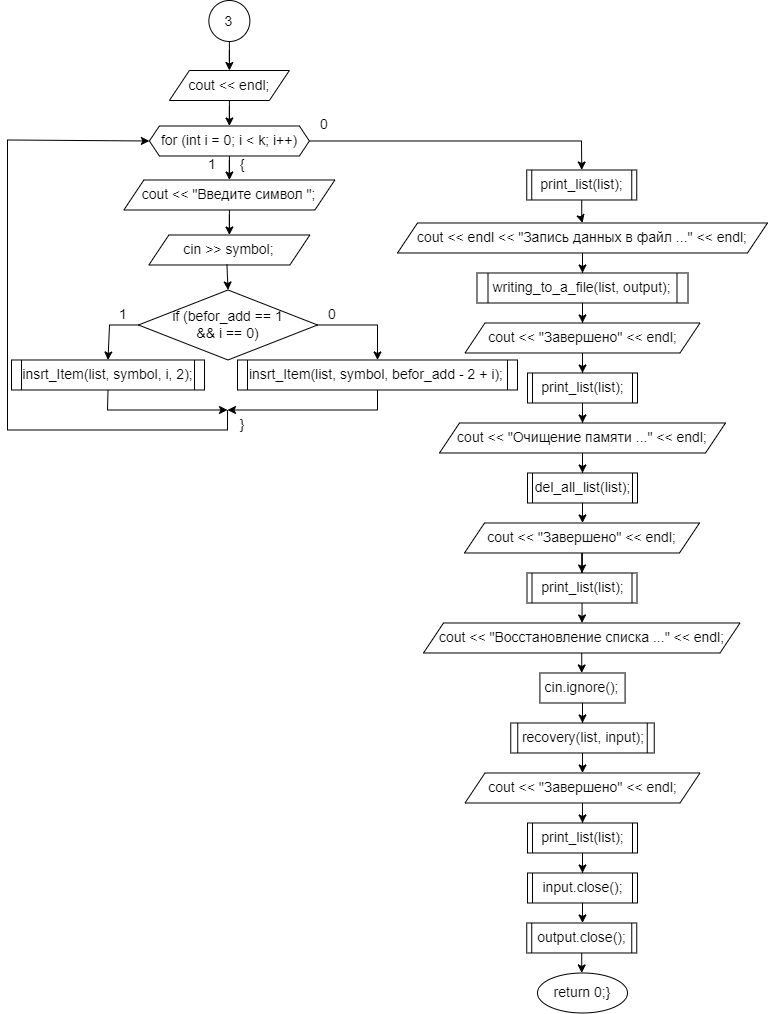
# **Двунаправленные списки**

## 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.







## Код программы

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

struct Node {

char data;

Node\* ptr\_to\_prev\_node = nullptr;

Node\* ptr\_to\_next\_node = nullptr;

};

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);

void pop\_element(List& list, char num\_del\_el, int n);

void print\_list(List& list);

void del\_all\_list(List& list);

void writing\_to\_a\_file(List& list, ofstream& file);

void recovery(List& list, ifstream& file);

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, symbol\_key;

do {

cout << "Введите количество элементов ";

cin >> n;

} while (n < 1);

cout << endl;

List list;

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

cout << "Введите символ ";

cin >> symbol;

if (i == 0) insrt\_Item(list, symbol);

else insrt\_Item(list, symbol, i - 1);

}

print\_list(list);

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

cin >> symbol\_key;

pop\_element(list, symbol\_key, n);

cout << "Выполнено удаление элемента(-ов) с заданным ключом" << endl;

--n;

print\_list(list);

do {

cout << "Введите номер элемента, ПЕРЕД которым необходимо добавить новые элементы ";

cin >> befor\_add;

} while (befor\_add < 1 || befor\_add > n);

cout << endl;

do {

cout << "Введите количество элементов, которые необходимо добавить ";

cin >> k;

} 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);

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);

cout << "Восстановление списка ..." << endl;

cin.ignore();

recovery(list, input);

cout << "Завершено" << endl;

print\_list(list);

input.close();

output.close();

return 0;

