МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет: Информационных технологий

Кафедра: Программной инженерии

Выполнила: студентка 1 курса 5 группы

специальности ПОИТ Дзивнель М.А.

Проверил: преподаватель

Белодед Николай Иванович

**ТЕСТИРОВАНИЕ КОДА**

По дисциплине “Основы алгоритмизации и программирования”

Минск

2023

СОДЕРЖАНИЕ

Введение3

Программа 14

Программа 27

Программа 314

**Введение**

В данном документе демонстрируется тестирование программ, которые реализуют следующие алгоритмы: удаление звена из однонаправленного списка с заглавным звеном, реализация основных операций над ортогональными списками, включение звена в кольцо с удаленным заглавным звеном, удаление звена с заданным значением информационного поля из кольца с удаленным заглавным звеном.

**Программа 1**

Код программы, который осуществляет удаление звена из однонаправленного списка с заглавным звеном:

#include <iostream>

using namespace std;

struct node // объявление структуры node

{

int elem; // поле элемента списка

node\* sled; // указатель на звено следующего элемента списка

};

class Spisok // объявление класса Spisok (для работы со списком)

{

private:

node\* phead, \* Res; // указатели на начало(голову) списка и текущее положение(звено) списка

public:

Spisok() { phead = new(node); Res = NULL; } // конструктор класса Spisok

~Spisok() { delete phead; } // деструктор класса Spisok

void POSTROENIE(); // функция для построения списка

void VYVOD(); // функция для вывода списка

node\* POISK(int); // функция для поиска звена по элементу

void YDALE(); // функция для удаления звена(звено стоит после элемента, на кот. указывает ссылка)

void YDALE1(); // функция для удаления звена(звено, на которое указывает ссылка)

void OCHISTKA(); // функция для очистки памяти

};

void main() // главная функция программы

{

setlocale(LC\_ALL, "Rus"); // установка русского языка в консоли

Spisok A; // создание объекта класса Spisok

int el; // переменная элемента списка

node\* Res\_Zn; // указатель на текущее звено списка

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

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

cout << "\nВведите элемент звена, после которого ";

cout << "осуществляется удаление:\n";

cin >> el; // ввод элемента для удаления

Res\_Zn = A.POISK(el); // поиск звена по элементу

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

{

A.YDALE(); A.VYVOD(); // вызов функции для удаления звена и функции для вывода списка

}

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

cout << "\nВведите удаляемый элемент звена:\n";

cin >> el; // ввод элемента для удаления

Res\_Zn = A.POISK(el); // поиск звена по элементу

if (A.POISK(el) != NULL)

{

A.YDALE1(); A.VYVOD(); cout << endl; // вызов функции для удаления звена, функции для вывода списка и вывода символа перев

}

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

A.OCHISTKA(); // вызов метода OCHISTKA

cout << "\n";

system("PAUSE");

}

void Spisok::POSTROENIE()

//построение однонаправленного списка с заглавным звеном.

//phead - указатель на заглавное звено

{

node\* t; //объявление указателя на звено списка.

int el; //объявление переменной для хранения вводимых значений элементов списка.

t = phead; (\*t).sled = NULL; //устанавливаем указатель t на заглавное звено и присваиваем полю следующее значение NULL, т.к. список пока пуст.

cout << "Вводите элементы звеньев списка: ";

cin >> el; //вводим элемент списка.

while (el != 0) {

(\*t).sled = new (node); //создаем новое звено списка и присваиваем его полю следующее предыдущего звена.

t = (\*t).sled; //перемещаем указатель t на новое звено.

(\*t).elem = el; //присваиваем полю элемента списка значение введенного числа.

(\*t).sled = NULL; //присваиваем полю следующее значение NULL, т.к. это последний элемент списка.

cin >> el; //считываем следующий элемент списка.

}

}

void Spisok::VYVOD()

//вывод содержимого однонаправленного списка.

//phead - указатель на заглавное звено.

{

node\* t;//объявление указателя на звено списка.

t = phead; t = (\*t).sled; // инициализация t первым звеном списка

cout << "Список: ";

while (t != NULL) // пока не достигнут конец списка

{

cout << (\*t).elem << " ";// вывод элемента текущего звена

t = (\*t).sled; //переходим к следующему звену

}

}

node\* Spisok::POISK(int el)

//поиск звена с элементом el в списке, заданном указателем phead

//в случае успешного поиска в Res находится адрес искомого звена

//списка. В противном случае Res содержит NULL.

{

node\* t;

Res = NULL; t = phead; t = (\*t).sled; // инициализация t первым звеном списка

while (t != NULL && Res == NULL)// пока не достигнут конец списка и не найден искомый элемент

if ((\*t).elem == el) Res = t;// если текущий элемент равен искомому, сохраняем адрес звена

else t = (\*t).sled;// переходим к следующему звену

return Res; // возвращаем адрес звена с искомым элементом или NULL, если элемент не найден

}

void Spisok::YDALE()

//удаление звена, расположенного после звена,

//на которое указывает ссылка Res.

