МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет: Информационных технологий

Кафедра: Программной инженерии

Тестирование кодов

По дисциплине” Основы Алгоритмизации и Программирования”

***Выполнил:*** *студент 1 курса 8 группы*

*Федорович В.Г*

**Специальность:** *ПИ 1 курс.*

**Проверил:** преподаватель *Белодед Николай Иванович*

Минск

2024

Оглавление

[**Введение:** 3](#_Toc164811380)

[Код тестирования 1 (ДЕК) 4](#_Toc164811381)

[Код тестирования 2 (Очередь) 8](#_Toc164811382)

# **Введение:**

В этом реферате представлено:

1. Тестирование структуры данных DEC

2. Тестирование структуры данных Queue (Очереди)

Обе структуры реализованы на основе однонаправленных линейных списков

Код тестирования 1 (ДЕК)

#include<iostream>

using namespace std;

struct node

{

int elem;

node\* sled;

};

class Spisok

{

private:

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

int el\_left, el\_right; // Переменные для хранения элементов, удаленных с левого и правого концов

public:

// Прототипы функций

void POSTROENIE(); // Построение списка

void VYVOD(); // Вывод списка

void VSTAV1(int); // Вставка узла в правый конец

void VSTAV2(int); // Вставка узла в левый конец

int SetElLeft() { return el\_left; } // Получение элемента, удаленного с левого конца

int SetElRight() { return el\_right; } // Получение элемента, удаленного с правого конца

void YDALE1(); // Удаление узла с правого конца

void YDALE2(); // Удаление узла с левого конца

void OCHISTKA(); // Очистка списка

};

void main()

{

setlocale(LC\_ALL, "Rus");

Spisok A; // Создание переменной класса Spisok

int el; // Переменная для хранения вводимых пользователем эл. списка

A.POSTROENIE(); A.VYVOD();

cout << "Добавим звено справа.\n";

cout << "Введите элемент добавляемого звена: ";

cin >> el;

A.VSTAV1(el); A.VYVOD();

cout << "Добавим звено слева.\n";

cout << "Введите элемент добавляемого звена: ";

cin >> el;

A.VSTAV2(el); A.VYVOD();

cout << "Удалим звено справа.\n";

A.YDALE1(); A.VYVOD(); cout << A.SetElRight() << endl;

cout << "Удалим зввено слева.\n";

A.YDALE2(); A.VYVOD(); cout << A.SetElLeft() << endl;

A.OCHISTKA();

cout << "\n";

system("PAUSE");

}

// ld - указатель на левый конец дека,

// rd - Указатель на правый конец дека.

void Spisok::POSTROENIE()

{

node\* k; // Временный указатель на узел

int el; // Элемент дека, вводимый пользователем

cout << "Вводите содержимое звеньев дека: \n"; // Предложение для пользователя

cin >> el; // Ввод первого элемента

if (el != 0) // Если элемент не равен 0

{

k = new (node); // Выделение памяти под новый узел

(\*k).elem = el; (\*k).sled = NULL; // Инициализация нового узла

ld = k; rd = k; cin >> el; // Присваивание указателю на левый конец адреса первого узла

while (el != 0) // Цикл, пока пользователь вводит ненулевые элементы

{

VSTAV1(el); cin >> el; // Вызов метода для вставки элемента справа

}

}

else // Если первый элемент равен 0

{

rd = NULL; ld = NULL; // Установка указателей на конец дека в NULL

}

}

// Метод VYVOD() для вывода содержимого дека:

// ld - указатель на левый конец дека.

void Spisok::VYVOD()

{

node\* k; // Временный указатель на узел

k = ld;// присваивание временному указателю адреса левого конца

cout << "Дек: ";

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

{

cout << (\*k).elem << " "; k = (\*k).sled; // Вывод элемента и переход к следующему узлу

}

cout << endl;

}

// ld - указатель на левый конец дека,

// rd - указатель на правый конец дека.

void Spisok::VSTAV1(int el)

{

node\* k; // Временный указатель на узел

k = new (node); // Выделение памяти под новый узел

(\*k).elem = el; (\*k).sled = NULL; // Инициализация нового узла

if (rd != NULL) // Если дек не пуст

{

(\*rd).sled = k; rd = k; // Присоединение нового узла к деку справа

}

else // Если дек пуст

{

rd = k; ld = k; // Установка указателей на конец дека на новый узел

}

}

// ld - указатель на левый конец дека,

// rd - указатель на правый конец дека.

void Spisok::VSTAV2(int el)

