Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Реферат

По дисциплине «Основы алгоритмизации и программирования»

На тему «Тестирование кода двусвязного списка»

Выполнила:

Студент(ка) 1 курса 7 группы

Подшиваленко Диана Игоревна

Проверил:

Белодед Николай Иванович

2024, Минск

#include <iostream>

#include <Windows.h>

using namespace std;

struct node { // структура узла списка

int elem; // информационное поле

node\* sled; // указатель на следующий элемент

node\* pred; // указатель на предыдущий элемент

};

class Spisok { // класс двустороннего списка

private: // приватная область

node\* nsp, \* ksp; // указатели на начало и конец списка

public: // публичная область

Spisok() { // конструктор списка

nsp = ksp = NULL; // изначально указатели на начало и на конец списка NULL

}

void Postroenie(); // прототип функции построения списка

void VyvodForward(); //прототип функции вывода с начала списка

void VyvodBack(); // прототип функции вывода с конца списка

void Ochistka(); // прототип функции очистки

void InsAfter(int, node\*); // прототип функции вставки элемента после введенного

void InsBefore(int, node\*); // прототип функции для вставки элемента перед введенным

void Delete(node\*); // прототип функции удаления заданного элемента

void DelAfter(node\*); // прототип функции удаления элемента после введенного

node\* PoiskForward(int); // прототип функции поиск элемента с начала списка

node\* PoiskBack(int); // прототип функции поиска элемента с конца списка

};

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

Spisok A; // создаем объект класса Spisok

node\* Res; // буферный указатель

int el, el1; // две буферные переменные для информационных полей

A.Postroenie(); // вызываем функцию построения списка

A.VyvodForward(); // вызываем функцию вывода элементов с начала списка

A.VyvodBack(); // вызываем функцию вывода элементов с конца списка

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

cin >> el; // считываем элемент, после которого будем вставлять новый элемент

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

cin >> el1; // считываем элемент, который будем вставлять

Res = A.PoiskForward(el); // присваиваем буферному указателю результат поиска элемента с начала списка

if (Res != NULL) { // если найден

A.InsAfter(el1, Res); // вызываем функцию вставки после найденного элемента

A.VyvodForward();// вызываем функцию построения списка

A.VyvodBack();// вызываем функцию вывода элементов с конца списка

}

else {// иначе

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

}

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

cin >> el; // считываем элемент, перед которым хотим вставить новый элемент

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

cin >> el1; // считываем элемент, который хотим вставить

Res = A.PoiskBack(el); // присваиваем буферному указателю результат поиска элемента с конца списка

if (Res != NULL) {// если найден

A.InsBefore(el1, Res); // вызываем функцию вставки перед найденным элементом

A.VyvodForward(); // вызываем функцию вывода элементов с начала списка

A.VyvodBack();// вызываем функцию вывода элементов с конца списка

}

else {

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

}

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

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

Res = A.PoiskForward(el);// присваиваем буферному указателю результат поиска элемента с начала списка

if (Res != NULL) { // если найден

A.DelAfter(Res); // вызываем функцию удаления элемента после выбранного

A.VyvodForward(); // вызываем функцию вывода элементов с начала списка

A.VyvodBack();// вызываем функцию вывода элементов с конца списка

}

else {

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

}

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

cin >> el; // считываем элемент, который хотим удалить

Res = A.PoiskForward(el); // присваиваем буферному указателю результат поиска элемента с начала списка

if (Res != NULL) { // если найден

A.Delete(Res); // вызываем функцию удаления заданного элемента

A.VyvodForward(); // вызываем функцию вывода элементов с начала списка

A.VyvodBack();// вызываем функцию вывода элементов с конца списка

}

else {

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

}

A.Ochistka(); // вызываем функцию очистки списка

cout << '\n';

}

void Spisok::Postroenie() { // функция построения списка

node\* rsp; // буферный указатель на элемент списка

int el; // буферная переменная для считывания числа

nsp = new node; // выделяем память под начальный элемент списка

rsp = nsp; // присваиваем буферному указателю адрес первого элемента списка

(\*nsp).pred = NULL; // присваиваем указателю на предыдущий элемент NULL

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

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

cin >> el; // считываем элемент пользователя

while (el != 0) { // в цикле: пока элемент не равен 0(признак окончания последовательности)

(\*rsp).sled = new node; // выделяем память для следующего элемента

(\*(\*rsp).sled).pred = rsp; // для следующего элемента присваиваем указателю на предыдущий элемент адрес указателя rsp

rsp = (\*rsp).sled; // присваиваем указателю rsp адрес следующего элемента

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

(\*rsp).elem = el; // в информационное поле текущего элемента записываем введенный пользователем элемент

cin >> el; // считываем следующий

}

ksp = rsp; // присваиваем указателю на последний элемент адрес указателя rsp

}

void Spisok::VyvodForward() { // функция для вывода элементов списка, начиная с начала

node\* rsp; // буферный указатель

rsp = (\*nsp).sled; // присваиваем указателю rsp адрес элемента, следующего за первым

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

while (rsp != NULL) { // пока указатель rsp не NULL

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

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

}

cout << '\n';

}

void Spisok::VyvodBack() { // функция для вывода элементов списка, начиная с конца

node\* rsp; // буферный указатель

rsp = ksp; // присваиваем указателю rsp адрес последнего элемента

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

while ((\*rsp).pred != NULL) { // пока предыдущий элемент элемента, на который указывает rsp не NULL

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

rsp = (\*rsp).pred; // переходим на предыдущий элемент

}

cout << '\n';

}

node\* Spisok::PoiskForward(int el) { // функция для поиска заданного элемента, начиная с начала

node\* q; // буферный указатель

node\* Res; // указатель под искомый элемент

Res = NULL; // изначально присваиваем ему значение NULL

q = (\*nsp).sled; // присваиваем указателю q адрес элемента, следующего за начальным

while (q != NULL && Res == NULL) { // пока q не равен NULL(то есть не дошли до конца) и пока Res равен NULL(то есть не нашли нужный элемент)

if ((\*q).elem == el) { // проверяем информационное поле элемента на сходство с искомым и, если они равны, то

Res = q; // присваиваем указателю Res адрес искомого элемента

break; // выходим из цикла

}

else { // иначе

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

}

}

return Res; // возвращаем указатель на найденный элемент

}

node\* Spisok::PoiskBack(int el) { // функция для поиска элемента, начиная с конца

node\* q; // буферный указатель

node\* Res; // указатель под искомый элемент

Res = NULL;// изначально присваиваем ему значение NULL

q = ksp;// присваиваем указателю q адрес последнего элемента

while (q != NULL && Res == NULL) {// пока q не равен NULL(то есть не дошли до начала) и пока Res равен NULL(то есть не нашли нужный элемент)

if ((\*q).elem == el) {// проверяем информационное поле элемента на сходство с искомым и, если они равны, то

Res = q;// присваиваем указателю Res адрес искомого элемента

break; // выходим из цикла

}

else {

q = (\*q).pred; // переходим на предыдущий элемент

}

}

return Res;// возвращаем указатель на найденный элемент

}

void Spisok::InsAfter(int el, node\* Res) { // функция для вставки элемента перед элементом, на который указывает Res

node\* q; // указатель на новый элемент

q = new node; // выделяем память под него

(\*q).elem = el; // присваиваем информационному полю элемента, на который указывает q, введенное значение

if ((\*Res).sled != NULL) { // если элемент, после которого должны вставить новый элемент последний, то

(\*q).sled = (\*Res).sled; // присваиваем указателю нового элемента на следующий элемент адрес указателя следующего элемента, после которого должны вставить элемент

(\*q).pred = (\*(\*Res).sled).pred; // присваиваем указателю на предыдущий элемент нового элемента адрес элемента, который является предыдущим элементом следующего элемента после нужного

(\*(\*Res).sled).pred = q; // присваиваем указателю, который указывает на предыдущий элемент следующего элемнета адрес, на который указывает q

(\*Res).sled = q; // присваиваем указателю, который указывает на следующий элемент, после которого хотим вставить элемент, адрес элемента, на который указывает q

}

else { // иначе

(\*q).sled = NULL; // указателю на следующий элемент после нового присваиваем NULL

(\*q).pred = Res; // указателю на предыдущий элемент перед новым присваиваем адрес элемента, после которого должны вставить элемент

ksp = q; // присваиваем указателю на последний элемент списка адрес элемента, на который указывает q

(\*Res).sled = q; // указатель нужного элемента ссылается на новый

}

}

void Spisok::InsBefore(int el, node\* Res) { // функция для вставки элемента перед заданным

node\* q; // указатель на новый элемент

q = new node; // выделяем память под него

(\*q).elem = el; // присваиваем информационному полю элемента, на который указывает q, введенное значение

(\*q).sled = (\*(\*Res).pred).sled; // присваиваем указателю на следующий элемент нового элемента адрес элемента, который является следующим элементом предыдущего элемента, перед которым должны вставить элемент

(\*q).pred = (\*Res).pred; // присваиваем указателю на предыдущий элемент нового элемента адрес элемента, который является предыдущим элементом элемента, перед которым должны вставить элемент

(\*(\*Res).pred).sled = q; // присваиваем указателю на следующий элемент предыдущего элемента адрес q

(\*Res).pred = q; // присваиваем указателю на предыдующий элемент адрес q

}

void Spisok::Delete(node\* Res) { // функция для удаления заданного элемента

if ((\*Res).sled != NULL) { // если заданный элемент не последний

(\*(\*Res).sled).pred = (\*Res).pred; // присваиваем указателю на предыдущий элемент следующего элемента адрес указателя на предыдущий элемент нужного

(\*(\*Res).pred).sled = (\*Res).sled; // присваиваем указателю на следующий элемент предыдущего элемента адрес указателя на следующий элемент нужного

delete Res; // очищаем память удаляемого элемента

}

else {// иначе

(\*(\*Res).pred).sled = NULL; // присваиваем указателю на следующий элемент предыдущего элемента NULL

ksp = (\*ksp).pred; // присваиваем указателю на последний элемент адрес элемента, который идет перед последним

delete Res;// очищаем память удаляемого элемента

}

}

void Spisok::DelAfter(node\* Res) { // функция для удаления элемента, который находится после введенного

node\* q; // указатель для удаляемого элемента

if ((\*Res).sled == NULL) { // если следующего элемента нет

cout << "Указано последнее звено\n"; // выводим сообщение

}

else {// иначе

if ((\*(\*Res).sled).sled != NULL) { //если указатель на следующий элемент следующего элемента не NULL, то

q = (\*Res).sled; // присваиваем указателю q адрес элемента, следущего за нужнымнн

(\*(\*(\*Res).sled).sled).pred = Res; // присваиваем указателю на предыдущий элемент указателя на следующий элемент следующего элемента адрес Res

(\*Res).sled = (\*(\*Res).sled).sled; // присваиваем указателю следующего элемента адрес следующего элемента, следующего за Res

delete q; // очищаем память удаляемого элемента

}

else { // иначе

q = (\*Res).sled; // присваиваем указателю на удаляемый элемент адрес элемента, следующего за Res

(\*Res).sled = NULL; // присваиваем следующему элементу после Res NULL

ksp = (\*ksp).pred; // предпоследний элемнет становится последним

delete q;// очищаем память удаляемого элемента

}

}

}

void Spisok::Ochistka() { // функция очистки списка

node\* q, \* q1; // буферные указатели для очистки

q = nsp; // присваиваем указателю q адрес первого элемента

q1 = (\*q).sled; // присваиваем q1 адрес элемента, следующего за первым

while (q1 != NULL) { // пока q1 не NULL

q = q1; // присваиваем адрес q1 указателю q

q1 = (\*q1).sled; // присваиваем указателю q1 адрес следующего элемента

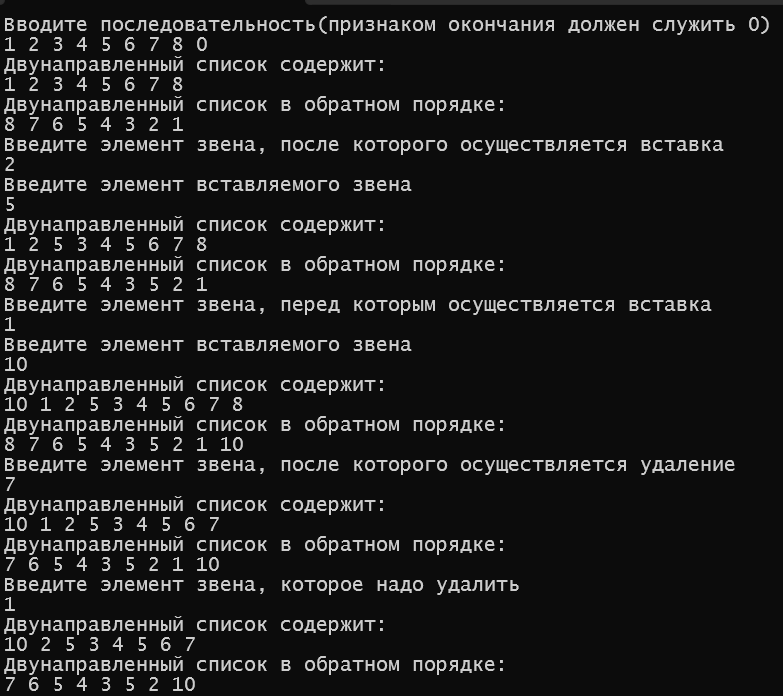
delete q; // очищаем память

}

delete nsp; // очищаем память начального элемента

nsp = ksp = NULL; // начальный и последний эллемент NULL

}

Результат выполнения: