Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Реферат

По дисциплине «Основы алгоритмизации и программирования»

На тему «Тестирование кода лекции очереди и дека»

Выполнила:

Студент(ка) 1 курса 7 группы

Подшиваленко Диана Игоревна

Проверил:

Белодед Николай Иванович

2024, Минск

**Содержание**

1. [Тестирование кода очереди на основе односвязного списка … 3](#_1.Тестирование_кода_очереди)
2. [Тестирование кода дека на основе односвязного списка … 5](#_2.Тестирование_кода_дека)

# **1.Тестирование кода очереди на основе односвязного списка**

#include <iostream>

#include <string.h>

#include <stdio.h>

using namespace std;

struct node { // структура узла

int elem; // информационное поле

node\* sled; // указатель на следующий элемент

};

class Spisok { // класс списка

private: // приватная часть

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

int klad; // вспомагательный указатель для сохранения данных удаленных элементов

public:// публичная зона

Spisok() { // конструктор списка

no = ko = NULL; // присваиваем указателю на начало и конец очереди NULL

}

void POSTROENIE(); // прототип функции построения очереди

void VYVOD(); // прототип функции вывода элементов очереди

void DOBAVLENIE(int); // прототип функции добавления элемента

int Set\_Udal() { // функция для получения удаленного элемента

return klad; // возвращаем значение удаляемого элемента

}

void YDALENIE(); // прототип функции удаления элемента

void OCHISTKA(); // прототип функции очистки очереди

};

void main()

{

setlocale(LC\_ALL, "Rus");

Spisok A; // создаем объект класса Spisok

int el; // буферная переменная

A.POSTROENIE(); // вызываем функцию построения

A.VYVOD(); // вызываем функцию вывода

cout << "Введите добавляемый элемент\n";

cin >> el; // считываем введенный элемент

A.DOBAVLENIE(el); // вызываем функцию добавления введенного элемента

A.VYVOD(); // вызываем функцию вывода

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

A.YDALENIE(); // вызываем функцию удаления элемента

A.VYVOD(); // вызываем функцию вывода элемента

el = A.Set\_Udal(); // присваиваем переменной el значение удаленного элемента, которое мы получаем с помощью функции, которая возвращает это значение

cout << "Информационное поле удаленного звена " << el << '\n';

A.OCHISTKA(); // вызываем функцию очистки очереди

cout << '\n';

system("PAUSE");

}

void Spisok::POSTROENIE() { // функция для построения очереди

node\* r; // буферный указатель

int el; // буферная переменная

cout << "Введите элементы очереди\n";

cin >> el; // считываем элемент

if (el != 0) { // если не встретился 0(признак окончания ввода)

r = new node; // выделяем память для нового элемента

(\*r).elem = el; // записываем в информационное поле текущее введенное значение

(\*r).sled = NULL; // присваиваем указателю на следующий элемент NULL

no = r; // присваиваем указателю на начало адрес, на который указывает r

ko = r;// присваиваем указателю на конец адрес, на который указывает r

cin >> el; // считываем следующий элемент

while (el != 0) { // пока не встретился 0(признак окончания ввода)

r = new node;// выделяем память для нового элемента

(\*r).elem = el; // записываем в информационное поле текущее введенное значение

(\*r).sled = NULL;// присваиваем указателю на следующий элемент NULL

(\*ko).sled = r; // присваиваем указателю на следующий за конечным элементом адрес, на который указывает r

ko = r; // присваиваем указателю на конечный элемент адрес, на который ссылается r

cin >> el; // считываем следующий элемент

}

}

else { // иначе

r = NULL; // присваиваем r NULL

no = r; // присваиваем указателю на начало адрес, на который указывает r

ko = r;// присваиваем указателю на конец адрес, на который указывает r

}

}

void Spisok::VYVOD() { // функция для вывода элементов очереди

node\* r; // буферный указатель

cout << "Очередь: \n";

r = no; // устанавливаем r на начало очереди

while (r != NULL) { // пока не дошли до конца

cout << (\*r).elem << ' '; // выводим информационное поле текущего элемента

r = (\*r).sled; // переходим на следующий элемент

}

cout << '\n';

}

void Spisok::DOBAVLENIE(int el) { // функция для добавления элемента

node\* r; // буферный указатель

r = new node; // выделяем для него память

(\*r).elem = el; // записываем в информационное поле переданное значение

(\*r).sled = NULL;// присваиваем указателю на следующий элемент NULL

if (no != NULL) { // если очередь не пуста

(\*ko).sled = r; // присваиваем указателю на следующий за конечным элементом адрес, на который указывает r

ko = r; // присваиваем указателю на конечный элемент адрес, на который ссылается r

}

else { // иначе

no = r;// присваиваем указателю на начало адрес, на который указывает r

ko = r;// присваиваем указателю на конец адрес, на который указывает r

}

}

void Spisok::YDALENIE() { // функция для удаления элемента

node\* q; // буферный указатель

if (no == NULL) { // если очередь пуста

cout << "Очередь пуста\n"; // выводим сообщение об этом

return; // завершаем работу функции

}

//иначе

klad = (\*no).elem; // присваиваем специальной переменной для хранения удаленного значения текущее значение удаляемого элемента

q = no; // переходим на следующий элемент

no = (\*no).sled; // ссылаем указатель на начало на следующий за ним элемент

delete q; // очищаем память

}

void Spisok::OCHISTKA() { // функция очистки очереди

node\* q; // буферный указатель

q = no; // ссылаем его на начало

if (no != NULL) { // пока не дошли до конца

while (no != ko) { // пока указатель на начало и на конец не ссылаются на один и тот же элемент

no = (\*q).sled; // указатель на начало ссылаем на следующий элемент

delete q; // очищаем память

q = no; // ссылаем указатель q на новое начало

}

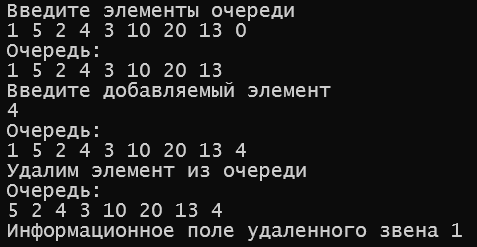
delete no; // очищаем память

}

no = ko = NULL; // присваиваем указателю на начало и конец очереди NULL

}

Результат выполнения:



# **2.Тестирование кода дека на основе односвязного списка**

#include <iostream>

#include <string.h>

#include <stdio.h>

using namespace std;

struct node { // структура узла

int elem; // информационное поле

node\* sled; // указатель на следующий элемент

};

class Spisok { // класс списка

private: // приватная часть

node \*ld, \*rd; // указатели на начало слева и справа соответственно

int el\_left, el\_right; // специальные переменные для хранения удалемых слева и справа элементов

public:// публичная зона

void POSTROENIE(); // прототип функции построения дека

void VYVOD(); // прототип функции вывода элементов дека

void VSTAV1(int); // прототип функции добавления элемента справа

void VSTAV2(int); // прототип функции добавления элемента слева

int SetElLeft() { // функция для возвращения удаленного слева элемента

return el\_left; // возвращаем удаленное значение элемента слева

}

int SetElRight() {// функция для возвращения удаленного справа элемента

return el\_right;// возвращаем удаленное значение элемента справа

}

void YDALE1(); // прототип функции удаления элемента справа

void YDALE2(); // прототип функции удаления элемента слева

void OCHISTKA(); // прототип функции очистки дека

};

void main()

{

setlocale(LC\_ALL, "Rus");

Spisok A; // создаем объект класса Spisok

int el; // буферная переменная

A.POSTROENIE(); // вызываем функцию построения

A.VYVOD(); // вызываем функцию вывода

cout << "Добавим звено справа\n";

cout << "Введите элемент добавляемого звена\n";

cin >> el; // считываем введенный элемент

A.VSTAV1(el); // вызываем функцию добавления введенного элемента справа

A.VYVOD(); // вызываем функцию вывода

cout << "Добавим звено слева\n";

cout << "Введите элемент добавляемого звена\n";

cin >> el; // считываем введенный элемент

A.VSTAV2(el); // вызываем функцию добавления введенного элемента слева

A.VYVOD(); // вызываем функцию вывода

cout << "Удалим звено справа\n";

A.YDALE1(); // вызываем функцию удаления элемента справа

A.VYVOD(); // вызываем функцию вывода элемента

cout << "Удаленное звено " << A.SetElRight() << '\n';

cout << "Удалим звено слева\n";

A.YDALE2(); // вызываем функцию удаления элемента слева

A.VYVOD(); // вызываем функцию вывода элемента

cout << "Удаленное звено " << A.SetElLeft() << '\n';

A.OCHISTKA(); // вызываем функцию очистки дека

cout << '\n';

system("PAUSE");

}

void Spisok::POSTROENIE() { // функция для построения дека

node\* k; // буферный указатель

int el; // буферная переменная

cout << "Введите элементы дека\n";

cin >> el; // считываем элемент

if (el != 0) { // если не встретился 0(признак окончания ввода)

k = new node; // выделяем память для нового элемента

(\*k).elem = el; // записываем в информационное поле текущее введенное значение

(\*k).sled = NULL; // присваиваем указателю на следующий элемент NULL

ld = k; // ссылаем левый указатель на начало на элемент k

rd = k; // ссылаем правый указатель на начало на элемент k

cin >> el; // считываем следующий элемент

while (el != 0) { // пока не встретился 0(признак окончания ввода)

VSTAV1(el); // вставляем элементы справа

cin >> el; // считываем следующий элемент

}

}

else { // иначе

rd = NULL; // присваиваем правому указателю NULL

ld = NULL;// присваиваем левому указателю NULL

}

}

void Spisok::VYVOD() { // функция для вывода элементов дека

node\* k; // буферный указатель

k = ld; // ссылаем на начало слева

cout << "Дек: \n";

while (k != NULL) { // пока не дошли до конца

cout << (\*k).elem << ' '; // выводим информационное поле текущего элемента

k = (\*k).sled; // переходим на следующий элемент

}

cout << '\n';

}

void Spisok::VSTAV1(int el) { // функция для добавления элемента справа

node\* k; // буферный указатель

k = new node; // выделяем для него память

(\*k).elem = el; // записываем в информационное поле переданное значение

(\*k).sled = NULL;// присваиваем указателю на следующий элемент NULL

if (rd != NULL) { // если дек не пуст

(\*rd).sled = k; // присваиваем указателю на следующий за конечным элементом адрес, на который указывает k

rd = k; // присваиваем указателю на конечный элемент адрес, на который ссылается k

}

else { // иначе

rd = k; // ссылаем указатель на начало справа на k

ld = k; // ссылаем указатель на начало слева на k

}

}

void Spisok::VSTAV2(int el) { // функция для добавления элемента слева

node\* k; // буферный указатель

k = new node; // выделяем для него память

(\*k).elem = el; // записываем в информационное поле переданное значение

(\*k).sled = ld;// ссылаем k на начало слева

if (ld != NULL) { // если дек не пуст

ld = k; // присваиваем указателю на начальный элемент адрес, на который ссылается k

}

else { // иначе

rd = k;// ссылаем указатель на начало справа на k

ld = k;// ссылаем указатель на начало слева на k

}

}

void Spisok::YDALE1() { // функция для удаления элемента справа

node\* z; // буферный указатель

node\* k; // буферный указатель

if (rd == ld) { // если указатели на начало слева и справа ссылаются

el\_right = (\*rd).elem; // записываем в элемент для хранения удаленного значение справа этот элемент

delete rd; // очищаем память

ld = rd = NULL; // присваиваем указаетлям на начало справа и начало слева значение NULL

cout << "Дек пуст\n"; // выводим сообщение, что дек пуст

return; // завершаем работу функции

}

//иначе

z = ld; // ссылаем z на начало дека слева

k = (\*ld).sled; // k ссылаем на следующий элемент

while (k != rd) { // пока k не будет ссылаться на начало справа

z = k; // z ссылаем на k

k = (\*k).sled; // переходим на следующий элемент

}

el\_right = (\*rd).elem;// записываем в элемент для хранения удаленного значение справа этот элемент

(\*z).sled = NULL; // а указателю на следующий элемент присваиваем NULL

delete rd; // очищаем память

rd = z; // перенастраиваем указатель на начало справа на z

}

void Spisok::YDALE2() { // функция для удаления элемента слева

node\* q; // буферный указатель

if (ld != NULL) { // если дек не пуст

el\_left = (\*ld).elem;// записываем в элемент для хранения удаленного значение слева этот элемент

q = ld; // ссылаем q на начало слева

ld = (\*ld).sled; // перенастраиваем указатель на начало слева на следующий элемент

delete q; // освобождаем память

return; // завершаем работу функции

}

//иначе

cout << "Дек пуст\n"; // выводим сообщение

}

void Spisok::OCHISTKA() { // функция очистки дека

node\* q, \*k; // буферные указатели

k = ld; // ссылаем его на начало слева

if (k!= NULL) { // пока не дошли до конца

q = (\*k).sled; // ссылаем q на следующий элемент

while (q != NULL) { // пока не дошли до конца

delete k; // очищаем память

k = q; // ссылаем k на q

q = (\*k).sled; // переходим на следующий элемент

}

delete k;// очищаем память

}

}

Результат выполнения:

