

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение  
высшего образования  
«Пермский национальный исследовательский политехнический университет»  
(ПНИПУ)  
Электротехнический факультет  
Кафедра «Информационные технологии и автоматизированные системы»  
(ИТАС)

## **Лабораторная работа**

на тему

### **«Информационные динамические структуры»**

Выполнил

Студент группы ИВТ-23-16

Адаев Даниил Дмитриевич

Проверил

Доцент кафедры ИТАС

Д. В. Яруллин

г. Пермь, 2024

## Постановка задачи

Написать программу, в которой создаются динамические структуры и выполнить их обработку в соответствии со своим вариантом.

Для каждого вариант 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.

### Порядок выполнения работы

1. Написать функцию для создания списка. Функция может создавать пустой список, а затем добавлять в него элементы.
2. Написать функцию для печати списка. Функция должна предусматривать вывод сообщения, если список пустой.
3. Написать функции для удаления и добавления элементов списка в соответствии со своим вариантом.
4. Выполнить изменения в списке и печать списка после каждого изменения.
5. Написать функцию для записи списка в файл.
6. Написать функцию для уничтожения списка.
7. Записать список в файл, уничтожить его и выполнить печать (при печати должно быть выдано сообщение "Список пустой").
8. Написать функцию для восстановления списка из файла.
9. Восстановить список и распечатать его.
10. Уничтожить список.

### Вариант 2

Записи в линейном списке содержат ключевое поле типа `int`.  
Сформировать однонаправленный список. Удалить из него элемент с заданным ключом, добавить элемент перед элементом с заданным ключом;

# Однонаправленный список

## Словесный алгоритм

Директива `_CRR_SECURE_NO_WARNINGS` дает доступ к старым функциям.

Структура `elem` задает элемент списка, функция `unidir` создает однонаправленный список, забитый случайными числами и возвращает указатель на последний элемент. Функция `print_list` выводит список с последнего до первого элемента или объявляет, что список пуст.

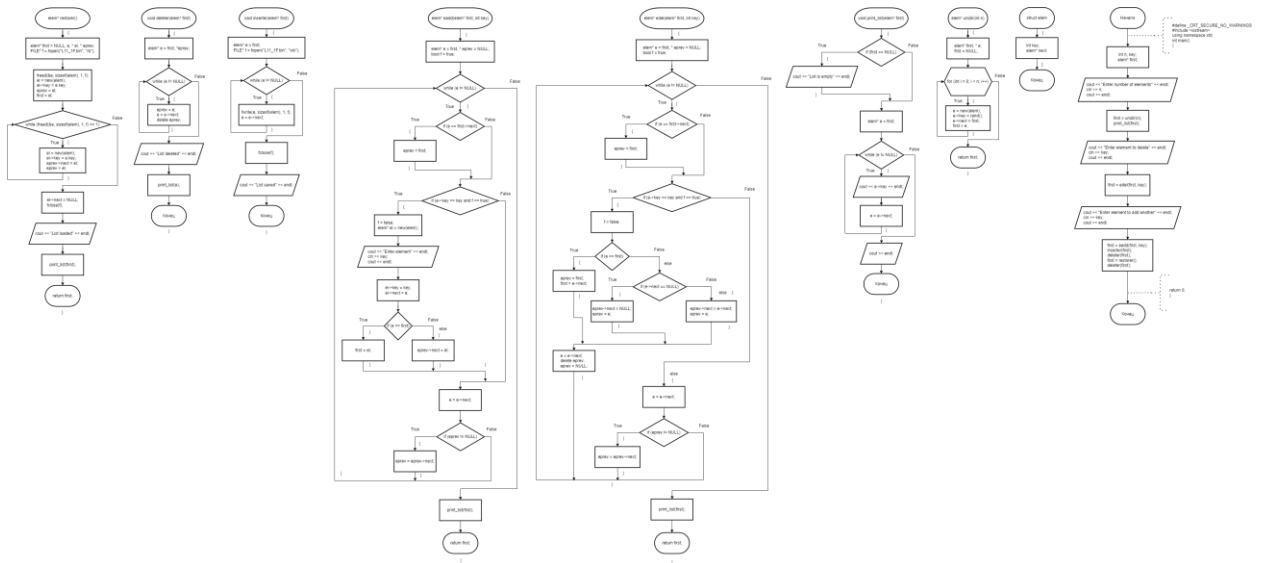
Функция `edel` соединяет элементы между удаляемым элементом и удаляет последний, затем выводит получившийся список функцией `print_list` и возвращает указатель на его начало.

Функция `eadd` добавляет элемент перед элементом с заданным значением, выводит получившийся список функцией `print_list` и возвращает указатель на его начало.

Функция `inserter` записывает элементы списка в файл и объявляет, что список сохранен. Функция `deleter` один за другим удаляет элементы списка, объявляет об его удалении и вызывает функцию `print_list`, которая объявляет, что список пуст. Функция `restorer` считывает элементы из файла и создает из них однонаправленный список, объявляет об его создании, выводит функцией `print_list` и возвращает указатель на последний элемент.

Главная функция объявляет целочисленные переменные `n` и `key` для размера списка и вводимого значения соответственно, объявляет указатель на начало списка `first`. Алгоритм запрашивает ввести количество элементов. Оно используется в функции `unidir`, создающей список, на который указывает `first`. Список выводится. Запрашивается значение элемента для удаления функцией `edel`, затем значение для добавления элемента через `eadd`. Список сохраняется, удаляется, считывается из файла и снова удаляется функциями `inserter`, `deleter`, и `restorer` соответственно.

## Блок-схема



## Код программы

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>

using namespace std;

struct elem
{
    int key;
    elem* next;
};

elem* unidir(int n)
{
    elem* first, * e;
    first = NULL;

    for (int i = 0; i < n; i++)
    {
        e = new(elem);
        e->key = rand();
        e->next = first;
        first = e;
    }

    return first;
}

void print_list(elem* first)
{
    if (first == NULL)
    {
        cout << "List is empty" << endl;
    }

    elem* e = first;

    while (e != NULL)
    {
        cout << e->key << endl;
    }
}
```

```

        e = e->next;
    }

    cout << endl;
}

elem* edel(elem* first, int key)
{
    elem* e = first, * eprev = NULL;
    bool f = true;

    while (e != NULL)
    {
        if (e == first->next)
        {
            eprev = first;
        }

        if (e->key == key and f == true)
        {
            f = false;

            if (e == first)
            {
                eprev = first;
                first = e->next;
            }
            else if (e->next == NULL)
            {
                eprev->next = NULL;
                eprev = e;
            }
            else
            {
                eprev->next = e->next;
                eprev = e;
            }

            e = e->next;
            delete eprev;
            eprev = NULL;
        }
        else
        {
            e = e->next;

            if (eprev != NULL)
            {
                eprev = eprev->next;
            }
        }
    }

    print_list(first);

    return first;
}

elem* eadd(elem* first, int key)
{
    elem* e = first, * eprev = NULL;
    bool f = true;

    while (e != NULL)
    {

```

```

        if (e == first->next)
        {
            eprev = first;
        }

        if (e->key == key and f == true)
        {
            f = false;
            elem* el = new(elem);

            cout << "Enter element" << endl;
            cin >> key;
            cout << endl;

            el->key = key;
            el->next = e;

            if (e == first)
            {
                first = el;
            }
            else
            {
                eprev->next = el;
            }
        }

        e = e->next;

        if (eprev != NULL)
        {
            eprev = eprev->next;
        }
    }

    print_list(first);

    return first;
}

void inserter(elem* first)
{
    elem* e = first;
    FILE* f = fopen("L11_1F.bin", "wb");

    while (e != NULL)
    {
        fwrite(e, sizeof(elem), 1, f);
        e = e->next;
    }

    fclose(f);

    cout << "List saved" << endl;
}

void deleter(elem* first)
{
    elem* e = first, *eprev;

    while (e != NULL)
    {
        eprev = e;
        e = e->next;
        delete eprev;
    }
}

```

