Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Реферат

По дисциплине «Основы алгоритмизации и программирования»

На тему «Тестирование кода двусвязного списка из лекции»

Выполнил:

Студент 1 курса 8 группы

Лужецкий Владислав Константинович

Проверил:

Белодед Николай Иванович

2024, Минск

#include <iostream>

#include <Windows.h>

using namespace std;

// Структура, представляющая узел списка

struct node {

int elem; // Значение узла

node\* sled; // Указатель на следующий узел

node\* pred; // Указатель на предыдущий узел

};

// Класс, представляющий двусвязный список

class Spisok {

private:

node\* nsp; // Указатель на начало списка

node\* ksp; // Указатель на конец списка

public:

Spisok() {

nsp = ksp = NULL;

}

void Postroenie(); // Прототип для построения списка

void VyvodForward(); // Прототип для вывода списка в прямом направлении

void VyvodBack(); // Прототип для вывода списка в обратном направлении

void Ochistka(); // Прототип для очистки списка

void InsAfter(int, node\*); // Прототип для вставки элемента после указанного узла

void InsBefore(int, node\*);// Прототип для вставки элемента перед указанным узлом

void Delete(node\*); // Прототип для удаления указанного узла

void DelAfter(node\*); // Прототип для удаления узла после указанного узла

node\* PoiskForward(int); // Прототип для поиска узла с указанным значением в прямом направлении

node\* PoiskBack(int); // Прототип для поиска узла с указанным значением в обратном направлении

};

int main() {

SetConsoleCP(1251); // Устанавливаем кодировку консоли для поддержки русских символов

SetConsoleOutputCP(1251);

Spisok A; // Создаем объект класса Spisok

node\* Res;

int el, el1;

A.Postroenie(); // Выполняем построение списка

A.VyvodForward(); // Выводим список в прямом направлении

A.VyvodBack(); // Выводим список в обратном направлении

cout << "Введите элемент звена, после которого осуществляется вставка\n";

cin >> el; // Вводим элемент звена

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

cin >> el1; // Вводим элемент вставляемого звена

Res = A.PoiskForward(el); // Ищем узел с заданным элементом в прямом направлении

if (Res != NULL) {

// Если узел найден и != NULL, выполняем вставку после него

A.InsAfter(el1, Res);

A.VyvodForward(); // Вывод в прямом порядке

A.VyvodBack(); // Вывод в обратном порядке

}

else {

cout << "Звена с заданным элементом в списке нет!\n";

}

cout << "Введите элемент звена, перед которым осуществляется вставка\n";

cin >> el; // Вводим элемент звена, перед которым будет осуществляться вставка

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

cin >> el1; // Вводим элемент вставляемого звена

Res = A.PoiskBack(el); // Ищем узел с заданным элементом в обратном направлении

if (Res != NULL) { // Если узел найден и != NULL, выполняем вставку перед ним

A.InsBefore(el1, Res);

A.VyvodForward();

A.VyvodBack();

}

else {

cout << "Звена с заданным элементом в списке нет!\n";

}

cout << "Введите элемент звена, после которого осуществляется удаление\n";

cin >> el; // Вводим элемент звена, после которого будет осуществляться удаление

Res = A.PoiskForward(el); // Ищем узел с заданным элементом в прямом направлении

if (Res != NULL) { // Если узел найден и != NULL, удаляем узел, следующий за ним

A.DelAfter(Res);

A.VyvodForward();

A.VyvodBack();

}

else {

cout << "Звена с заданным элементом в списке нет!\n";

}

cout << "Введите элемент звена, которое надо удалить\n";

cin >> el; // Вводим элемент звена, которое нужно удалить

Res = A.PoiskForward(el); // Ищем узел с заданным элементом в прямом направлении

if (Res != NULL) { // Если узел найден и !=NULL, удаляем его

A.Delete(Res);

A.VyvodForward();

A.VyvodBack();

}

else {

cout << "Звена с заданным элементом в списке нет!\n";

}

A.Ochistka(); // Очищаем список

cout << '\n';

}

void Spisok::Postroenie() {

node\* rsp; // Указатель на текущий узел списка

int el; // Значение элемента списка

nsp = new node; // Выделение памяти

rsp = nsp; // Присваиваем rsp адрес nsp

(\*nsp).pred = NULL; // Устанавливаем указатель на предыдущий узел равным NULL

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

cout << "Вводите последовательность (признаком окончания должен служить 0)\n";

cin >> el;

while (el != 0) {

(\*rsp).sled = new node; // Выделяем память для след. узла

(\*(\*rsp).sled).pred = rsp; // Устанавливаем указатель на предыдущий узел нового узла равным rsp

rsp = (\*rsp).sled; // Обновляем rsp на следующий узел

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

(\*rsp).elem = el; // Присваиваем новому узлу значение элемента списка

cin >> el; // Считываем следующий элемент списка

}

ksp = rsp; // Присваиваем ksp адрес последнего узла списка

}

void Spisok::VyvodForward() {

node\* rsp; // Указатель на текущий узел списка

rsp = (\*nsp).sled; // Присваиваем rsp адрес следующего узла после начального узла

cout << "Двунаправленный список содержит: \n";

while (rsp != NULL) {

cout << (\*rsp).elem << ' '; // Выводим значение элемента текущего узла

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

}

cout << '\n';