{

node\* q;

q = (\*Res).sled;//указатель на звено, расположенное после звена, на которое указывает Res.

if (q != NULL)//если звено существует

{

//если звено, после которого нужно удалять,

//не является последним, то:

(\*Res).sled = (\*(\*Res).sled).sled; //изменяем указатель на следующее звено на указатель на звено через одно

delete q;//освобождаем память, занятую удаляемым звеном

}

else //выводим соответствующее сообщение

cout << "Звено с заданным элементом - последнее!\n";

}

void Spisok::YDALE1()

//удаление звена, на которое указывает ссылка Res.

{

node\* q, \* q1, \* q2;

q = (\*Res).sled; //указатель на звено, которое нужно удалить

if (q != NULL)//если звено существует

{

(\*Res).elem = (\*q).elem; //значение элемента текущего звена заменяем на значение элемента удаляемого звена

(\*Res).sled = (\*q).sled;//меняем указатель на следующее звено на указатель на звено, следующее за удаляемым

delete q;//освобождаем память, занятую удаляемым звеном

}

else

{

q1 = phead; q2 = (\*q1).sled;//указатели на предыдущее и текущее звенья

while (q2 != Res)//ищем звено, следующее за текущим

{

q1 = q2; q2 = (\*q2).sled;

}

(\*q1).sled = NULL; q2 = NULL; delete Res;//меняем указатель на следующее звено у предыдущего звена на NULL и освобождаем память удаляемого звена

}

}

void Spisok::OCHISTKA()

//удаление однонаправленного списка из памяти

// phead - указатель на заглавное звено списка

{

node\* q, \* q1;// рабочие указатели

q = phead;// указатель q указывает на заглавное звено

q1 = (\*q).sled; // указатель q1 "опережает" указатель q

while (q1 != NULL)

{

q = q1; // передвигаем указатель q на следующее звено

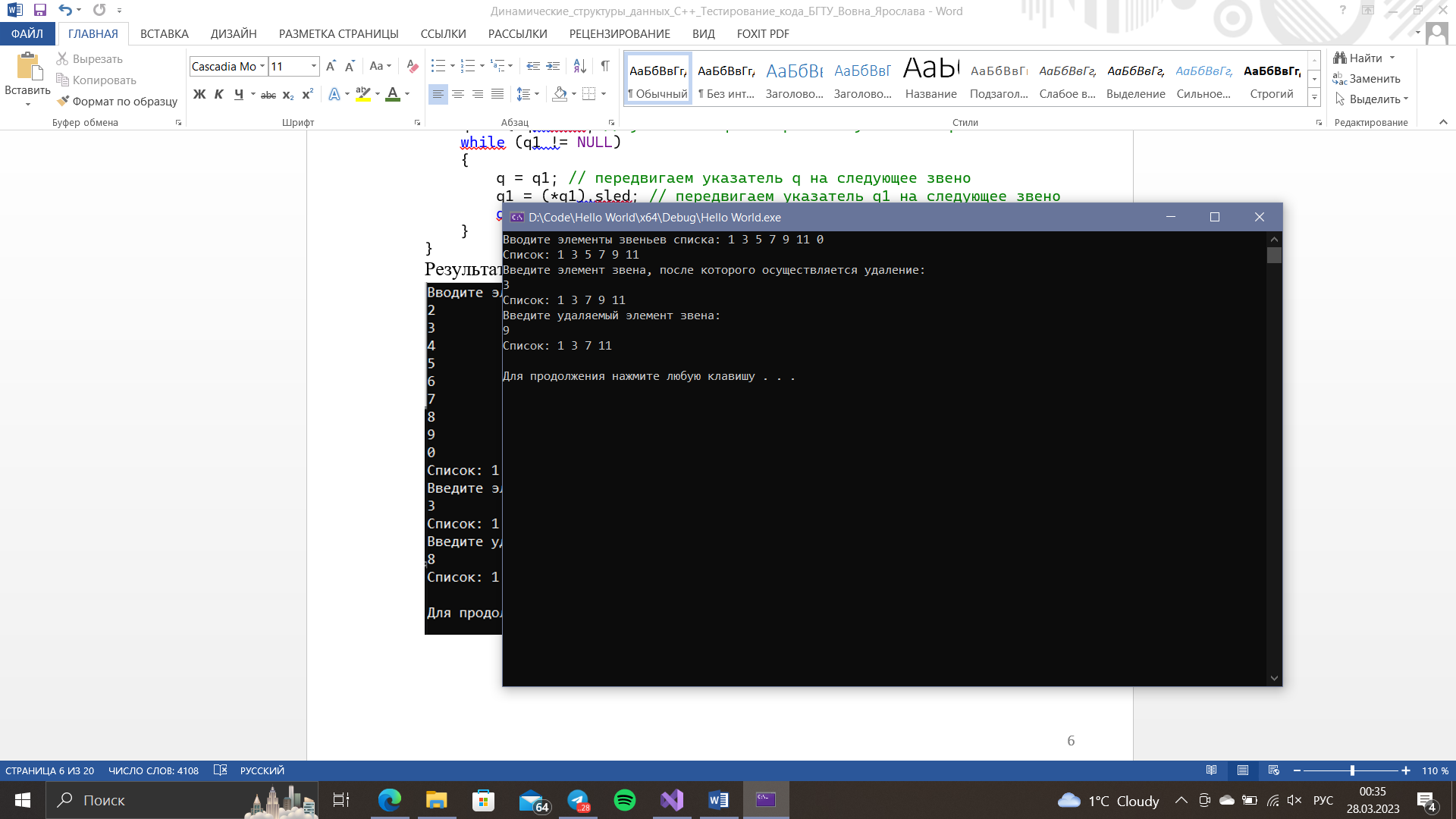
q1 = (\*q1).sled; // передвигаем указатель q1 на следующее звено

delete q; // удаляем предыдущее звено из памяти

}

}

Результат программы:



**Программа 2**

Код программы, который осуществляет основные операции над ортогональными списками:

#include <iostream>

using namespace std;

struct nodeVis // объявление структуры nodeVis

{

int elem; //информационное поле звена висюльки

nodeVis\* vniz;//указатель на звено висюльки

};

struct nodeGir // объявление структуры nodeGir

{

int elem;//информационное поле звена гирлянды

nodeVis\* vniz;//указатель на звено висюльки

nodeGir\* sled;//указатель на звено гирлянды

};

class GirVis { // объявление класса GirVis

private:

nodeGir\* phead;//начало(голова) гирлянды

nodeVis\* pheadVis;//начало(голова) висюльки

void VisVyvod(); // метод для вывода информации о висюльке

public:

GirVis() { phead = new (nodeGir); } // конструктор класса GirVis

~GirVis() { delete phead; } //деструктор класса GirVis

nodeVis\* VisPostr(); //метод построения новой висюльки

nodeVis\* VisPoisk(int); //метод для поиска висюльки в гирлянде

void SetpheadVis(nodeVis\* r) { pheadVis = r; } //определение головы висюльки

void VisVstav(nodeVis\*, int); //метод для вставки новой висюльки в определенное место гирлянды

void Vis1Vstav(nodeVis\*, int);//метод для вставки первой висюльки в определенное место гирлянды

void VisUdale(nodeVis\*); //метод для удаления висюльки из гирлянды

void Vis1Udale(nodeVis\*); //метод для удаления первой висюльки из гирлянды

void GirPostr(); //метод для добавления новой гирлянды

void GirVyvod(); //метод для вывода информации о гирлянде

nodeGir\* GirPoisk(int); //метод для поиска гирлянды по ее номеру

void OCHISTKA(); //метод для очистки гирлянды

void OCHISTKA1(); //метод для очистки первой висюльки гирлянды

};

void main()

{

setlocale(LC\_ALL, "Rus"); // подключение русского языка в консоли

GirVis A; //создание объекта класса GirVis

int el, elGir, elVis;

nodeGir\* Res; //рабочий указатель

nodeVis\* ResVis; //указатель на звено висюльки

A.GirPostr(); //добавление новой гирлянды в объект класса A

A.GirVyvod();//вывод информации о гирлянде из объекта класса A

cout << "\nВведите элемент звена гирлянды, ";

cout << "чьи висюльки будем изменять:\n";

cin >> elGir;//ввод номера звена гирлянды

cout << "\nВведите элемент звена висюльки, после которого ";

cout << "осуществляется вставка:\n";

cin >> elVis;//ввод номера звена висюльки

cout << "\nВведите вставляемый элемент:\n";

cin >> el;//ввод значения новой висюльки

//воиск элемента elGir в гирлянде

Res = A.GirPoisk(elGir);

if (Res != NULL)

{

//поиск элемента elVis в висюльке

A.SetpheadVis((\*Res).vniz); //установка головы висюльки на элемент под гирляндой Res

ResVis = A.VisPoisk(elVis); //поиск элемента elVis в висюльке

if (ResVis != NULL)

A.VisVstav(ResVis, el); //вставка новой висюльки ResVis после висюльки Res в гирлянде

else cout << "Элемента в висюльке нет!\n";

}

else cout << "Элемента в гирлянде нет\n";