```

    }

    cout << "List deleted" << endl;

    print_list(e);
}

elem* restorer()
{
    elem* first = NULL, e, * el, * eprev;
    FILE* f = fopen("L11_1F.bin", "rb");

    fread(&e, sizeof(elem), 1, f);
    el = new(elem);
    el->key = e.key;
    eprev = el;
    first = el;

    while (fread(&e, sizeof(elem), 1, f) == 1)
    {
        el = new(elem);
        el->key = e.key;
        eprev->next = el;
        eprev = el;
    }

    el->next = NULL;
    fclose(f);

    cout << "List loaded" << endl;

    print_list(first);

    return first;
}

int main()
{
    int n, key;
    elem* first;

    cout << "Enter number of elements" << endl;
    cin >> n;
    cout << endl;

    first = unidir(n);
    print_list(first);

    cout << "Enter element to delete" << endl;
    cin >> key;
    cout << endl;

    first = edel(first, key);

    cout << "Enter element to add another" << endl;
    cin >> key;
    cout << endl;

    first = eadd(first, key);
    inserter(first);
    deleter(first);
    first = restorer();
    deleter(first);

    return 0;
}

```

## } Выводы

```
Enter number of elements
5

19169
26500
6334
18467
41

Enter element to delete
6334

19169
26500
18467
41

Enter element to add another
41

Enter element
-1

19169
26500
18467
-1
41

List saved
List deleted
List is empty

List loaded
19169
26500
18467
-1
41

List deleted
List is empty
```

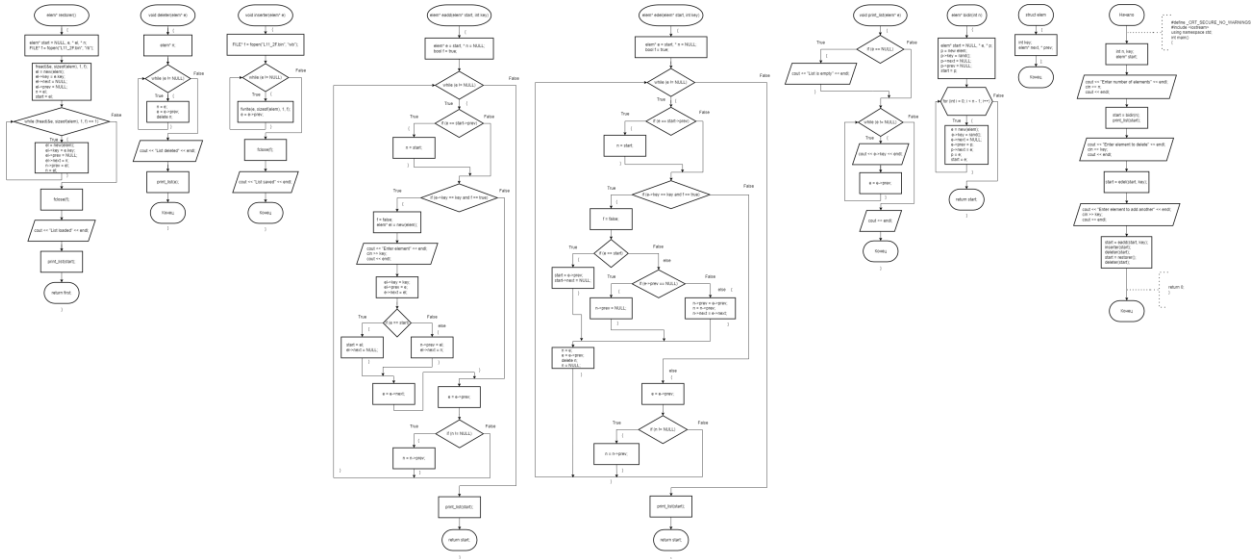


# Двунаправленный список

## Словесный алгоритм

Структура алгоритма аналогична заданию с однонаправленным списком, за исключением дополнительного указателя для каждого элемента и связанных с ним операций.

## Блок-схема



## Код программы

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>

using namespace std;

struct elem
{
    int key;
    elem* next, * prev;
};

elem* bidir(int n)
{
    elem* start = NULL, * e, * p;
    p = new elem;
    p->key = rand();
    p->next = NULL;
    p->prev = NULL;
    start = p;

    for (int i = 0; i < n - 1; i++)
    {
        e = new(elem);
        e->key = rand();
        e->next = NULL;
        e->prev = p;
        p->next = e;
        p = e;
        start = e;
    }
}
```

```

        return start;
    }

void print_list(elem* e)
{
    if (e == NULL)
    {
        cout << "List is empty" << endl;
    }

    while (e != NULL)
    {
        cout << e->key << endl;

        e = e->prev;
    }

    cout << endl;
}

elem* edel(elem* start, int key)
{
    elem* e = start, * n = NULL;
    bool f = true;

    while (e != NULL)
    {
        if (e == start->prev)
        {
            n = start;
        }

        if (e->key == key and f == true)
        {
            f = false;

            if (e == start)
            {
                start = e->prev;
                start->next = NULL;
            }
            else if (e->prev == NULL)
            {
                n->prev = NULL;
            }
            else
            {
                n->prev = e->prev;
                n = n->prev;
                n->next = e->next;
            }

            n = e;
            e = e->prev;
            delete n;
            n = NULL;
        }
        else
        {
            e = e->prev;

            if (n != NULL)
            {
                n = n->prev;
            }
        }
    }
}

```

```

        }
    }
    print_list(start);
    return start;
}

elem* eadd(elem* start, int key)
{
    elem* e = start, * n = NULL;
    bool f = true;

    while (e != NULL)
    {
        if (e == start->prev)
        {
            n = start;
        }

        if (e->key == key and f == true)
        {
            f = false;
            elem* el = new(elem);

            cout << "Enter element" << endl;
            cin >> key;
            cout << endl;

            el->key = key;
            el->prev = e;
            e->next = el;

            if (e == start)
            {
                start = el;
                el->next = NULL;
            }
            else
            {
                n->prev = el;
                el->next = n;
            }

            e = e->next;
        }

        e = e->prev;

        if (n != NULL)
        {
            n = n->prev;
        }
    }

    print_list(start);

    return start;
}

void inserter(elem* e)
{
    FILE* f = fopen("L11_2F.bin", "wb");

```

```

        while (e != NULL)
        {
            fwrite(e, sizeof(elem), 1, f);
            e = e->prev;
        }

        fclose(f);

        cout << "List saved" << endl;
    }

void deleter(elem* e)
{
    elem* n;

    while (e != NULL)
    {
        n = e;
        e = e->prev;
        delete n;
    }

    cout << "List deleted" << endl;

    print_list(e);
}

elem* restorer()
{
    elem* start = NULL, e, * el, * n;
    FILE* f = fopen("L11_2F.bin", "rb");

    fread(&e, sizeof(elem), 1, f);
    el = new(elem);
    el->key = e.key;
    el->next = NULL;
    el->prev = NULL;
    n = el;
    start = el;

    while (fread(&e, sizeof(elem), 1, f) == 1)
    {
        el = new(elem);
        el->key = e.key;
        el->prev = NULL;
        el->next = n;
        n->prev = el;
        n = el;
    }

    fclose(f);

    cout << "List loaded" << endl;

    print_list(start);

    return start;
}

int main()
{
    int n, key;
    elem* start;

```

```

    cout << "Enter number of elements" << endl;
    cin >> n;
    cout << endl;

    start = bidir(n);
    print_list(start);

    cout << "Enter element to delete" << endl;
    cin >> key;
    cout << endl;

    start = edel(start, key);

    cout << "Enter element to add another" << endl;
    cin >> key;
    cout << endl;

    start = eadd(start, key);
    inserter(start);
    deleter(start);
    start = restorer();
    deleter(start);

    return 0;
}

```

## Выводы

```
Enter number of elements
5

19169
26500
6334
18467
41

Enter element to delete
6334

19169
26500
18467
41

Enter element to add another
41

Enter element
-1

19169
26500
18467
-1
41

List saved
List deleted
List is empty

List loaded
19169
26500
18467
-1
41

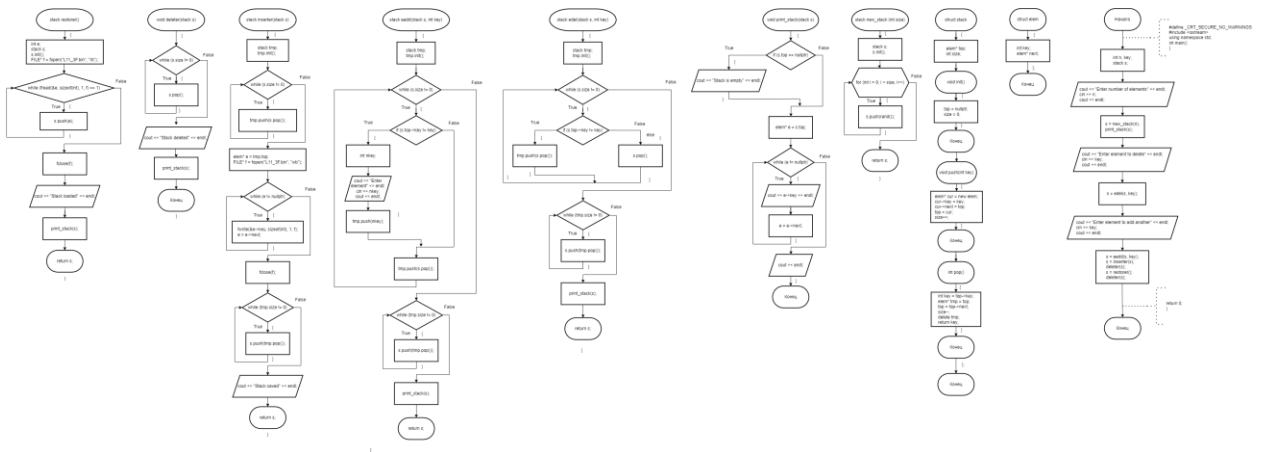
List deleted
List is empty
```

# Стек

## Словесный алгоритм

Структура алгоритма аналогична предыдущим, за исключением структуры стека с присущими функциями

## Блок-схема



## Код программы

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>

using namespace std;

struct elem
{
    int key;
    elem* next;
};

struct stack
{
    elem* top;
    int size;

    void init()
    {
        top = nullptr;
        size = 0;
    }

    void push(int key)
    {
        elem* cur = new elem;
        cur->key = key;
        cur->next = top;
        top = cur;
        size++;
    }
}
```

```

    int pop()
    {
        int key = top->key;
        elem* tmp = top;
        top = top->next;
        size--;
        delete tmp;
        return key;
    }
};

stack new_stack (int size)
{
    stack s;
    s.init();

    for (int i = 0; i < size; i++)
    {
        s.push(rand());
    }

    return s;
}

void print_stack(stack s)
{
    if (s.top == nullptr)
    {
        cout << "Stack is empty" << endl;
    }

    elem* e = s.top;
    while (e != nullptr)
    {
        cout << e->key << endl;

        e = e->next;
    }

    cout << endl;
}

stack edel(stack s, int key)
{
    stack tmp;
    tmp.init();

    while (s.size != 0)
    {
        if (s.top->key != key)
        {
            tmp.push(s.pop());
        }
        else
        {
            s.pop();
        }
    }

    while (tmp.size != 0)
    {
        s.push(tmp.pop());
    }
}

```



```

    }

    print_stack(s);

    return s;
}

stack eadd(stack s, int key)
{
    stack tmp;
    tmp.init();

    while (s.size != 0)
    {
        if (s.top->key == key)
        {
            int nkey;

            cout << "Enter element" << endl;
            cin >> nkey;
            cout << endl;

            tmp.push(nkey);
        }
        tmp.push(s.pop());
    }

    while (tmp.size != 0)
    {
        s.push(tmp.pop());
    }

    print_stack(s);

    return s;
}

stack inserter(stack s)
{
    stack tmp;
    tmp.init();

    while (s.size != 0)
    {
        tmp.push(s.pop());
    }

    elem* e = tmp.top;
    FILE* f = fopen("L11_3F.bin", "wb");

    while (e != nullptr)
    {
        fwrite(&e->key, sizeof(int), 1, f);
        e = e->next;
    }

    fclose(f);

    while (tmp.size != 0)
    {
        s.push(tmp.pop());
    }

    cout << "Stack saved" << endl;
}

```

```

        return s;
    }

    void deleter(stack s)
    {
        while (s.size != 0)
        {
            s.pop();
        }

        cout << "Stack deleted" << endl;

        print_stack(s);
    }

    stack restorer()
    {
        int e;
        stack s;
        s.init();
        FILE* f = fopen("L11_3F.bin", "rb");

        while (fread(&e, sizeof(int), 1, f) == 1)
        {
            s.push(e);
        }

        fclose(f);

        cout << "Stack loaded" << endl;

        print_stack(s);

        return s;
    }

    int main()
    {
        int n, key;
        stack s;

        cout << "Enter number of elements" << endl;
        cin >> n;
        cout << endl;

        s = new_stack(n);
        print_stack(s);

        cout << "Enter element to delete" << endl;
        cin >> key;
        cout << endl;

        s = edel(s, key);

        cout << "Enter element to add another" << endl;
        cin >> key;
        cout << endl;

        s = eadd(s, key);
        s = inserter(s);
        deleter(s);
        s = restorer();
        deleter(s);
    }

```

```
    return 0;  
}
```

## Выводы

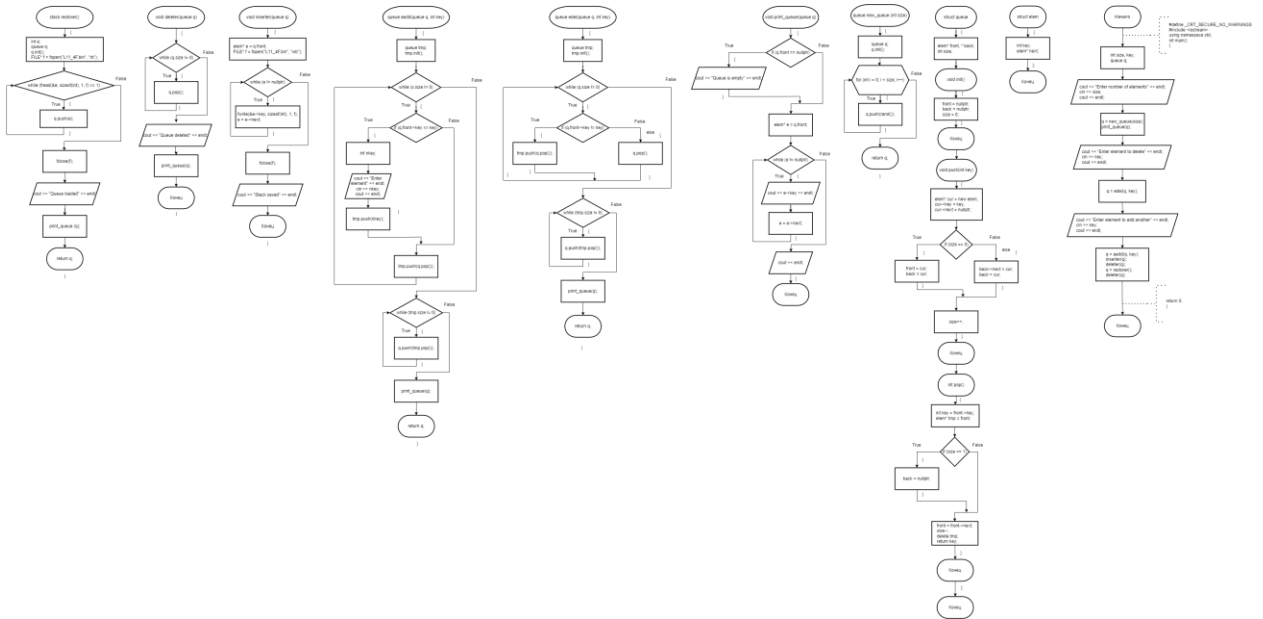
```
Enter number of elements  
5  
  
19169  
26500  
6334  
18467  
41  
  
Enter element to delete  
41  
  
19169  
26500  
6334  
18467  
  
Enter element to add another  
6334  
  
Enter element  
-1  
  
19169  
26500  
-1  
6334  
18467  
  
Stack saved  
Stack deleted  
Stack is empty  
  
Stack loaded  
19169  
26500  
-1  
6334  
18467  
  
Stack deleted  
Stack is empty
```

# Очередь

## Словесный алгоритм

Структура алгоритма аналогична предыдущим, за исключением структуры очереди с присущими функциями.

## Блок-схема



## Код программы

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>

using namespace std;

struct elem
{
    int key;
    elem* next;
};

struct queue
{
    elem* front, * back;
    int size;

    void init()
    {
        front = nullptr;
        back = nullptr;
        size = 0;
    }

    void push(int key)
    {
        elem* cur = new elem;
        cur->key = key;
```

```

        cur->next = nullptr;

        if (size == 0)
        {
            front = cur;
            back = cur;
        }
        else
        {
            back->next = cur;
            back = cur;
        }

        size++;
    }

    int pop()
    {
        int key = front->key;
        elem* tmp = front;

        if (size == 1)
        {
            back = nullptr;
        }

        front = front->next;
        size--;
        delete tmp;
        return key;
    }
};

queue new_queue (int size)
{
    queue q;
    q.init();

    for (int i = 0; i < size; i++)
    {
        q.push(rand());
    }

    return q;
}

void print_queue(queue q)
{
    if (q.front == nullptr)
    {
        cout << "Queue is empty" << endl;
    }

    elem* e = q.front;

    while (e != nullptr)
    {
        cout << e->key << endl;

        e = e->next;
    }

    cout << endl;
}

```

```

queue edel(queue q, int key)
{
    queue tmp;
    tmp.init();

    while (q.size != 0)
    {
        if (q.front->key != key)
        {
            tmp.push(q.pop());
        }
        else
        {
            q.pop();
        }
    }

    while (tmp.size != 0)
    {
        q.push(tmp.pop());
    }

    print_queue(q);

    return q;
}

queue eadd(queue q, int key)
{
    queue tmp;
    tmp.init();

    while (q.size != 0)
    {
        if (q.front->key == key)
        {
            int nkey;

            cout << "Enter element" << endl;
            cin >> nkey;
            cout << endl;

            tmp.push(nkey);
        }
        tmp.push(q.pop());
    }

    while (tmp.size != 0)
    {
        q.push(tmp.pop());
    }

    print_queue(q);

    return q;
}

void inserter(queue q)
{
    elem* e = q.front;
    FILE* f = fopen("L11_4F.bin", "wb");

    while (e != nullptr)
    {

```

```

        fwrite(&e->key, sizeof(int), 1, f);
        e = e->next;
    }

    fclose(f);

    cout << "Queue saved" << endl;
}

void deleter(queue q)
{
    while (q.size != 0)
    {
        q.pop();
    }

    cout << "Queue deleted" << endl;

    print_queue(q);
}

queue restorer()
{
    int e;
    queue q;
    q.init();
    FILE* f = fopen("L11_4F.bin", "rb");

    while (fread(&e, sizeof(int), 1, f) == 1)
    {
        q.push(e);
    }

    fclose(f);

    cout << "Queue loaded" << endl;

    print_queue (q);

    return q;
}

int main()
{
    int size, key;
    queue q;

    cout << "Enter number of elements" << endl;
    cin >> size;
    cout << endl;

    q = new_queue(size);
    print_queue(q);

    cout << "Enter element to delete" << endl;
    cin >> key;
    cout << endl;

    q = edel(q, key);

    cout << "Enter element to add another" << endl;
    cin >> key;
    cout << endl;
}

```

```
    q = eadd(q, key);  
    inserter(q);  
    deleter(q);  
    q = restorer();  
    deleter(q);  
  
    return 0;  
}
```



## Выводы

```
Enter number of elements
5

41
18467
6334
26500
19169

Enter element to delete
6334

41
18467
26500
19169

Enter element to add another
41

Enter element
-1

-1
41
18467
26500
19169

Queue saved
Queue deleted
Queue is empty

Queue loaded
-1
41
18467
26500
19169

Queue deleted
Queue is empty
```