ТЕМА7.
ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ.
Очереди

7.1. Работа со однонаправленной очередью

Пример 7.1. Работа с однонаправленной очередью.

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

struct TNode {
  int inf;    // Информационная часть структуры
  TNode* a;   // Адресная часть структуры
};
```

```
struct queue {
Tnode *front = nullptr; // Указатель на начало  очереди
Tnode *back = nullptr;  // Указатель на конец очереди


// Проверка наличия элементов в очереди
    bool empty() {
        if (front) return false;
        else return true;
    }
```

```
// Добавление элемента в очередь
    void push(int inf) {
        TNode* spt = new TNode;
        spt->inf = inf;
        spt->a = nullptr;
          // Если элементы отсутствуют
        if (!front) front = back = spt;
        else { // иначе
            back->a = spt;
            back = spt;
        }
    }
```

```
//  Удаление элемента из очереди
    void pop() {
        TNode* spt = front;
        front = front->a;
        delete spt;
        if (!front) back = nullptr;
    }
```

```
// Вывод очереди на экран
    void print() {
        TNode* spt = front;
        while (spt != nullptr) {
            cout << spt->inf << " ";
            spt = spt->a;
        }
    }
};
```

```
int main() {
    queue s;
s.push(4); s.push(2); s.push(1); s.push(6); s.push(9);
        s.print(); // Выводит: 4 2 1 6 9
 while (!s.empty()) s.pop();
  if (s.empty()) cout << "Queue is empty";
    return 0;
}
```

7.2. Работа с двусвязанными списками

Пример 7.2. Работа с двусвязанным списком.

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

struct TNode {
  int inf;                 // Информационная часть
  TNode* left;             // Адресная часть
  TNode* right;            // Адресная часть
};
```

```
struct list {

TNode* front = nullptr; // Указатель на начало  очереди
TNode* back = nullptr;  // Указатель на конец очереди


  // Проверка наличия элементов в очереди
   bool empty() {
       if (front) return false;
       else return true;
   }
```

```
// Добавление элемента в очередь
    void push(int inf) {
      TNode* spt = new TNode;
            spt->inf = inf;
            spt->right = nullptr;
          if (!front) {
            spt->left = nullptr;
            front = back = spt;
            return;   }
        back->right = spt;
        spt->left = back;
        back = spt;
    }
```

```
// Удаление элемента из очереди
    void pop() {
        TNode* spt = front;
        front = front->right;
        delete spt;
        if (!front) back = nullptr;
        else
        front->left = nullptr;
    }
```

```
// Вывод очереди на экран
    void print() {
        TNode* spt = front;
        while (spt != nullptr) {
            cout << spt->inf << " ";
            spt = spt->right;
        }
    }
```

```
// Поиск элемента с заданным ключем
    TNode* search(int x) {
        if (!front) return nullptr;
        TNode* spt = front;
while (spt->inf != x && spt->right != nullptr)
                                        spt = spt->right;
        if (spt->inf == x) return spt;
        else return nullptr;
    }
```

```
// Поиск элемента с заданным ключем
void del(int x) {
        TNode* spt = search(x);
        if (!spt) return;
        if (front == back)
        {
          front = nullptr;
          back = nullptr;
        }
```

```
else
        if (!spt->left) {
            front = spt->right;
            front->left = nullptr;   }
        else
            if (!spt->right) {
                back = spt->left;
                back->right = nullptr;  }
            else {
                spt->right->left = spt->left;
                spt->left->right = spt->right;   }
    delete spt;
}
```

```
// Добавление элемента после заданного
    void pushleft(TNode* spp, int inf) {
        TNode* spt = new TNode;
        spt->inf = inf;
        spt->right = spp->right;
        spt->left = spp;
        spp->right = spt;
        if (spt->right) spt->right->left = spt;
    }

};
```

```
void main() {
    list s;
s.push(4); s.push(2); s.push(1); s.push(6); s.push(9);
            s.print(); // Выводит: 4 2 1 6 9
   s.del(6);
            s.print(); // Выводит: 4 2 1 9
   s.pushleft(s.search(2), 7);
             s.print(); // Выводит: 4 2 7 1 9
   while (!s.empty()) s.pop();
}
```

7.3. Работа с двусвязанными циклическими списками