A.GirVyvod(); //вывод информации о гирлянде из объекта класса A

cout << "\nВведите элемент гирлянды, чью висюльку будем изменять:\n"; cin >> elGir; //ввод номера звена гирлянды

cout << "Введите элемент висюльки, перед которым ";

cout << "осуществляется вставка:\n";

cin >> elVis; //ввод номера звена висюльки

cout << "Введите вставляемый элемент:\n";

cin >> el; //ввод значения новой висюльки

//поиск элемента elGir в гирлянде

Res = A.GirPoisk(elGir); //поиск элемента elGir в гирлянде и сохранение результата в Res

if (Res != NULL)

{

//поиск элемента elVis в висюльке

A.SetpheadVis((\*Res).vniz); //установка головы висюльки для элемента Res

ResVis = A.VisPoisk(elVis); //поиск элемента elVis в висюльке и сохранение результата в ResVis

if (ResVis != NULL)

A.Vis1Vstav(ResVis, el);//вставка элемента el после элемента ResVis в висюльке

else cout << "Элемента в висюльке нет!\n";

}

else cout << "Элемента в гирлянде нет!\n";

A.GirVyvod(); //вывод гирлянды

cout << "\nВведите элемент гирлянды, чью висюльку будем изменять:\n";

cin >> elGir;

cout << "Введите элемент висюльки, после которого нужно удалить:\n";

cin >> elVis;

//поиск элемента elGir в гирлянде

Res = A.GirPoisk(elGir);

if (Res != NULL)

{

//поиск элемента elVis в висюльке

A.SetpheadVis((\*Res).vniz); //установка головы висюльки для элемента Res

ResVis = A.VisPoisk(elVis); //поиск элемента elVis в висюльке и сохранение результата в ResVis

if ((ResVis != NULL) && ((\*ResVis).vniz != NULL))

A.VisUdale(ResVis); //удаление элемента ResVis из висюльки

else cout << "Элемента в висюльке нет!\n";

}

else cout << "Элемента в гирлянде нет!\n";

A.GirVyvod(); //вывод гирлянды

cout << "\nВведите элемент гирлянды, чью висюльку будем изменять:\n";

cin >> elGir;

cout << "Введите элемент висюльки, который удаляется:\n";

cin >> elVis;

//поиск элемента elGir в гирлянде

Res = A.GirPoisk(elGir);

if (Res != NULL)

{

//поиск элемента elVis в висюльке

A.SetpheadVis((\*Res).vniz); // определение головы висюльки

ResVis = A.VisPoisk(elVis);// вызов метода для поиска элемента elVis в висюльке и сохранение результата в ResVis

if ((ResVis != NULL) && ((\*ResVis).vniz != NULL))// если элемент найден и он не последний в висюльке

A.Vis1Udale(ResVis); // вызов метода для удаления элемента ResVis

else cout << "Элемента в висюльке нет или он последний!\n";

}

else cout << "Элемента в гирлянде нет!\n";

A.GirVyvod();// вызов метода для вывода гирлянды на экран

A.OCHISTKA();// вызов метода для очистки памяти, занятой объектами класса

cout << "\n";

system("PAUSE");

}

void GirVis::OCHISTKA()

{

nodeGir\* q, \* q1;//рабочие указатели

q = phead;// указатель q указывает на первый элемент гирлянды

q1 = (\*q).sled; //указатель q1 "опережает" указатель q

while (q1 != NULL) // пока q1 указывает на какой-то элемент

{

q = q1; q1 = (\*q1).sled; // смещение указателей q и q1 на следующий элемент гирлянды

pheadVis = (\*q).vniz; // указатель на голову висюльки устанавливается на первый элемент висюльки у текущего элемента гирлянды

OCHISTKA1(); // вызов функции очистки висюльки

delete q; // освобождение памяти выделенной для текущего элемента гирлянды

}

}

void GirVis::OCHISTKA1()

{

nodeVis\* q, \* q1;

q = pheadVis; // указатель q указывает на первый элемент висюльки

q1 = (\*q).vniz; // указатель q1 указывает на второй элемент висюльки

while (q1 != NULL) // пока q1 указывает на какой-то элемент

{

q = q1; q1 = (\*q1).vniz; // смещение указателей q и q1 на следующий элемент висюльки

delete q; // освобождение памяти выделенной для текущего элемента висюльки

}

}

void GirVis::GirPostr()

//построение однонаправленного списка с заглавным звеном,

//заданного указателем phead (построение гирлянды)

{

nodeGir\* t;

int el;

t = phead; (\*t).sled = NULL; //указатель t присваиваем phead и обнуляем его следующий элемент

cout << "Вводите элемент гирлянды: \n";

cin >> el;

while (el != 0)

{

(\*t).sled = new (nodeGir); //создание нового элемента гирлянды с помощью оператора new

t = (\*t).sled; //переход к следующему элементу гирлянды

(\*t).elem = el; (\*t).sled = NULL; //присваивание значений полям созданного элемента гирлянды

(\*t).vniz = VisPostr(); //создание висюльки для текущего элемента гирлянды с помощью функции VisPostr()

cout << " Вводите элемент гирлянды: \n";

cin >> el;

}

}

nodeVis\* GirVis::VisPostr()

//построение однонаправленного списка с заглавным звеном

//(построение висюльки), где pheadVis - указатель на висюльку

{

nodeVis\* t;

int el;

//создадим заглавное звено списка

pheadVis = new (nodeVis);

t = pheadVis; (\*t).vniz = NULL; //создание заглавного звена списка висюльки

cout << "Вводите элементы звеньев висюльки: \n";

cin >> el;

while (el != 0)

{

(\*t).vniz = new (nodeVis); //создание нового элемента висюльки.

t = (\*t).vniz; (\*t).elem = el; (\*t).vniz = NULL; //присваивание значений полям созданного элемента висюльки.

cin >> el;

}

return pheadVis;

}

void GirVis::GirVyvod()

//Вывод содержимого однонаправленного списка, заданного

//указателем phead (вывод содержимого гирлянды)

{

nodeGir\* t;

t = phead; t = (\*t).sled; // установка указателя на первый элемент списка

cout << "Гирлянда: ";

while (t != NULL)

{

cout << (\*t).elem << " "; //вывод текущего элемента гирлянды

pheadVis = (\*t).vniz; //переход к первому элементу висюльки

VisVyvod(); //вывод висюльки

t = (\*t).sled; //переход к следующему элементу гирлянды

}

}

nodeGir\* GirVis::GirPoisk(int el)

//поиск элемента el в списке, заданном указателем phead.

//в случае успешного поиска возвращается адрес звена списка,

//содержащего элемент el. В противном случае - NULL

{

nodeGir\* t, \* r;

r = NULL; t = phead; //инициализация рабочих указателей t и r

t = (\*t).sled; //переход к первому элементу списка

while (t != NULL && r == NULL) //поиск элемента el в списке

if ((\*t).elem == el) r = t;

else t = (\*t).sled;

return r;

}

void GirVis::VisVyvod()

//вывод содержимого однонаправленного списка с заглавным звеном,

//заданного указателем pheadVis (вывод содержимого висюльки)

{

nodeVis\* t;

t = pheadVis; //инициализация рабочего указателя t

t = (\*t).vniz; //переход к первому элементу висюльки

cout << "(";

while (t != NULL)

{

cout << (\*t).elem << " "; //вывод текущего элемента висюльки

t = (\*t).vniz; //переход к следующему элементу висюльки

}

cout << ")";

}

nodeVis\* GirVis::VisPoisk(int el)

//поиск элемента el в списке, заданном указателем pheadVis.

//В случае успешного поиска возвращается адрес звена списка,

//содержащего элемент el. В противном случае - NULL

{

nodeVis\* t, \* r;

r = NULL; t = pheadVis; //инициализируем указатель r значением NULL и указатель t значением pheadVis

t = (\*t).vniz; //затем перемещаем указатель t на следующее звено в списке

while (t != NULL && r == NULL) //запускаем цикл по всем звеньям списка, пока не дойдем до конца списка или не найдем элемент el

if ((\*t).elem == el) r = t; //если элемент найден, присваиваем его адрес r

else t = (\*t).vniz; // в противном случае перемещаем указатель t на следующее звено.

return r; //возвращаем адрес звена списка, содержащего элемент el или NULL, если элемент не найден

}

void GirVis::VisVstav(nodeVis\* r, int el)

//включение звена с информационным полем el

//после звена, на которое указывает r

//(включение звена в висюльку)

{

nodeVis\* q;

q = new (nodeVis); //выделяем память для нового звена списка

(\*q).elem = el; (\*q).vniz = (\*r).vniz; (\*r).vniz = q;

//инициализируем информационное поле звена нового звена списка

//значением el, а указатель на следующее звено - указателем на следующее звено звена,

//на которое указывает r. Затем присваиваем указателю на следующее звено звена,

//на которое указывает r, адрес нового звена списка

}

void GirVis::Vis1Vstav(nodeVis\* r, int el)

//включение звена с информационным полем el

//перед звеном, на которое указывает r

//(включение звена в висюльку)

{

nodeVis\* q; //объявляем указатель q на структуру nodeVis

q = new (nodeVis); //выделяем память для нового звена списка

(\*q).elem = (\*r).elem; // копируем значение элемента из узла r в новый узел

(\*q).vniz = (\*r).vniz;// связываем новый узел с последующим узлом после r

(\*r).elem = el; // заменяем значение элемента в узле r на новое значение el

(\*r).vniz = q; // связываем узел r с новым узлом q, теперь q следует за r в списке

}

void GirVis::VisUdale(nodeVis\* r)

//удаление звена, расположенного после звена,

//на которое указывает ссылка r

//(удаление звена висюльки)

{

nodeVis\* q;

q = (\*r).vniz;

// если звено, расположенное после звена, на которое указывает ссылка r, не является последним

if ((\*r).vniz != NULL)

{

(\*r).vniz = (\*(\*r).vniz).vniz; // переназначаем указатель r на звено, расположенное после удаляемого звена

delete q; // освобождаем память, занятую удаляемым звеном

}

else cout << "Звено с заданным элементом - последнее!\n";

}

void GirVis::Vis1Udale(nodeVis\* r)

//удаление звена, на которое указывает ссылка r

//(удаление звена висюльки)

{

nodeVis\* g;

if ((\*r).vniz != NULL)

{

g = (\*r).vniz; // сохраняем указатель на удаляемое звено

(\*r).elem = (\*(\*r).vniz).elem; // перемещаем элемент из следующего звена в текущее звено

(\*r).vniz = (\*(\*r).vniz).vniz; // переназначаем указатель r на звено, расположенное после удаляемого звена

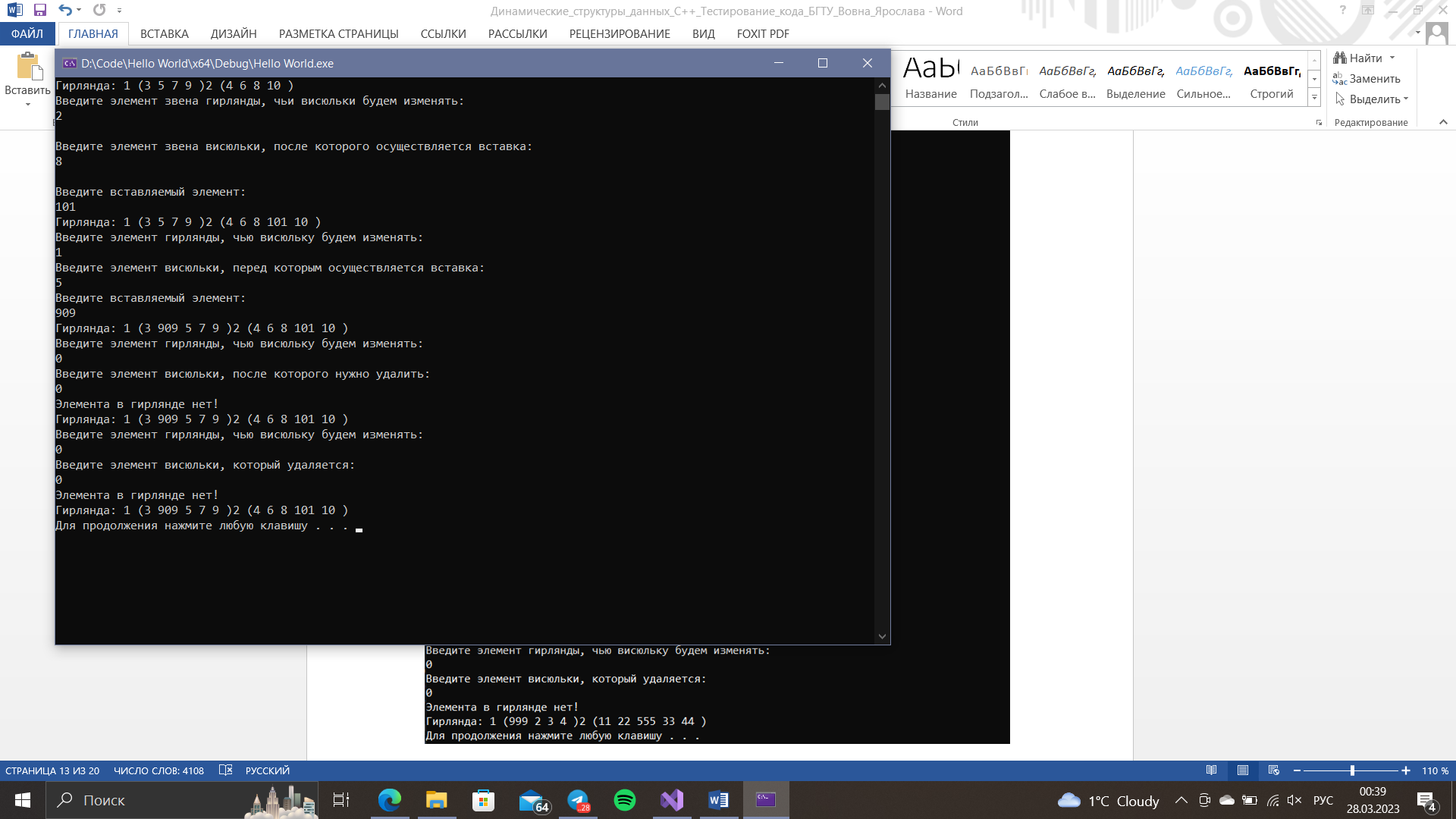
delete g; // освобождаем память, занятую удаляемым звеном

}

else cout << "Не умею удалять последнее звено!\n";

}

Результат программы:



**Программа 3**

Код программы, который осуществляет: включение звена в кольцо с удаленным заглавным звеном, удаление звена с заданным значением информационного поля из кольца с удаленным заглавным звеном:

#include <iostream>

using namespace std;

struct node // объявление структуры node

{

int elem; // поле элемента списка

node\* sled; // указатель на следующее звено списка

};

class Spisok { // объявление класса Spisok для работы со списком

private:

node\* phead, \* Res; // указатели на голову списка и текущее звено списка

public:

Spisok() { phead = new(node); Res = NULL; } // конструктор класса Spisok

~Spisok() { delete phead; } // деструктор класса Spisok

void POSTROENIE(); // функция для построения списка

void VYVOD(); // функция для вывода списка

node\* POISK(int); // функция для поиска звена по элементу

void InsAfter(int); //функция для вставки звена после текущего звена

void InsBefore(int); //функция для вставки звена перед текущим звеном

void Delete(); // функция для удаления звена.

void DelAfter(); //функция для удаления звена, следующего за текущим звеном

void OCHISTKA(); // функция для очистки памяти.

};

void main()

{

setlocale(LC\_ALL, "Rus");

Spisok A; // создание объекта класса Spisok

int el, el1; // переменные для элементов списка

node\* Res\_Zn; // указатель на текущее звено списка

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

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

cout << "\nВведите элемент звена, после которого ";

cout << "осуществляется вставка: ";

cin >> el;

cout << "\nВведите элемент вставляемого звена: ";

cin >> el1;

if (A.POISK(el) != NULL) //проверяем, найдено ли звено с заданным элементом в кольце

{

A.InsAfter(el1); A.VYVOD(); //вызываем функцию InsAfter, чтобы вставить новое звено после звена с заданным элементом, и функцию VYVOD, чтобы вывести на экран измененное кольцо

}

else cout << "Звена с заданным элементом в кольце нет!";

cout << "\nВведите элемент звена, перед которым ";

cout << "осуществляется вставка: ";

cin >> el;

cout << "Введите элемент вставляемого звена: ";

cin >> el1;

if (A.POISK(el) != NULL)

{

A.InsBefore(el1); A.VYVOD(); //вызываем функцию InsBefore, чтобы вставить новое звено перед звеном с заданным элементом, и функцию VYVOD, чтобы вывести на экран измененное кольцо

}

else cout << "Звена с заданным элементом в кольце нет!";

cout << "\nВведите элемент удаляемого звена: ";

cin >> el;

if (A.POISK(el) != NULL)

{

A.Delete(); A.VYVOD(); //вызываем функцию Delete, чтобы удалить звено с заданным элементом, и функцию VYVOD, чтобы вывести на экран измененное кольцо

}

else cout << "Звена с заданным элементом в кольце нет!";

cout << "\nВведите элемент звена, ";

cout << "после которого нужно удалять: ";

cin >> el;

if (A.POISK(el) != NULL)

{

A.DelAfter(); A.VYVOD(); //вызываем функцию DelAfter, чтобы удалить звено со следующим за заданным элементом, и функцию VYVOD, чтобы вывести на экран измененное кольцо

}

else cout << " Звена с заданным элементом в кольце нет!";

A.OCHISTKA(); // вызов функции для очистки

cout << "\n";

system("PAUSE");

}

void Spisok::POSTROENIE()

//построение кольцевого списка с удаленным заглавным звеном

//phead - указатель на заглавное звено

{

node\* t; // объявление указателя на звено

int el; // объявление переменной типа int

t = phead; // присвоение указателю t значения phead

(\*t).sled = NULL; // обнуление указателя следующего звена

cout << "Вводите элементы кольца: ";

cin >> el;

while (el != 0)

{

(\*t).sled = new (node); // выделение памяти под новое звено

t = (\*t).sled; // присвоение указателю t значения указателя на следующее звено

(\*t).elem = el; // присвоение элементу звена значения переменной el

cin >> el; // ввод значения в переменную el

}

(\*t).sled = (\*phead).sled; // присвоение указателю следующего звена значения указателя на следующее звено после заглавного звена

}

void Spisok::VYVOD()

//вывод содержимого кольцевого списка с удаленным заглавным звеном

//phead - указатель на заглавное звено

{

node\* t; // объявление указателя на звено

t = (\*phead).sled; // присвоение указателю t значения указателя на следующее звено после заглавного звена

cout << "Кольцо: ";

if (t != NULL) // если указатель на следующее звено не равен NULL

{

cout << (\*t).elem << " "; // вывод значения элемента звена

t = (\*t).sled; // присвоение указателю t значения указателя на следующее звено

while (t != (\*phead).sled) // цикл продолжается пока указатель на следующее звено не равен указателю на следующее звено после заглавного звена

{

cout << (\*t).elem << " "; // вывод значения элемента звена.

