БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

**Динамические структуры данных. Дек. Тестирование кода.**

**По:** Основы алгоритмизации и программирования

**Выполнил:** Корнелюк

Валентин Владимирович

1 курс 4 группа ПОИТ

**Преподаватель:**

Белодед Николай Иванович

**г. Минск**

2023 г.

**Оглавление**

[Формирование дека, просмотр его содержимого, добавление элемента к деку и удаление элемента из дека. 3](#_Toc131623666)

# Формирование дека, просмотр его содержимого, добавление элемента к деку и удаление элемента из дека.

#include<iostream>

using namespace std;

struct node

{

int elem; // Значение узла

node\* sled; // Указатель на следующий узел

};

class Spisok

{

private:

node\* ld, \* rd; // Указатели на левый и правый концы списка

int el\_left, el\_right; // Количество элементов слева и справа от центра списка

public:

// Объявляем методы класса

void POSTROENIE(); // Метод для построения списка

void VYVOD(); // Метод для вывода списка на экран

void VSTAV1(int); // Метод для вставки элемента в конец списка

void VSTAV2(int); // Метод для вставки элемента в начало списка

int SetElLeft() { return el\_left; } // Метод для получения количества элементов слева от центра списка

int SetElRight() { return el\_right; } // Метод для получения количества элементов справа от центра списка

void YDALE1(); // Метод для удаления элемента справа от центра списка

void YDALE2(); // Метод для удаления элемента слева от центра списка

void OCHISTKA(); // Метод для очистки списка

};

void main()

{

setlocale(LC\_ALL, "Rus"); // Установка локали для отображения русского языка в консоли

Spisok A; // Создание объекта класса Spisok

int el; // Переменная для хранения вводимых значений элементов дека

A.POSTROENIE(); A.VYVOD(); // Вызов методов класса для построения и вывода дека

cout << "Добавим звено справа.\n";

cout << "Введите элемент добавляемого звена: ";

cin >> el; // Ввод элемента для добавления в дек

A.VSTAV1(el); A.VYVOD(); // Вызов методов класса для добавления элемента в дек и вывода дека

cout << "Добавим звено слева.\n";

cout << "Введите элемент добавляемого звена: ";

cin >> el; // Ввод элемента для добавления в дек

A.VSTAV2(el); A.VYVOD(); // Вызов методов класса для добавления элемента в дек и вывода дека

cout << "Удалим звено справа.\n";

A.YDALE1(); A.VYVOD(); cout << A.SetElRight() << endl; // Вызов методов класса для удаления элемента из дека, вывода дека и значения правого конца дека

cout << "Удалим зввено слева.\n";

A.YDALE2(); A.VYVOD(); cout << A.SetElLeft() << endl; // Вызов методов класса для удаления элемента из дека, вывода дека и значения левого конца дека

A.OCHISTKA(); // Вызов метода класса для освобождения памяти, выделенной под звенья дека

cout << "\n";

system("PAUSE"); // Ожидание нажатия клавиши для завершения программы

}

void Spisok::POSTROENIE()

// Функция для построения дека:

// ld - указатель на левый конец дека,

// rd - указатель на правый конец дека.

{

node\* k; // объявление указателя на звено дека

int el; // объявление переменной для считывания значений элементов

cout << "Вводите содержимое звеньев дека: \n"; // вывод текста на экран

cin >> el; // считывание значения первого элемента дека

if (el != 0) // проверка, что первый элемент не равен 0 (если равен, то дек будет пуст)

{

k = new (node); // создание нового звена дека

(\*k).elem = el; (\*k).sled = NULL; // заполнение поля elem нового звена и установка указателя на следующее звено равным NULL

ld = k; rd = k; // установка указателей на начало и конец дека на новое звено

cin >> el; // считывание следующего значения

while (el != 0) // цикл по добавлению остальных элементов, пока не будет введен 0

{

VSTAV1(el); // добавление нового элемента в дек справа

cin >> el; // считывание следующего значения

}

}

else

{

rd = NULL; ld = NULL; // установка указателей на начало и конец дека на NULL, если дек пуст

}

}

void Spisok::VYVOD()

//Вывод содержимого дека:

// ld - указатель на левый конец дека.

{

node\* k;

k = ld; // присваиваем указатель на левый конец дека переменной k

cout << "Дек: ";

while (k != NULL) // пока не достигнут конец дека

{

cout << (\*k).elem << " "; // выводим элемент текущего звена

k = (\*k).sled; // переходим к следующему звену

}

cout << endl;

}

void Spisok::VSTAV1(int el)

// Помещение звена, содержащего элемент el, в дек справа.

// ld - указатель на левый конец дека,

// rd - указатель на правый конец дека.

{

node\* k;

k = new (node); // выделяем память под новое звено

(\*k).elem = el; // записываем в него переданный элемент

(\*k).sled = NULL; // указываем, что звено находится справа

if (rd != NULL) // если дек не пуст

{

(\*rd).sled = k; // связываем предыдущее звено с добавляемым

rd = k; // обновляем указатель на правый конец дека

}

else // если дек пуст

{

rd = k; // обновляем указатели на оба конца дека

ld = k;

}

}

void Spisok::VSTAV2(int el)

//Помещение звена, содержащего элемент el, в дек слева.

// ld - указатель на левый конец дека,

// rd - указатель на правый конец дека.

{

node\* k;

k = new (node);// Выделяем память под новый узел.

(\*k).elem = el;// Задаем значение элемента нового узла.

(\*k).sled = ld;// Назначаем указатель на следующий узел равным текущему левому.

if (ld != NULL) ld = k;// Если левый узел не равен NULL, то обновляем указатель на левый узел.

else // Иначе, если левый узел равен NULL, то дек пустой и новый узел становится и левым, и правым.

{

ld = k;

rd = k;

}

}

void Spisok::YDALE1()

//Удаление звена из дека справа

//с сохранением удаляемого звена в переменной el\_right.

// ld - указатель на левый конец дека,

// rd - указатель на правый конец дека.

{

node\* z;

node\* k;

if (rd == ld) // Если дек состоит из одного элемента.

{

el\_right = (\*rd).elem;

delete rd;

ld = rd = NULL;

cout << "Дек пуст!\n";

}

else // Если в деке больше одного элемента.

{

z = ld; // Назначаем указатель на предпоследний узел равным левому узлу.

k = (\*ld).sled; // Назначаем указатель на последний узел равным следующему узлу после левого.

while (k != rd)// Перебираем узлы, пока не найдем последний узел.

{

z = k; // Передвигаем указатель на предпоследний узел на следующий узел.

k = (\*k).sled; // Передвигаем указатель на последний узел на следующий узел.

}

el\_right = (\*rd).elem; // Сохраняем элемент последнего узла в переменной el\_right.

(\*z).sled = NULL; // Удаляем ссылку на последний узел, делая предпоследний узел последним.

delete rd; // Удаляем последний узел.

rd = z; // Назначаем указатель на правый узел равным предпоследнему узлу.

}

}

void Spisok::YDALE2()

// Удаление звена из дека слева

// с сохранением удаляемого звена в переменной el\_left.

// ld - указатель на левый конец дека,

// rd - указатель на правый конец дека.

{

node\* q;

if (ld != NULL) // проверяем, что дек не пуст

{

el\_left = (\*ld).elem; // сохраняем значение элемента, который будем удалять

q = ld; // сохраняем указатель на удаляемый узел

ld = (\*ld).sled; // сдвигаем указатель на левый конец дека

delete q; // удаляем узел

}

else cout << "Дек пуст!\n"; // выводим сообщение, если дек пустой

}

void Spisok::OCHISTKA()

{

node\* k, \* q;

k = ld;

if (k != NULL) // если дек не пуст

{

q = (\*k).sled;

while (q != NULL)

{

delete k; // удаляем звено k

k = q;

q = (\*k).sled;

}

delete k; // удаляем последнее звено

}

}