}

void Spisok::VyvodBack() {

node\* rsp; // Указатель на текущий узел списка

rsp = ksp; // Присваиваем rsp адрес последнего узла списка

cout << "Двунаправленный список в обратном порядке: \n";

while ((\*rsp).pred != NULL) {

cout << (\*rsp).elem << ' '; // Выводим значение элемента текущего узла

rsp = (\*rsp).pred; // Переходим к предыдущему узлу списка

}

cout << '\n';

}

node\* Spisok::PoiskForward(int el) {

node\* q; // Указатель на текущий узел списка

node\* Res; // Указатель на узел с найденным элементом

Res = NULL; // Инициализируем указатель Res значением NULL

q = (\*nsp).sled; // Присваиваем q адрес следующего узла после начального узла

while (q != NULL && Res == NULL) {

if ((\*q).elem == el) {

Res = q; // Найден узел с элементом el, присваиваем его адрес указателю Res

break; // Прекращаем выполнение цикла

}

else {

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

}

}

return Res; // Возвращаем найденный узел или NULL, если элемент не найден

}

node\* Spisok::PoiskBack(int el) {

node\* q; // Указатель на текущий узел списка

node\* Res; // Указатель на узел с найденным элементом

Res = NULL; // Инициализируем указатель Res значением NULL

q = ksp; // Присваиваем q адрес последнего узла списка

while (q != NULL && Res == NULL) {

if ((\*q).elem == el) {

Res = q; // Найден узел с элементом el, присваиваем его адрес указателю Res

break; // Прекращаем выполнение цикла

}

else {

q = (\*q).pred; // Переходим к предыдущему узлу списка

}

}

return Res; // Возвращаем найденный узел или NULL, если элемент не найден

}

void Spisok::InsAfter(int el, node\* Res) {

node\* q; // Указатель на текущий узел списка

q = new node; // Выделяем память под узел

(\*q).elem = el; // Присваиваем узлу значение элемента el

if ((\*Res).sled != NULL) { // Если у узла Res есть следующий узел

(\*q).sled = (\*Res).sled; // Устанавливаем следующий узел на следующий узел Res

(\*q).pred = (\*(\*Res).sled).pred; // Устанавливаем предыдущий узел нового узла на предыдущий узел следующего узла Res

(\*(\*Res).sled).pred = q; // Устанавливаем предыдущий узел следующего узла Res на новый узел q

(\*Res).sled = q; // Устанавливаем следующий узел Res на новый узел q

}

else { // Если у узла Res нет следующего узла

(\*q).sled = NULL; // Устанавливаем следующий узел на NULL

(\*q).pred = Res; // Устанавливаем предыдущий узел на Res

ksp = q; // Обновляем указатель на последний узел списка (ksp) на узел q

(\*Res).sled = q; // Устанавливаем следующий узел Res на узел q

}

}

void Spisok::InsBefore(int el, node\* Res) {

node\* q; // Указатель на узел списка

q = new node; // Выделяем память под q

(\*q).elem = el; // Присваиваем узлу значение элемента el

(\*q).sled = (\*(\*Res).pred).sled; // Устанавливаем следующий узел q на следующий узел предыдущего узла Res

(\*q).pred = (\*Res).pred; // Устанавливаем предыдущий узел q на предыдущий узел Res

(\*(\*Res).pred).sled = q; // Устанавливаем следующий узел предыдущего узла Res на узел q

(\*Res).pred = q; // Устанавливаем предыдущий узел Res на узел q

}

void Spisok::Delete(node\* Res) {

if ((\*Res).sled != NULL) {

(\*(\*Res).sled).pred = (\*Res).pred; // Устанавливаем предыдущий узел следующего узла Res на предыдущий узел Res

(\*(\*Res).pred).sled = (\*Res).sled; // Устанавливаем следующий узел предыдущего узла Res на следующий узел Res

delete Res; // Удаляем узел Res

}

else {

(\*(\*Res).pred).sled = NULL; // Устанавливаем следующий узел предыдущего узла Res на NULL

ksp = (\*ksp).pred; // Обновляем указатель на последний узел списка (ksp) на предыдущий узел от ksp

delete Res; // Удаляем узел Res

}

}

void Spisok::DelAfter(node\* Res) {

node\* q;

if ((\*Res).sled == NULL) { // Если следующией равен NULL

cout << "Указано последнее звено\n";

}

else {

if ((\*(\*Res).sled).sled != NULL) {

q = (\*Res).sled;

(\*(\*(\*Res).sled).sled).pred = Res; // Устанавливаем предыдущий узел следующего узла следующего узла Res на Res

(\*Res).sled = (\*(\*Res).sled).sled; // Устанавливаем следующий узел Res на следующий узел следующего узла Res

delete q; // Удаляем узел следующего узла

}

else {

q = (\*Res).sled;

(\*Res).sled = NULL; // Устанавливаем следующий узел Res на NULL

ksp = (\*ksp).pred; // Обновляем указатель на последний узел списка (ksp) на предыдущий узел от ksp

delete q; // Удаляем узел следующего узла

}

}

}

void Spisok::Ochistka() {

node\* q, \* q1;

q = nsp; // Устанавливаем q на первый узел списка

q1 = (\*q).sled; // Устанавливаем q1 на следующий узел после первого узла

while (q1 != NULL) {

q = q1; // Перемещаем q на следующий узел

q1 = (\*q1).sled; // Перемещаем q1 на следующий узел после q

delete q; // Удаляем узел q

}

delete nsp; // Удаляем первый узел списка

nsp = ksp = NULL; // Обнуляем указатели на первый и последний узлы списка

}