t = (\*t).sled; // присвоение указателю t значения указателя на следующее звено

}

}

else cout << "пусто!\n";

}

node\* Spisok::POISK(int el)

//поиск элемента el в кольцевом списке phead

//если элемент найден, то Res содержит указатель на звено,

//содержащее элемент el. В противном случае - NULL

{

node\* t;

Res = NULL; t = (\*phead).sled;

while ((\*t).sled != (\*phead).sled && Res == NULL)

if ((\*t).elem == el) Res = t;

else t = (\*t).sled;

if (Res == NULL && (\*t).elem == el)

Res = t;

return Res;

}

void Spisok::InsAfter(int el)

//включение звена с информационным полем el в кольцо

//после звена, на которое указывает ссылка Res

{

node\* q;

q = new (node); //создается новое звено

(\*q).elem = el; //информационному полю созданного звена присваивается значение el

(\*q).sled = (\*Res).sled; //указатель на следующее звено нового звена присваивается указателю на следующее звено звена, на которое указывает ссылка Res

(\*Res).sled = q; //указатель на следующее звено звена, на которое указывает ссылка Res, присваивается указателю на созданное звено

}

void Spisok::InsBefore(int el)

//включение звена с информационным полем el в кольцо

//перед звеном, на которое указывает ссылка Res

{

node\* q;

q = new (node);

(\*q).elem = (\*Res).elem; (\*q).sled = (\*Res).sled; //для включения звена перед звеном, на которое указывает ссылка Res, создается новое звено, в которое копируется информационное поле и указатель на следующее звено звена, на которое указывает ссылка Res

(\*Res).elem = el; (\*Res).sled = q; //информационному полю звена, на которое указывает ссылка Res, присваивается значение el, а указатель на следующее звено этого звена присваивается указателю на созданное звено

}

void Spisok::Delete()

//удаление звена, на которое указывает ссылка Res,

//из кольцевого списка с удаленным заглавным звеном,

//заданного указателем phead

{

node\* z, \* q;

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

// если Res указывает не на последний элемент в списке

{

q = (\*Res).sled; // сохраняем указатель на следующий элемент

(\*Res).elem = (\*((\*Res).sled)).elem; // копируем информационное поле следующего элемента в Res

(\*Res).sled = (\*((\*Res).sled)).sled; // ссылка Res теперь указывает на следующий элемент в списке

delete q; // удаляем следующий элемент

}

else if ((\*Res).sled == Res)

// если Res указывает на последний элемент в списке и в списке есть только один элемент

{

//в кольце единственное звено

q = (\*phead).sled; (\*phead).sled = NULL;

// удаляем единственный элемент из списка

delete q; cout << "Кольцо пусто!";

}

else

// если Res указывает на последний элемент в списке и в списке есть более одного элемента

{

//удаляется "последнее" звено кольца

z = phead; q = (\*phead).sled;

while (q != Res)

{

z = q; q = (\*q).sled;

// идём по списку до тех пор, пока не найдём элемент, который следует за Res

}

(\*z).sled = (\*((\*z).sled)).sled; // ссылка на элемент следующий за Res теперь указывает на элемент, следующий за удаляемым элементом

delete q; // удаляем элемент, следующий за Res

}

}

void Spisok::DelAfter()

//удаление звена, расположенного после звена,

//на которое указывает ссылка Res,

//из кольцевого списка с удаленным заглавным звеном,

//заданного указателем phead

{

node\* q;

if ((\*Res).sled != (\*phead).sled) //если ссылка Res не указывает на последнее звено списка

{

q = (\*Res).sled; //то запоминаем адрес удаляемого звена, переставляем указатель на следующее звено и освобождаем память, выделенную для удаляемого звена

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

delete q;

}

else if ((\*Res).sled == Res) //если удаляемое звено - единственное звено в списке

{

q = (\*phead).sled; //то запоминаем адрес удаляемого звена, удаляем его и выводим сообщение о том, что список пустой

(\*phead).sled = NULL;

delete q;

cout << "Кольцо пусто!";

}

else //если удаляемое звено - первое в списке и не единственное

{

q = (\*phead).sled; //то запоминаем адрес удаляемого звена, переставляем указатель на следующее звено,перемещаем указатель на начало списка на следующее звено и освобождаем память,выделенную для удаляемого звена

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

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

delete q;

}

}

void Spisok::OCHISTKA()

{

node\* q, \* q1;// рабочие указатели

q = phead;

q1 = (\*q).sled; // указатель q1 "опережает" указатель q

do {

q = q1; //запоминаем адрес текущего звена, переставляем указатель на следующее звено и освобождаем память, выделенную для текущего звена

q1 = (\*q1).sled;

delete q;

} while (q1 != (\*phead).sled); //пока указатель на следующее звено не указывает на заглавное звено списка

}

Результат программы:

