БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

**Динамические структуры данных. Дек на основе двунаправленного списка. 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**

[Пример дека на основе двунаправленного списка. Построение дека, вставка и удаление элементов из дека. 3](#_Toc132752315)

# Пример дека на основе двунаправленного списка. Построение дека, вставка и удаление элементов из дека.

#include <iostream>// Подключение стандартной библиотеки ввода-вывода.

using namespace std;// Использование пространства имен std.

struct node// Определение структуры звена списка.

{

int elem; // Информационное поле звена.

node\* sled; // Указатель на следующий элемент списка.

node\* pred; // Указатель на предыдущий элемент списка.

};

class Spisok// Определение класса дека.

{

private:

node\* nd; // Указатель на начало дека.

node\* kd; // Указатель на конец дека.

int klad; // Информационное поле удаленного элемента.

public:

void BuiltDeck(); // Создание дека.

void VyvodDeck(); // Вывод дека.

void InsLeft(int); // Вставка элемента слева.

void InsRight(int); // Вставка элемента справа.

void DelLeft(); // Удаление элемента слева.

void DelRight(); // Удаление элемента справа.

int Get\_Klad() { return klad; } // Получение значения удаленного элемента.

void Ochistka(); // Очистка дека.

};

void main() // Главная функция программы.

{

setlocale(LC\_ALL, "Rus"); // Установка локали для вывода на кириллице.

Spisok A;// Создание объекта класса дека.

int el;// Объявление переменной для ввода элементов дека.

A.BuiltDeck();// Создание дека;

A.VyvodDeck();// Вывод дека на экран

cout << "Введите элемент звена, вставляемого справа: ";

cin >> el; A.InsRight(el); A.VyvodDeck();// Вставка элемента справа и вывод дека на экран

cout << "Введите элемент звена, вставляемого слева: ";

cin >> el; A.InsLeft(el); A.VyvodDeck();// Вставка элемента слева и вывод дека на экран

cout << "Удалим звено справа: \n";

A.DelRight(); A.VyvodDeck();// Удаление элемента слева и вывод дека на экран

cout << "Был удален элемент: " << A.Get\_Klad() << endl;//Вывод значения удаленного элемента

cout << "Удалим звено слева: \n";

A.DelLeft(); A.VyvodDeck();// Удаление элемента справа и вывод дека на экран

cout << "Был удален элемент: " << A.Get\_Klad() << endl;//Вывод значения удаленного элемента

A.Ochistka(); //очистка дека

cout << "\n";

system("PAUSE");// Ожидание ввода пользователем.

}

void Spisok::BuiltDeck()

// Построение дека на базе двунаправленного

// списка с заглавным звеном.

// nd - указатель на начало дека,

// \*kd - указатель на конец дека.

{

node\* q;// Объявление указателей на узлы структуры node и переменной el, содержащей элемент

node\* z;

int el;

nd = new(node);// Создание заголовочного узла дека и его инициализация

z = nd;

(\*nd).pred = (\*nd).sled = NULL;

cout << "Введите последовательность: \n"; // Пользователь вводит последовательность элементов, которые будут помещены в дек

cin >> el;

while (el != 0)// Пока пользователь не введет 0

{

(\*z).sled = new (node);// Создаем новый узел и связываем его с предыдущим узлом

(\*((\*z).sled)).pred = z;

z = (\*z).sled; // Сдвигаем указатель на новый узел

(\*z).sled = NULL;

(\*z).elem = el;// Инициализируем элемент узла

cin >> el;// Считываем новый элемент

}

if ((\*nd).sled != NULL)// Если в деке есть хотя бы один элемент, то обновляем указатели nd и kd, удаляем заголовочный узел

{

q = nd; nd = (\*nd).sled; (\*nd).pred = NULL;

kd = z; delete q;

}

else // Если в деке нет элементов, то удаляем заголовочный узел и обнуляем указатели nd и kd

{

delete nd; nd = kd = NULL;

}

}

void Spisok::VyvodDeck()

// Вывод содержимого дека.

// nd - указатель на начало дека.

{

node\* z;

z = nd; // Присвоение указателю z начального адреса дека.

cout << "Содержимое дека: "; // Вывод сообщения о содержимом дека.

if (z != NULL) // Проверка на пустоту дека.

while (z != NULL) // Цикл для вывода каждого элемента дека.

{

cout << (\*z).elem << " "; // Вывод значения текущего элемента.

z = (\*z).sled; // Переход к следующему элементу.

}

else cout << "он пуст!\n"; // Вывод сообщения, если дек пустой.

cout << endl; // Перевод строки после вывода содержимого дека.

}

void Spisok::InsLeft(int el)

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

// nd - указатель на начало дека,

// kd - указатель на конец дека.

{

node\* q;

q = new(node); // выделяем память под новый узел

(\*q).elem = el; // присваиваем элемент новому узлу

if (nd == NULL) // если дек пуст

{

nd = q; (\*q).sled = (\*q).pred = NULL; kd = q; // добавляем первый элемент и указываем на него nd и kd

}

else

{

(\*q).sled = nd; (\*q).pred = NULL; // новый узел становится первым и не имеет предыдущего

(\*nd).pred = q; nd = q; // предыдущий узел становится вторым, а указатель на первый элемент обновляется

}

}

void Spisok::InsRight(int el)

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

// nd - указатель на начало дека,

// kd - указатель на конец дека.

{

node\* q;

q = new(node); // Создание нового звена и присваивание ему значение el.

(\*q).elem = el;

if (kd == NULL) // Если дек пуст, то новый элемент становится началом и концом дека.

{

nd = q; (\*q).sled = (\*q).pred = NULL; kd = q;

}

else // Иначе добавление звена в конец дека.

{

(\*q).sled = NULL; (\*q).pred = kd;

(\*kd).sled = q; kd = q;

}

}

void Spisok::DelLeft()

// Удаление звена из дека слева с помещением

// элемента удаляемого звена в переменную klad.

// nd - указатель на начало дека,

// kd - указатель на конец дека.

{

node\* q;

if ((\*nd).sled != NULL)// Если в деке есть больше одного элемента, то удаляем первый элемент.

{

q = nd;// q - указатель на первый элемент

klad = (\*q).elem;// запоминаем элемент удаляемого звена

nd = (\*nd).sled; // обновляем указатель на первый элемент

(\*nd).pred = NULL;// обнуляем указатель на предыдущий элемент

delete q;// удаляем первый элемент

}

else// Если в деке только один элемент, то удаляем его и обнуляем указатели.

{

q = nd;// q - указатель на единственный элемент в деке

klad = (\*q).elem;// запоминаем элемент удаляемого звена

nd = kd = NULL; // обнуляем указатели на начало и конец дека

delete q;// удаляем элемент из дека

cout << "Дек пуст!\n"; // выводим сообщение о том, что дек стал пустым

}

}

void Spisok::DelRight()

// Удаление звена из дека справа с помещением

// элемента удаляемого звена в переменную klad.

// nd - указатель на начало дека,

// kd - указатель на конец дека.

{

node\* q;

if ((\*kd).pred != NULL) // Если в деке больше одного элемента

{

q = kd; // q - указатель на последний элемент

klad = (\*q).elem; // запоминаем элемент удаляемого звена

kd = (\*kd).pred; // обновляем указатель на последний элемент

(\*kd).sled = NULL; // обнуляем указатель на следующий элемент

delete q; // удаляем последний элемент

}

else // Если в деке только один элемент

{

q = kd; // q - указатель на единственный элемент в деке

klad = (\*q).elem; // запоминаем элемент удаляемого звена

nd = kd = NULL; // обнуляем указатели на начало и конец дека

delete q; // удаляем элемент из дека

cout << "Дек пуст!\n"; // выводим сообщение о том, что дек стал пустым

}

}

void Spisok::Ochistka()

{

node\* q, \* q1;

q = nd; // Устанавливаем указатель на начало дека.

q1 = (\*q).sled; // Устанавливаем указатель на второй элемент дека.

while (q1 != NULL)

{ // Удаляем все элементы дека, начиная со второго до последнего.

delete q; q = q1; q1 = (\*q).sled;

}

delete q; // Удаляем последний элемент.

nd = kd = NULL; // Устанавливаем указатели начала и конца дека в NULL, так как дек пуст.

}

