МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет: Информационных технологий

Кафедра: Программной инженерии

Тестирование кодa

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

Содержание:

1. Код Тестивания с комментариями…………………3
2. Результат выполнения программы………………..9

Код Тестирования:

#include <iostream>

using namespace std;

struct node // структура, представляет элемент двунаправленного списка

{

int elem; // переменная, хранящая значение элемента списка

node\* sled; // указатель на следующий элемент списка

node\* pred; // указатель на предыдущий элемент списка

};

class Spisok

{

private:

node\* nsp; // указатель на начало списка

node\* ksp;// указатель на конец списка

public:

Spisok() { nsp = ksp = NULL; } // обозначение пустого списока

void Postroenie(); /\*Этот метод создает двунаправленный список.

Пользователь вводит последовательность чисел,

которая затем добавляется в список,

пока пользователь не введет ноль(который означает конец ввода)

Здесь создается новое звено списка и заполняется его элементом.\*/

void VyvodForward(); // Этот метод выводит содержимое списка в прямом направлении (от начала к концу)

void VyvodBack(); // Этот метод выводит содержимое списка в обратном направлении (от конца к началу)

void Ochistka(); // Этот метод очищает память, выделенную для списка

void InsAfter(int, node\*);// Этот метод вставляет новое звено после указанного узла списка

void InsBefore(int, node\*);// Этот метод вставляет новое звено перед указанным узлом списка

void Delete(node\*);// Этот метод удаляет указанный узел из списка

void DelAfter(node\*);// Этот метод удаляет узел, следующий за указанным узлом.

node\* PoiskForward(int);// Этот метод выполняет поиск элемента в списке в прямом направлении,

//начиная с начала списка. Он возвращает указатель на найденный элемент или NULL, если элемент не найден.

node\* PoiskBack(int); // Этот метод выполняет поиск элемента в списке в обратном направлении, начиная с конца списка.

// Он также возвращает указатель на найденный элемент или NULL, если элемент не найден

};

void main()

{

setlocale(0, "");

Spisok A; // создание объекта

node\* Res; // объявление указателя

int el; // объявление переменной для ввода звена вставки

int el1 = 0; // объявление переменных для вставляемого элемнта звена

A.Postroenie(); // построение двунаправленного списка.

A.VyvodForward();// выводод содержимого списка в прямом порядке

A.VyvodBack(); // выводод содержимого списка в обратном порядке

cout << "Введите элемент звена, после которого ";

cout << "осуществляется вставка: ";

cin >> el;

cout << "Введите элемент вставляемого звена: ";

cin >> el1;

Res = A.PoiskForward(el); // Вызывается метод для поиска элемента с заданным значением el в списке

if (Res != NULL) // Если элемент найден

{

A.InsAfter(el1, Res); // то вызывается метод InsAfter(el1, Res), который вставляет новый элемент с значением el1 после найденного элемента Res.

A.VyvodForward();// выводод содержимого списка в прямом порядке

A.VyvodBack(); // выводод содержимого списка в обратном порядке

}

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

cout << "Введите элемет звена, перед которым ";

cout << "осуществляется вставка";

cin >> el;

cout << "Введите элемент вставляемого звена: ";

cin >> el1;

Res = A.PoiskBack(el);

if (Res != NULL) // Если элемент найден

{

A.InsAfter(el1, Res); // то вызывается метод InsAfter(el1, Res), который вставляет новый элемент с значением el1 после найденного элемента Res.

A.VyvodForward();// выводод содержимого списка в прямом порядке

A.VyvodBack(); // выводод содержимого списка в обратном порядке

}

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

cout << "Введите элемент звена, после которого ";

cout << "осуществляется удаление ";

cin >> el;

Res = A.PoiskForward(el);// метод для поиска элемента списка с заданным значением el

if (Res != NULL) { // Если элемент найден

A.DelAfter(Res); // удаление найденого элемента из списка.

A.VyvodForward();// выводод содержимого списка в прямом порядке

A.VyvodBack(); // выводод содержимого списка в обратном порядке

}

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

cout << "Введите элемет звена, которое ";

cout << "надо удалить: ";

cin >> el;

Res = A.PoiskForward(el);// метод для поиска элемента списка с заданным значением el

if (Res != NULL) { // Если элемент найден

A.Delete(Res); // удаление найденого элемента из списка.

A.VyvodForward();// выводод содержимого списка в прямом порядке

A.VyvodBack(); // выводод содержимого списка в обратном порядке

}

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

A.Ochistka();// Освобождение памяти, занимаемую списком.

cout << "\n";

system("PAUSE"); // Приостановление выполнения программы, пока пользователь не нажмет клавишу для завершения

}

void Spisok::Postroenie()

{ // Построение двунаправленного списка с заглавным звеном

// nsp - указатель на начало списка

// ksp - указатель на конец списка

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

int el; // переменная для хранения элементов, вводимых пользователем.

nsp = new(node); // Cоздается новый узел и указатель nsp устанавливается на этот узел.

rsp = nsp; // rsp также указывает на этот узел тк он текущий

(\*nsp).pred = NULL; // Два указателя нового узла устанавливаются в NULL,

(\*nsp).sled = NULL; // потому что это начало списка, и пока он не содержит других элементов.

cout << "Вводите последовательность: \n";

cin >> el;

while (el != 0) { // будет выполняться до тех пор, пока пользователь не введет 0.

(\*rsp).sled = new(node); // Создает новый узел и присваивает указателю sled текущего узла (rsp) указатель на новый узел

(\*((\*rsp).sled)).pred = rsp; // Устанавливает указатель pred нового узла на текущий узел (rsp), что делает новый узел последующим для rsp

rsp = (\*rsp).sled; // Сдвигает указатель rsp на новый узел,теперь rsp указывает на новый узел, который был только что добавлен в список

(\*rsp).sled = NULL; // Потому что он пока является последним узлом списка.

(\*rsp).elem = el; // Запись введенного числа

cin >> el; // Ввод следующего значения

}

ksp = rsp; // указатель ksp устанавливается на последний созданный узел списка, чтобы указывать на его конец

}

void Spisok::VyvodForward()

{// Вывод содержимого двунаправленного списка от его начала

// nsp - указатель на начало списка

// ksp - указатель на конец списка

node\* rsp;

rsp = (\*nsp).sled;

cout << "Двунаправленный список содержит: ";

while (rsp != NULL)

{

cout << (\*rsp).elem << " ";

rsp = (\*rsp).sled;

}

cout << endl;

}

void Spisok::VyvodBack()

{// Вывод содержимого двунаправленного списка от его конца

// nsp - указатель на начало списка

// ksp - указатель на конец списка

node\* rsp; // Объявляет указатель rsp

rsp = ksp;// Устанавливает указатель rsp на последний узел списка

cout << "Двунаправленнй список в обратном порядке: ";

while ((\*rsp).pred != NULL) { // пока текущий узел не является заглавным узлом списка

cout << (\*rsp).elem << " "; // Выводит значение элемента текущего узла.

rsp = (\*rsp).pred; // Перемещает указатель rsp на предыдущий узел списка

}

cout << endl;

}

node\* Spisok::PoiskForward(int el)

// Функция возвращает указатель на найденный элемент el

// Двунаправленного списка, заданного указателями nsp

// и ksp, и NULL, если элемент в списке не найден.

{

node\* q; // Объявляем указатель q на узел списка

node\* Res; // Объявляем указатель Res, будет использоваться для хранения найденного элемента.

Res = NULL; // пока элемент еще не найден

q = (\*nsp).sled; // Начинает поиск с первого реального элемента списка,т.е. пропуская заголовочный узел

while (q != NULL && Res == NULL) // будет выполняться до тех пор, пока не будет достигнут конец списка и пока не будет найден элемент (Res == NULL)

if ((\*q).elem == el) // Проверяет, равно ли значение текущего узла el

{

Res = q; // означает, что элемент был найден

}

else q = (\*q).sled; // Если элемент не равен элемету списка , переходит к следующему элементу списка

return Res; // Возвращает указатель на найденный элемент или NULL, если элемент не был найден в списке

}

node\* Spisok::PoiskBack(int el)

// Функция возвращает указатель на найденный элемент el

// Двунаправленного списка, заданного указателями nsp

// и ksp, и NULL, если элемент в списке не найден.

{

node\* q; // Объявляем указатель q на узел списка

node\* Res; // Объявляем указатель Res, будет использоваться для хранения найденного элемента.

Res = NULL; // пока элемент еще не найден

q = ksp; // Начинает поиск с последнего реального элемента списка

while (q != NULL && Res == NULL) // будет выполняться до тех пор, пока не будет достигнут конец списка и пока не будет найден элемент (Res == NULL)

if ((\*q).elem == el) {// Проверяет, равно ли значение текущего узла el

Res = q; // означает, что элемент был найден

}

else q = (\*q).pred; // Если текущий элемент не равен данному пользователем, переходит к предыдущему элементу списка

return Res; // Возвращает указатель на найденный элемент или NULL, если элемнт не был найден в списке

}

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

// Вставка звена с информационным полем el в

// в Двунаправленный список, заданный указателями

// nsp и ksp, после звена, на которое указывает Res.

{

node\* q; // Объявляет указатель q на новый узел

q = new(node); // Создаем новый узел и присваивает его указателю q

(\*q).elem = el; // Устанавливает значение elem поля нового узла равным el.

if ((\*Res).sled != NULL) // Проверяет, существует ли следующий узел после указанного узла Res

{

(\*q).sled = (\*Res).sled; // Устанавливаем указатель sled нового узла так, чтобы он указывал на тот же узел, на который указывает sled узла Res

(\*q).pred = (\*Res).sled->pred; // Устанавливаем указатель pred нового узла, чтобы он указывал на предыдущий узел того же узла, на который указывает sled узла Res

(\*Res).sled->pred = q; // Устанавливаем указатель pred для узла, следующего за указанным узлом Res, так, чтобы он указывал на новый узел q

(\*Res).sled = q; // Устанавливем указатель sled для указанного узла Res, теперь он указывает на новый узел q.

}

else // Если указанный узел Res является последним узлом в списке:

{

(\*q).sled = NULL; // Устанавливаем указатель sled нового узла равным NULL, тк новый узел станет последним в списке

(\*q).pred = Res; // Устанавливаем указатель pred нового узла, чтобы он указывал на узел Res

ksp = q; // тк новый узел станет последним в списке. ksp указывает на последний элемент

(\*Res).sled = q; // Уст. указатель sled для указанного узла Res, чтобы он указывал на новый узел q

}

}

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

// Вставка звена с информационным полем el в

// в Двунаправленный список, заданный указателями

// nsp и ksp, перед звеном, на которое указывает Res.

{

node\* q;

q = new(node); // Создает новый узел и присваивает его указателю q

(\*q).elem = el; // Устанавливаем значение списка равным el

(\*q).sled = (\*Res).pred->sled; // Устанавливаем указатель sled нового узла, чтобы он указывал на следующий узел после узла (перед которым происходит вставка)

(\*q).pred = (\*Res).pred; // Устанавливаем указатель pred нового узла так, чтобы он указывал на узел, перед которым происходит вставка

(\*Res).pred->sled = q; // Устанавливаем указатель sled для узла, предшествующего указанному узлу Res так чтобы он указывал на новый узел q

(\*Res).pred = q; // Устанавливаем указатель pred для указанного узла Res, чтобы он указывал на новый узел q

}

void Spisok::Delete(node\* Res)

// Удаление звена из двунаправленного списка.

// nsp - указатель на начало списка

// ksp - указатель на конец списка

// Res - указатель на удаляеиое звено

{

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

{

(\*(\*Res).sled).pred = (\*Res).pred;

(\*(\*Res).pred).sled = (\*Res).sled;

delete Res;

}

else

{

(\*(\*Res).pred).sled = NULL;

ksp = (\*ksp).pred;

delete Res;

}

}

void Spisok::DelAfter(node\* Res)

// Удаление звена из двунаправленного списка.

// nsp - указатель на начало списка

// ksp - указатель на конец списка

// Res - указатель на звено, предыдущее удаляемому

{

node\* q;

if ((\*Res).sled != NULL) // Проверяем, существует ли у удаляемого узла следующий узел

{

(\*(\*Res).sled).pred = (\*Res).pred; // устанавливаем указатель pred следующего узла так, чтобы он указывал на узел, предшествующий удаляемому узлу.

(\*(\*Res).pred).sled = (\*Res).sled; // Устанавливаем указатель sled для узла, предшествующего удаляемому узлу, так чтобы он указывал на узел, следующий за удаляемым узлом

delete Res; // удаление узла

}

else // Если следующий узел не существует

{

(\*(\*Res).pred).sled = NULL; // Установка указатель sled для узла, предшествующего удаляемому узлу, равным NULL, чтобы удалить связь между предыдущим и удаляемым узлом

ksp = (\*ksp).pred; //Устанавливаем указатель ksp на конец списка, чтобы он указывал на предыдущий узел

delete Res;// Удаляет узел

}

}

void Spisok::Ochistka()

// Удаление звена из двунаправленного списка.

// nsp - указатель на начало списка

// ksp - указатель на конец списка

// Res - указатель на звено, предыдущее удаляемому

{

node\* q, \* q1;// Объявляет два указателя на узлы q и q1

q = nsp; // Устанавливаем указатель q на нач звено списка

q1 = (\*q).sled; // Устанавливаем указатель q1 на следующее звено списка

while (q1 != NULL)// пока q1 не станет равным NULL, т.е. пока не будет достигнут конец списка

{

q = q1; // Перемещаем указатель q на следующее звено списка

q1 = (\*q1).sled; // Перемещаем указатель q1 на следующее звено списка

delete q; // Удаляет текущее звено списка.

}

delete nsp;// Удаляет начальное звено списка.

nsp = ksp = NULL; // обнуление указателей

}