}

void insrt\_Item(List& list, const int& data, const int& index, int selector) {

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 {

if (index == 0 && selector == 2) {

new\_node->ptr\_to\_next\_node = list.head\_node;

list.head\_node = new\_node;

}

else {

int counter = 0;

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

while (counter != index) {

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;

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) {

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 if (pointer\_node == list.teil\_node) {

Node\* new\_Teil = list.teil\_node->ptr\_to\_prev\_node;

new\_Teil->ptr\_to\_next\_node = nullptr;

delete list.teil\_node;

list.teil\_node = new\_Teil;

pointer\_node = nullptr;

}

else {

pointer\_node = pointer\_node->ptr\_to\_prev\_node;

Node\* connection\_node = pointer\_node->ptr\_to\_next\_node;

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;

pointer\_node = pointer\_node->ptr\_to\_next\_node;

}

}

else {

pointer\_node = pointer\_node->ptr\_to\_next\_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;

}

}

}

void recovery(List& list, ifstream& file) {

string all\_str;

getline(file, all\_str);

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

int counter = 0;

while (getline(file, all\_str)) {

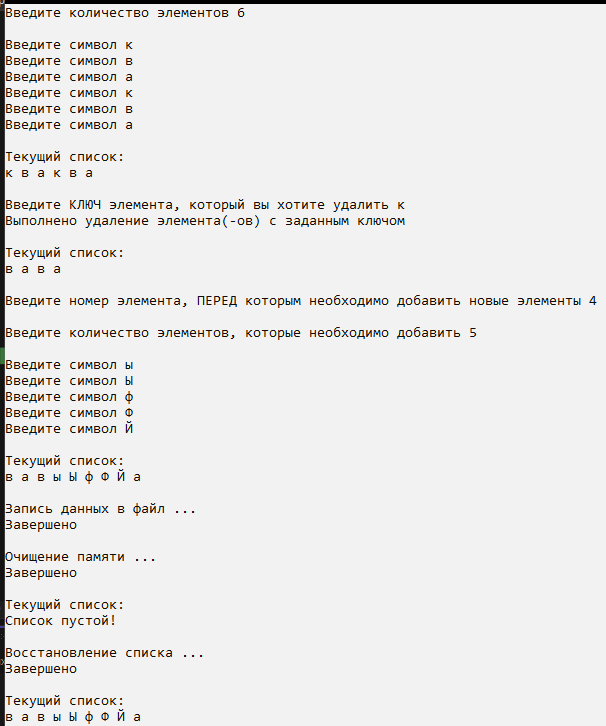
insrt\_Item(list, all\_str[0], counter);

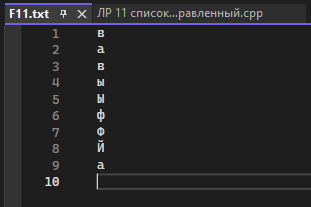
counter++;