{

node\* k; // Временный указатель на узел

k = new (node); // Выделение памяти под новый узел

(\*k).elem = el; (\*k).sled = ld; // Инициализация нового узла

if (ld != NULL) ld = k; // Если дек не пуст, новый узел становится левым концом

else { ld = k; rd = k; } // Если дек пуст, установка указателей на конец дека на новый узел

}

// Метод YDALE2() для удаления звена из дека справа

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

// ld - указатель на левый конец дека,

// rd - указатель на правый конец дека.

void Spisok::YDALE1()

{

node\* z; // Временный указатель на узел

node\* k; // Временный указатель на узел

if (rd == ld) // Если в деке только одно звено

{

el\_right = (\*rd).elem; delete rd; // Сохранение значения и удаление звена

ld = rd = NULL;// Установка указателей на конец дека в NULL

cout << "Дек пуст!\n";

}

else // Если в деке больше одного звена

{

z = ld; k = (\*ld).sled; // Присваивание временным указателям адресов

while (k != rd) // Пока не найдено звено, предшествующее правому концу

{

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

}

el\_right = (\*rd).elem; (\*z).sled = NULL; delete rd; // Сохранение значения и удаление звена

rd = z; // Установка правого конца на предыдущее звено

}

}

// Метод YDALE2() для удаления звена из дека слева

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

// ld - указатель на левый конец дека,

// rd - указатель на правый конец дека.

void Spisok::YDALE2()

{

node\* q; // Временный указатель на узел

if (ld != NULL) // Если дек не пуст

{

el\_left = (\*ld).elem; q = ld; // Сохранение значения и присвоение временному указателю адреса левого конца

ld = (\*ld).sled; delete q; // Установка левого конца на следующее звено и удаление звена

}

else cout << "Дек пуст!\n"; // Вывод сообщения, если дек пуст

}

// Метод OCHISTKA() для возврата выделенной памяти в "кучу".

void Spisok::OCHISTKA()

{

node\* k, \* q; // Временные указатели на узлы

k = ld; // Присваивание временному указателю адреса левого конца

if (k != NULL) // Если дек не пуст

{

q = (\*k).sled; // Присваивание временному указателю адреса следующего узла

while (q != NULL) // Пока не достигнут конец дека

{

delete k; // Освобождение памяти, занимаемой текущим узлом

k = q; // Переход к следующему узлу

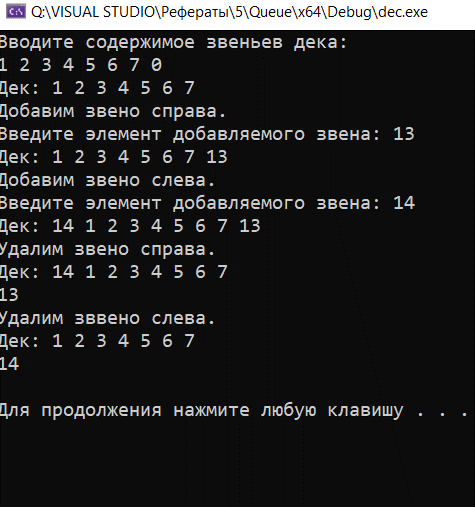
q = (\*q).sled; // Переход к следующему узлу

}

delete k; // Освобождение памяти, занимаемой последним узлом

}

}



Код тестирования 2 (Очередь)

#include <iostream>

using namespace std;

// Определение структуры узла для очереди

struct node

{

int elem; // Элемент данных

node\* sled; // Указатель на следующий узел

};

// Класс для работы с очередью

class Spisok {

private:

node\* no, \* ko; // Указатели на начало очереди и ее конец

int klad; // Переменная для хранения элемента, удаленного из очереди

public:

// Конструктор класса

Spisok()

{

no = ko = NULL; // Начальная нициализация указателей на начало и конец как NULL

}

// Методы для класса

void POSTROENIE(); // Построение очереди

void VYVOD(); // Вывод содержимого очереди

void DOBAVLENIE(int); // Добавление элемента в очередь

int SetUdal() { return klad; } // Получение инфо-поля(или просто содержания) удаленного узла

void YDALENIE(); // Удаление элемента из очереди

void OCHISTKA(); // Очистка очереди

};

// Точка входа в программу

int main()

{

setlocale(LC\_ALL, "Rus");

Spisok A; // Создание элемента класса Spisok

int el; // Переменная для хранения вводимых пользователем элементов

A.POSTROENIE(); // Построение очереди

A.VYVOD(); // Вывод содержимого очереди

cout << "Введите добавляемый элемент: ";

cin >> el;

A.DOBAVLENIE(el); // Добавление элемента в очередь

A.VYVOD(); // Вывод обновленной очереди

cout << "Удалим элемент из очереди.\n";

A.YDALENIE(); // Удаление элемента из очереди

A.VYVOD(); // Вывод обновленной очереди

el = A.SetUdal(); // Получение содердания удаленного узла

cout << "Информационное поле удаленного звена: " << el << endl;

A.OCHISTKA(); // Очистка очереди

cout << "\n";

system("PAUSE");

}

void Spisok::POSTROENIE()

{

node\* r; // Объявление рабочего указателя

int el; // Переменная для хранения вводимых пользователем элементов

cout << "Введите элементы очереди: ";

cin >> el; // Ввод первого элемента очереди

if (el != 0) // Проверка, не является ли первый элемент нулем то есит завершающим элементом

{

r = new (node); // Выделение памяти для нового узла

(\*r).elem = el; // Присваивание нового значения (введенного пользователем) элементу узла

(\*r).sled = NULL; // Установка указателя на следующий узел в NULL, так как это первый узел

no = r; // Установка указателя начала очереди на созданный узел

ko = r; // Установка указателя конца очереди на созданный узел

cin >> el; // Ввод следующего элемента очереди

while (el != 0) // Пока вводимые пользователм элементы не "0" (пока очередь не закончена)

{

r = new (node); // Выделение памяти под новый узел

(\*r).elem = el; // Присваивание нового значения (введенного пользователем) элементу узла

(\*r).sled = NULL; // Установка указателя на следующий узел в NULL

(\*ko).sled = r; // Установка указателя на следующий узел предыдущего узла на новый узел

ko = r; // Перемещение указателя на конец очереди на новый узел

cin >> el; // Ввод след. элемента очереди

}

}

else // Если введенный элемент = 0 (завершающий элемент)

{

r = NULL; // Устанавливаем указатель на узел в NULL

no = r; // Устанавливаем указатель на начало очереди в NULL

ko = r; // Устанавливаем указатель на конец очереди в NULL

}

}

void Spisok::VYVOD()

{

node\* r; // Объявление указателя на узел

cout << "Очередь: ";

r = no; // указатель на начало очереди

while (r != NULL) // Пока пока не дойдем до конца очереди

{

cout << (\*r).elem << " "; // Выводим значение текущего узла

r = (\*r).sled; // Переход к следующему узлу

}

cout << endl;

}

void Spisok::DOBAVLENIE(int el)

{

node\* r; // Объявление рабочего указателя на узел

r = new (node); // Выделение памяти под новый узел

(\*r).elem = el; // Присваивание значения элементу узла

(\*r).sled = NULL; // Установка указателя на следующий узел в NULL

if (no != NULL) // Проверка, не пуста ли очередь

{

(\*ko).sled = r; // Установка указателя на след. узел последнего узла на новый узел

ko = r; // Перемещение указателя на конец очереди на новый узел

}

else // Если очередь пуста

{

no = r; // Устанавливаем указатель начала очереди на новый узел

ko = r; // Устанавливаем указатель конца очереди на новый узел

}

}

void Spisok::YDALENIE()

{

node\* q; // Объявление рабочего указателя на узел

if (no == NULL) // Проверка, пуста ли очередь

{

cout << "Удалить элемент нельзя, так как очередь пуста!\n";

}

else

{

klad = (\*no).elem; // Сохранение значения элемента, который будет удален (Совет Бывалого)

q = no; // Присваивание указателю q адреса первого узла в очереди

no = (\*no).sled; // Перемещение указателя начала очереди на след. узел

delete q; // Удаление первого узла

}

}

void Spisok::OCHISTKA()

{

node\* q; // Объявление указателя на узелk

q = no; // Присваивание указателю q адреса начала очереди

if (no != NULL) // Проверка, не является ли очередь пустой

{

while (no != ko) // будет выполняться, пока не достигнут конец очереди

{

no = (\*q).sled; // Перемещение указателя начала очереди на следующий узел

delete q; // Удаление текущего узла

q = no; // Перемещение указателя q на след. узел

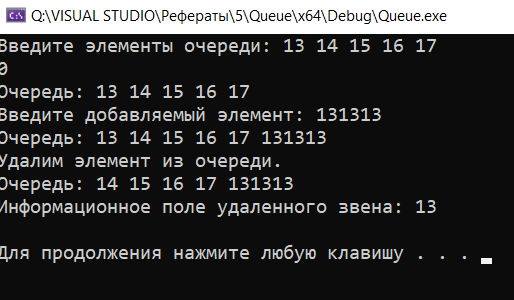
}

delete no; // Удаление последнего узла очереди

no = ko = NULL; // Установка указателей конца и начала в NULL

}

}

****