Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Реферат

По дисциплине «Основы алгоритмизации и программирования»

На тему «Тестирование кода очереди и дека, реализованных на основе односвязного списка»

Выполнил:

Студент 1 курса 8 группы

Лужецкий Владислав Константинович

Проверил:

Белодед Николай Иванович

2024, Минск

Оглавление

[Очередь, реализованная на основе односвязного списка 2](#_Toc164812070)

[Дек, реализованный на основе односвязного списка 5](#_Toc164812071)

# Очередь, реализованная на основе односвязного списка

#include <iostream>

#include <string.h>

#include <stdio.h>

using namespace std;

struct node { // Структура узла списка

int elem; // Информационное поле

node\* sled; // Указатель на следующий элемент

};

class Spisok {

private:

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

int klad; // Информационное поле удаленного звена

public:

Spisok() { // Конструктор класса

no = ko = NULL; // Инициализация указателей значением NULL

}

void POSTROENIE(); // Функция для построения списка

void VYVOD(); // Функция для вывода списка

void DOBAVLENIE(int); // Функция для добавления элемента в список

int Set\_Udal() { // Функция для получения значения удаленного звена

return klad; // Возвращает значение информационного поля удаленного звена

}

void YDALENIE(); // Функция для удаления элемента из списка

void OCHISTKA(); // Функция для очистки списка

};

void main() {

setlocale(LC\_ALL, "Rus");

Spisok A; // Создание объекта класса Spisok

int el; // Переменная для хранения

A.POSTROENIE(); // Вызов функции для построения списка

A.VYVOD(); // Вызов функции для вывода списка на экран

cout << "Введите добавляемый элемент\n";

cin >> el; // Ввод добавляемого элемента

A.DOBAVLENIE(el); // Вызов функции для добавления элемента в список

A.VYVOD(); // Вызов функции для вывода списка на экран

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

A.YDALENIE(); // Вызов функции для удаления элемента из списка

A.VYVOD(); // Вызов функции для вывода списка

el = A.Set\_Udal(); // Присваивание переменной значение информационного поля удаленного звена

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

A.OCHISTKA(); // Вызов функции для очистки списка

cout << '\n';

system("PAUSE");

}

void Spisok::POSTROENIE() {

node\* r; // Указатель на текущий узел

int el; // Переменная для хранения

cout << "Введите элементы очереди\n";

cin >> el; // Ввод переменной

if (el != 0) {

r = new node; // Выделяем память под новый узел

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

(\*r).sled = NULL; // Устанавливаем следующий указатель узла в NULL

no = r; // Присваиваем началу очереди на созданный узел

ko = r; // Присваиваем концу очереди на созданный узел

cin >> el; // Считываем следующий элемент

while (el != 0) {

r = new node; // Создаем новый узел

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

(\*r).sled = NULL;// Устанавливаем следующий указатель узла в NULL

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

ko = r; // Обновляем указатель на конец очереди

cin >> el; // Считываем следующий элемент

}

}

else { // Иначе

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

no = r; // Устанавливаем начало очереди в NULL

ko = r; // Устанавливаем конец очереди в NULL

}

}

void Spisok::VYVOD() {

node\* r;

cout << "Очередь: \n";

r = no; // Устанавливаем указатель на текущий узел в начало очереди

while (r != NULL) { // Пока != NULL

cout << (\*r).elem << ' '; // Выводим значение информационного поля текущего узла

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

}

cout << '\n';

}

void Spisok::DOBAVLENIE(int el) {

node\* r;

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

(\*r).elem = el; // Присваиваем значение el в информационное поле узла

(\*r).sled = NULL; // Устанавливаем следующий указатель узла на NULL

if (no != NULL) {

(\*ko).sled = r; // Устанавливаем следующий указатель узла на новый узел

ko = r; // Обновляем указатель на конец очереди

}

else {

no = r; // Устанавливаем начало очереди на новый узел

ko = r; // Устанавливаем конец очереди на новый узел

}

}

void Spisok::YDALENIE() {

node\* q;

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

cout << "Очередь пуста\n";

return;

}

klad = (\*no).elem; // Сохраняем значение информационного поля первого узла

q = no; // Устанавливаем указатель q на первый узел

no = (\*no).sled; // Переходим к следующему узлу

delete q; // Удаляем первый узел

}

void Spisok::OCHISTKA() {

node\* q;

q = no; // Устанавливаем указатель на начало очереди

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

while (no != ko) { // Пока != NULL

no = (\*q).sled; // Переходим к следующему узлу

delete q; // Удаляем текущий узел

q = no; // Обновляем указатель q

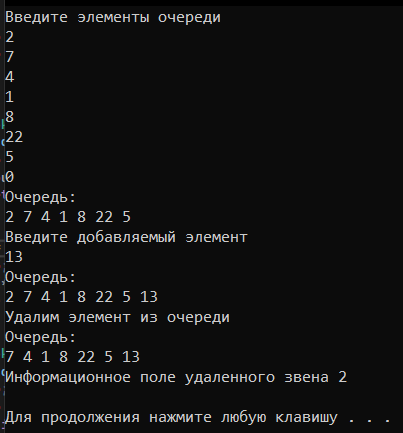
}

delete no; // Удаляем последний узел

}

no = ko = NULL; // Устанавливаем начало и конец очереди в NULL

}



# Дек, реализованный 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 {

private:

node\* ld, \* rd; // Указатели на левое и правое звено

int el\_left, el\_right; // Значения информационных полей левого и правого звена

public:

void POSTROENIE(); // Функция построения списка

void VYVOD(); // Функция вывода списка

void VSTAV1(int); // Функция вставки звена справа

void VSTAV2(int); // Функция вставки звена слева

int SetElLeft() { // Функция получения значения информ. поля левого звена

return el\_left;

}

int SetElRight() { // Функция получения значения информ. поля правого звена

return el\_right;

}

void YDALE1(); // Функция удаления звена справа

void YDALE2(); // Функция удаления звена слева

void OCHISTKA(); // Функция очистки списка

};

void main() {

setlocale(LC\_ALL, "Rus");

Spisok A; // Создание объекта класса Spisok

int el;

A.POSTROENIE(); // Вызов ф-ции для построения списка

A.VYVOD(); // Вызов ф-ции для вывода списка

cout << "Добавим звено справа\n";

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

cin >> el;

A.VSTAV1(el); // Вызов ф-ции для вставки звена справа

A.VYVOD(); // Вызов ф-ции для вывода списка

cout << "Добавим звено слева\n";

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

cin >> el;

A.VSTAV2(el); // Вызов ф-ции для вставки звена слева

A.VYVOD(); // Вызов ф-ции для вывода списка

cout << "Удалим звено справа\n";

A.YDALE1(); // Вызов ф-ции для удаления звена справа

A.VYVOD(); // Вызов ф-ции для вывода списка

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

cout << "Удалим звено слева\n";

A.YDALE2(); // Вызов ф-ции для удаления звена слева

A.VYVOD(); // Вызов ф-ции для вывода списка

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

A.OCHISTKA(); // Вызов ф-ции для очистки списка

cout << '\n';

system("PAUSE");

}

void Spisok::POSTROENIE() {

node\* k; // Указатель на узел

int el; // Переменная для ввода

cout << "Введите элементы дека\n";

cin >> el;

if (el != 0) {

k = new node; // Выделение памяти для нового узла

(\*k).elem = el; // Присвоение значению узла введенного элемента

(\*k).sled = NULL; // Устанавливаем указатель на следующий узел равным NULL

ld = k; // Присваиваем левое и правое звено равными созданному узлу

rd = k;

cin >> el;

while (el != 0) {

VSTAV1(el); // Вызов ф-ции для вставки элемента в дек

cin >> el;

}

}

else { // Иначе

rd = NULL; // Устанавливаем левое и правое звено равными NULL

ld = NULL;

}

}

void Spisok::VYVOD() {

node\* k; // Указатель на узел

k = ld; // Устанавливаем указатель на левое звено дека

cout << "Дек: \n";

while (k != NULL) {

cout << (\*k).elem << ' '; // Вывод значения текущего узла

k = (\*k).sled; // Переходим к следующему узлу

}

cout << '\n';

}

void Spisok::VSTAV1(int el) {

node\* k; // Указатель на узел

k = new node; // Выделение памяти для нового узла

(\*k).elem = el; // Присвоение значению узла введенного элемента

(\*k).sled = NULL; // Указатель на следующий узел равным NULL

if (rd != NULL) {

(\*rd).sled = k; // Указатель следующего узла в правом звене на новый узел

rd = k; // Переходим к новому узлу

}

else { // Если дек пустой

rd = k; // Устанавливаем как левое и правое звено равными новому узлу

ld = k;

}

}

void Spisok::VSTAV2(int el) {

node\* k;

k = new node; // Выделение памяти для нового узла

(\*k).elem = el; // Установливаем значения элемента введенным значением

(\*k).sled = ld; // Устанавливаем указателя на следующий узел на текущее левое звено

if (ld != NULL) {

ld = k; // Перенос текущего левого звена на новый узел

}

else { // Если дек пустой

rd = k; // Устанавливаем правого звена на новый узел

ld = k; // Устанавливаем левого звена на новый узел

}

}

void Spisok::YDALE1() {

node\* z;

node\* k;

if (rd == ld) { // Если правое и левое звено равны

el\_right = (\*rd).elem; // Запись значения правого элемента

delete rd; // Удаление правого звена

ld = rd = NULL; // Обнуление указателей на звенья

cout << "Дек пуст\n";

return;

}

z = ld; // Инициализация z

k = (\*ld).sled; // Инициализация k

while (k != rd) {

z = k; // Перенос текущего звена на следующее

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

}

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

(\*z).sled = NULL; // Устанавливаем указателя следующего узла на NULL

delete rd; // Удаление правого звена

rd = z; // Правое звено становится предыдущим звеном z

}

void Spisok::YDALE2() {

node\* q;

if (ld != NULL) {

el\_left = (\*ld).elem;// Запись значения левого элемента

q = ld; // Указатель q на левое звено

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

delete q; // Удаление предыдущего левого звена

return;

}

cout << "Дек пуст\n";

}

void Spisok::OCHISTKA() {

node\* q, \* k;

k = ld; // Указатель k указывает на левое звено

if (k != NULL) {

q = (\*k).sled; // Указатель q указывает на следующее звено

while (q != NULL) { // Пока есть следующие звенья

delete k; // Удаление текущего звена k

k = q; // Перенос указателя на следующее звено

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

}

delete k; // Удаление k

}

}