}

}

## Результат работы программы

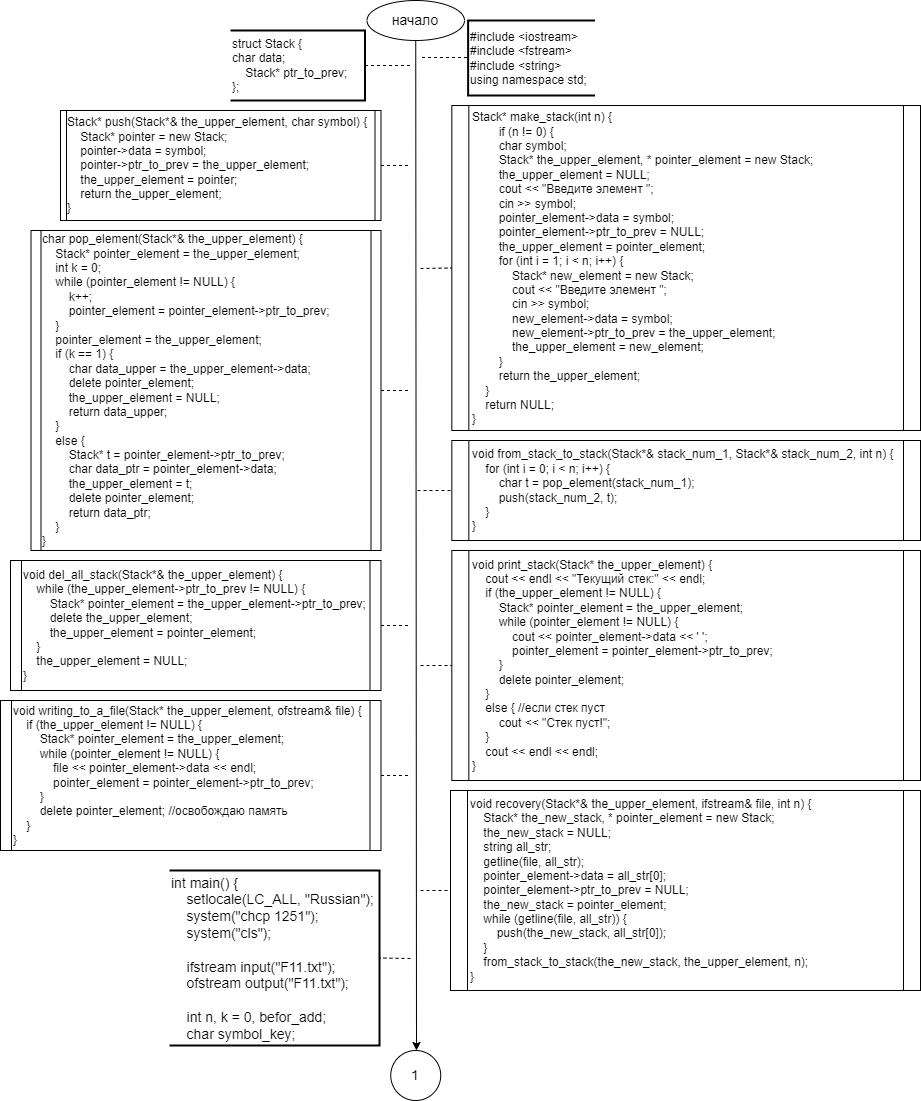


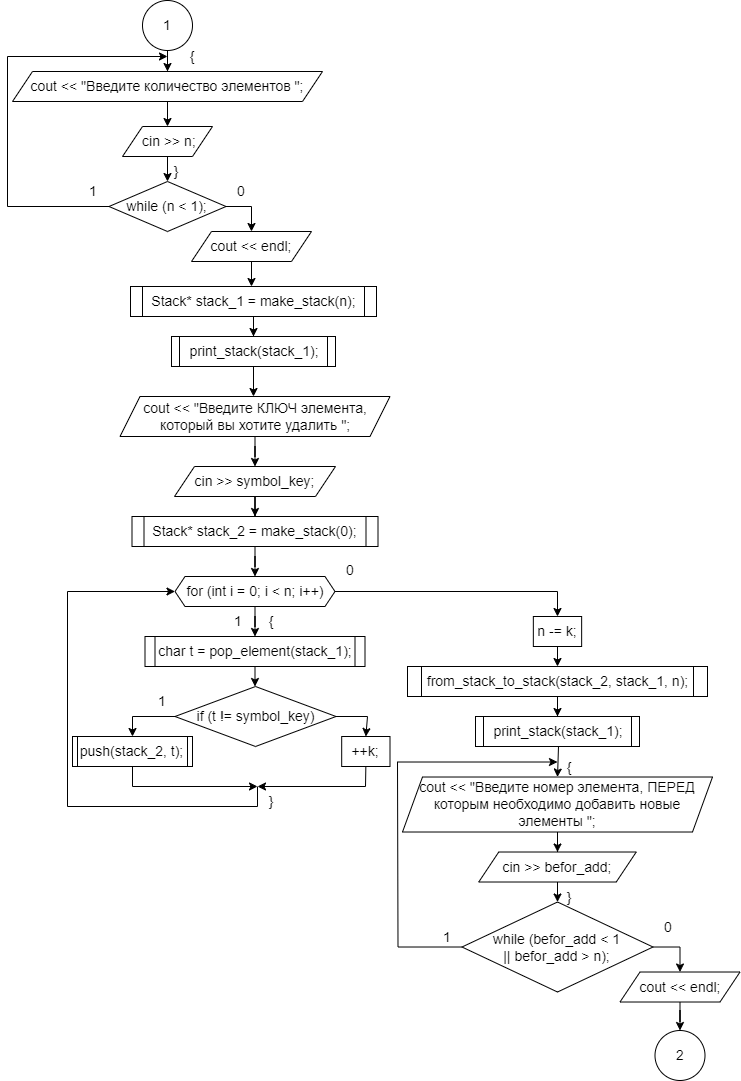


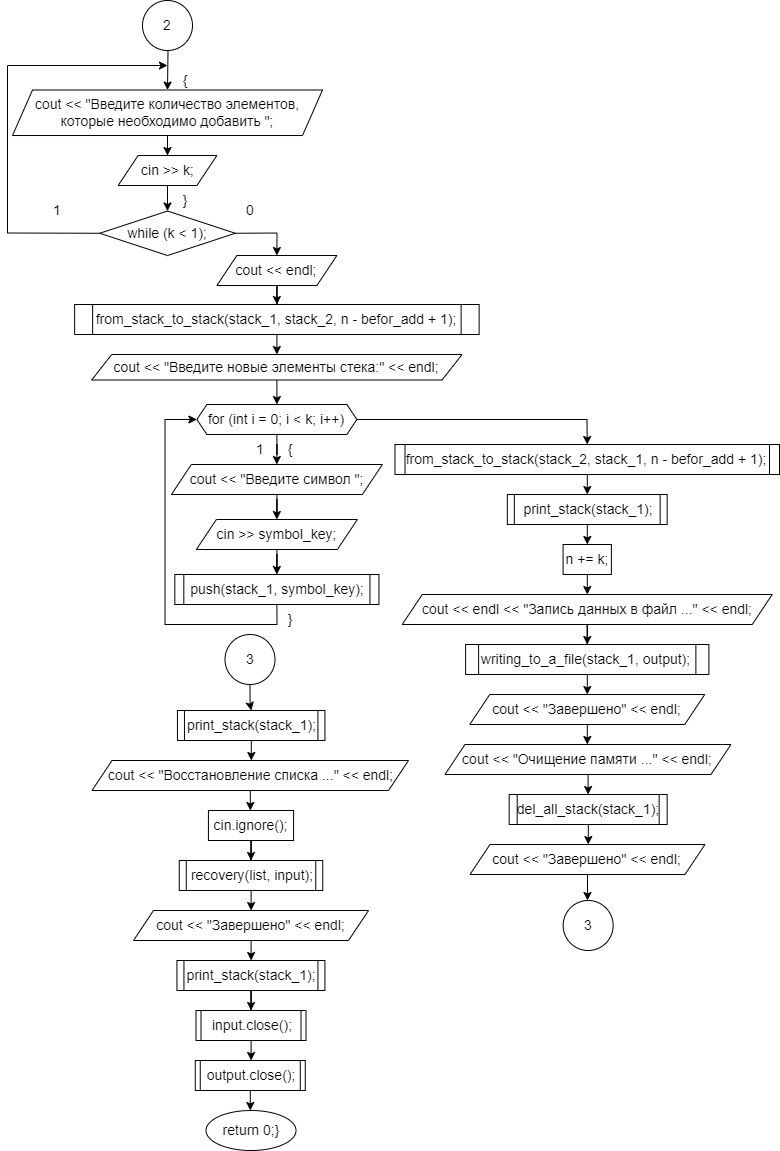
# **Стеки**

## 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.







## Код программы

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

struct Stack {

char data;

Stack\* ptr\_to\_prev;

};

Stack\* make\_stack(int n);

Stack\* push(Stack\*& the\_upper\_element, char symbol);

char pop\_element(Stack\*& the\_upper\_element);

void from\_stack\_to\_stack(Stack\*& stack\_num\_1, Stack\*& stack\_num\_2, int n);

void print\_stack(Stack\* the\_upper\_element);

void del\_all\_stack(Stack\*& the\_upper\_element);

void writing\_to\_a\_file(Stack\* the\_upper\_element, ofstream& file);

void recovery(Stack\*& the\_upper\_element, ifstream& file, int n);

int main() {

setlocale(LC\_ALL, "Russian");

system("chcp 1251");

system("cls");

ifstream input("F11.txt");

ofstream output("F11.txt");

int n, k = 0, befor\_add;

char symbol\_key;

do {

cout << "Введите количество элементов ";

cin >> n;

} while (n < 1);

cout << endl;

Stack\* stack\_1 = make\_stack(n);

print\_stack(stack\_1);

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

cin >> symbol\_key;

Stack\* stack\_2 = make\_stack(0);

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

char t = pop\_element(stack\_1);

if (t != symbol\_key) {

push(stack\_2, t);

}

else {

++k;

}

}

n -= k

from\_stack\_to\_stack(stack\_2, stack\_1, n);

print\_stack(stack\_1);

do {

cout << "Введите количество элементов, которые необходимо ДОБАВИТЬ ";

cin >> k;

} while (k < 1);

cout << endl;

do {

cout << "Введите номер элемента, ПЕРЕД которым необходимо добавить новые элементы ";

cin >> befor\_add;

} 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);

}

from\_stack\_to\_stack(stack\_2, stack\_1, n - befor\_add + 1);

print\_stack(stack\_1);

n += k;

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);

cout << "Завершено" << endl;

print\_stack(stack\_1);

cout << "Восстановление стека ..." << endl;

cin.ignore();

recovery(stack\_1, input, n);

cout << "Завершено" << endl;

print\_stack(stack\_1);

input.close();

output.close();

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;

pointer\_element->ptr\_to\_prev = NULL;

the\_upper\_element = pointer\_element;

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

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) {

Stack\* pointer = new Stack;

pointer->data = symbol;

pointer->ptr\_to\_prev = the\_upper\_element;

the\_upper\_element = pointer;

return the\_upper\_element;

}

char pop\_element(Stack\*& the\_upper\_element) {

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;

delete pointer\_element;

the\_upper\_element = NULL;

return data\_upper;

}

else {

Stack\* t = pointer\_element->ptr\_to\_prev;

char data\_ptr = pointer\_element->data;

the\_upper\_element = t;

delete pointer\_element;

return data\_ptr;

}

}

void from\_stack\_to\_stack(Stack\*& stack\_num\_1, Stack\*& stack\_num\_2, int n) {

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

char t = pop\_element(stack\_num\_1);

push(stack\_num\_2, t);

}

}

void print\_stack(Stack\* the\_upper\_element) {

cout << endl << "Текущий стек:" << endl;

if (the\_upper\_element != NULL) {

Stack\* pointer\_element = the\_upper\_element;

while (pointer\_element != NULL) {

cout << pointer\_element->data << ' ';

pointer\_element = pointer\_element->ptr\_to\_prev;

}

delete pointer\_element;

}

else {

cout << "Стек пуст!";

}

cout << endl << endl;

}

void del\_all\_stack(Stack\*& the\_upper\_element) {

while (the\_upper\_element->ptr\_to\_prev != NULL) {

Stack\* pointer\_element = the\_upper\_element->ptr\_to\_prev;

delete the\_upper\_element;

the\_upper\_element = pointer\_element;

}

the\_upper\_element = NULL;

}

void writing\_to\_a\_file(Stack\* the\_upper\_element, ofstream& file) {

if (the\_upper\_element != NULL) {

Stack\* pointer\_element = the\_upper\_element;

while (pointer\_element != NULL) {

file << pointer\_element->data << endl;

pointer\_element = pointer\_element->ptr\_to\_prev;

}

delete pointer\_element;

}

}

void recovery(Stack\*& the\_upper\_element, ifstream& file, int n) {

Stack\* the\_new\_stack, \* pointer\_element = new Stack;

the\_new\_stack = NULL;

string all\_str;

getline(file, all\_str);

pointer\_element->data = all\_str[0];

pointer\_element->ptr\_to\_prev = NULL;

the\_new\_stack = pointer\_element;

while (getline(file, all\_str)) {

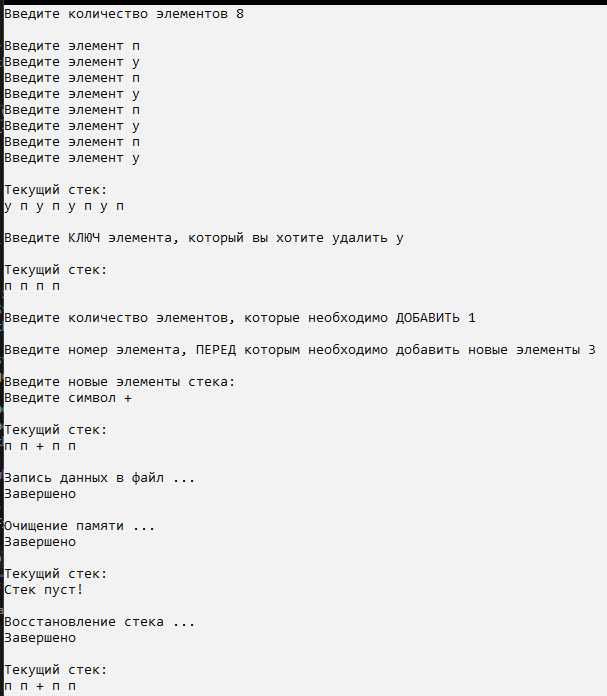
push(the\_new\_stack, all\_str[0]);

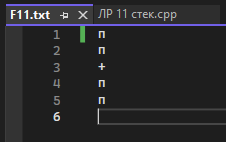
}

from\_stack\_to\_stack(the\_new\_stack, the\_upper\_element, n);

}

## Результат работы программы

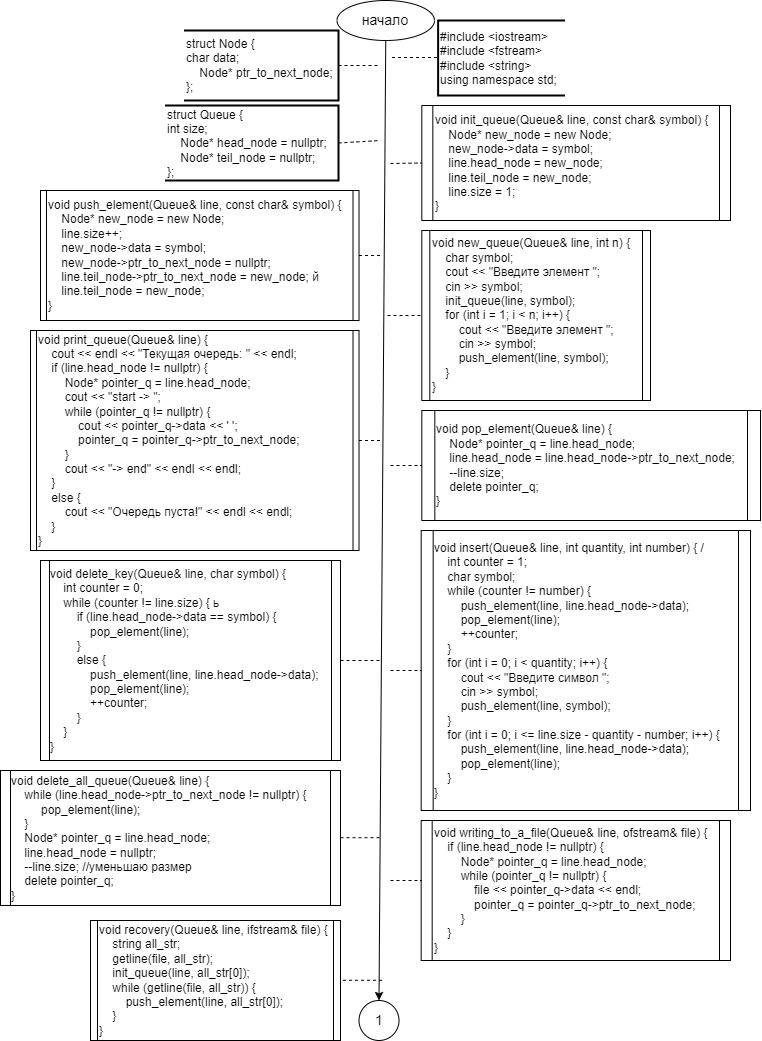


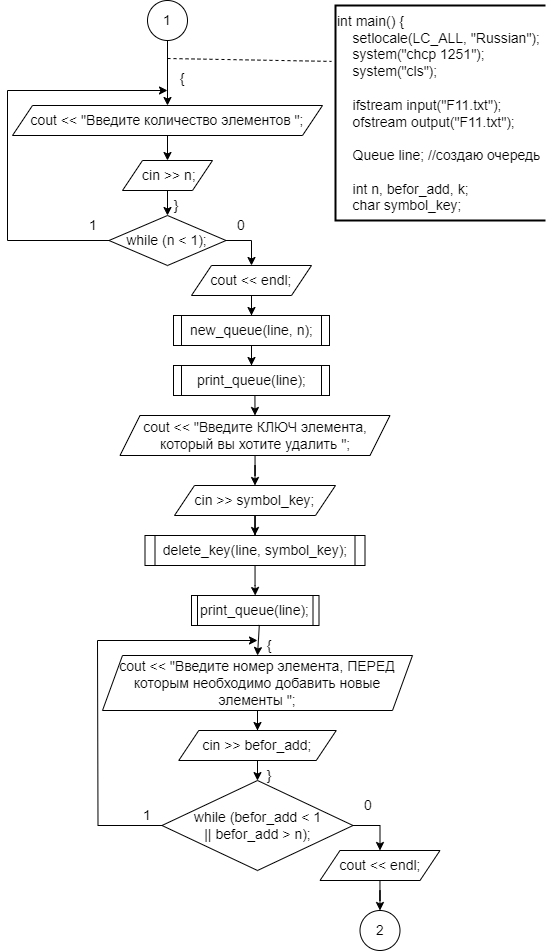


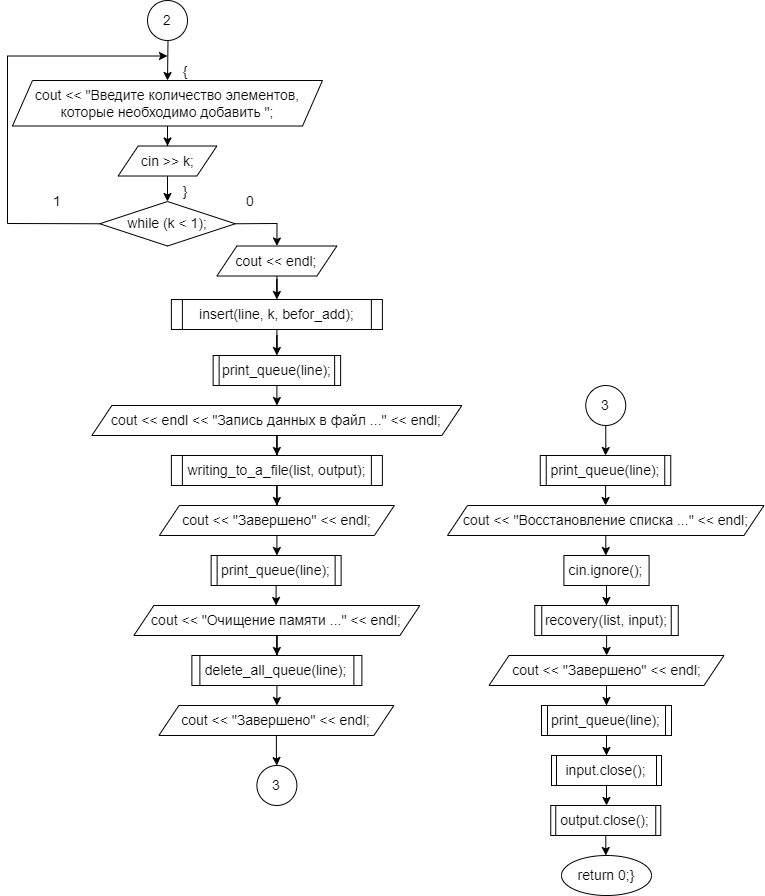
# **Очереди**

## 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.







## Код программы

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

struct Node {

char data;

Node\* ptr\_to\_next\_node;

};

struct Queue {

int size;

Node\* head\_node = nullptr;

Node\* teil\_node = nullptr;

};

void init\_queue(Queue& line, const char& symbol);

void push\_element(Queue& line, const char& symbol);

void new\_queue(Queue& line, int n);

void print\_queue(Queue& line);

void pop\_element(Queue& line);

void delete\_key(Queue& line, char symbol);

void insert(Queue& line, int quantity, int number);

void delete\_all\_queue(Queue& line);

void writing\_to\_a\_file(Queue& line, ofstream& file);

void recovery(Queue& line, ifstream& file);

int main() {

setlocale(LC\_ALL, "Russian");

system("chcp 1251");

system("cls");

ifstream input("F11.txt");

ofstream output("F11.txt");

Queue line;

int n, befor\_add, k;

char symbol\_key;

do {

cout << "Введите количество элементов ";

cin >> n;

} while (n < 1);

cout << endl;

new\_queue(line, n);

print\_queue(line);

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

cin >> symbol\_key;

delete\_key(line, symbol\_key);

print\_queue(line);

do {

cout << "Введите номер элемента, ПЕРЕД которым необходимо добавить новые элементы ";

cin >> befor\_add;

} while (befor\_add < 1 || befor\_add > line.size);

cout << endl;

do {

cout << "Введите количество элементов, которые необходимо добавить ";

cin >> k;

} while (k < 1);

cout << endl;

insert(line, k, befor\_add);

print\_queue(line);

cout << "Запись данных в файл ..." << endl;

writing\_to\_a\_file(line, output);

cout << "Завершено" << endl << endl;

cout << "Очищение памяти ..." << endl;

delete\_all\_queue(line);

cout << "Завершено" << endl;

print\_queue(line);

cout << "Восстановление очереди ..." << endl;

cin.ignore();

recovery(line, input);

cout << "Завершено" << endl;

print\_queue(line);

input.close();

output.close();

return 0;

}

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 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; //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;

}

cout << "-> end" << endl << endl;

}

else {

cout << "Очередь пуста!" << endl << endl;

}

}

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 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 insert(Queue& line, int quantity, int number) {

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 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;

delete pointer\_q;

}

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;

}

}

}

void recovery(Queue& line, ifstream& file) {

string all\_str;

getline(file, all\_str);

init\_queue(line, all\_str[0]);

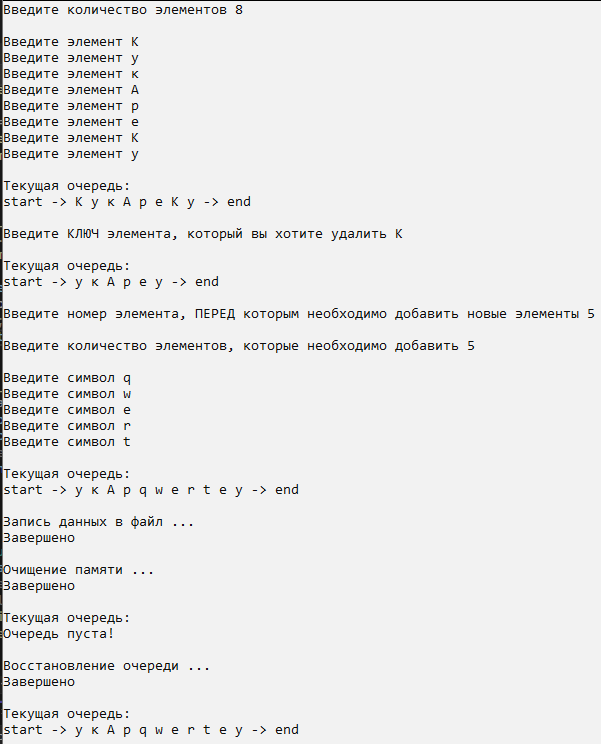
while (getline(file, all\_str)) {

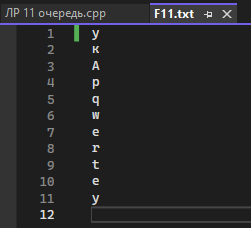
push\_element(line, all\_str[0]);

}

}

## Результат работы программы





**GitHub**

[https://github.com/Korovay4ik/Laboratory-works](https://github.com/Korovay4ik/Laboratory-works%20)