БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

**Динамические структуры данных. Двунаправленный список. Кольцевой список. Тестирование кода.**

**По:** Основы алгоритмизации и программирования

**Выполнил:** Корнелюк

Валентин Владимирович

1 курс 4 группа ПОИТ

**Преподаватель:**

Белодед Николай Иванович

**г. Минск**

2023 г.

**Оглавление**

[Построение двунаправленного списка. Помещение звена в двунаправленный список. Удаление звена из двунаправленного списка. Очистка двунаправленного списка. 3](#_Toc132146290)

[Построение двунаправленного кольца с удаленным заглавным звеном, вывод содержимого кольцевого списка, вставка звена, удаление звена. 9](#_Toc132146291)

# Построение двунаправленного списка. Помещение звена в двунаправленный список. Удаление звена из двунаправленного списка. Очистка двунаправленного списка.

#include <iostream> // подключение библиотеки для стандартных операций ввода/вывода

using namespace std; // директива using, которая позволяет использовать пространство имен std без указания префикса std.

struct node // определение структуры элемента списка

{

int elem; // информационное поле

node\* sled; // указатель на следующий элемент

node\* pred; // указатель на предыдущий элемент

};

class Spisok // класс для работы со списком

{

private:

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

public:

Spisok() { nsp = ksp = NULL; } // конструктор класса, инициализирующий начало и конец списка

void Postroenie(); // функция для построения списка

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

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

void Ochistka(); // функция для очистки списка

void InsAfter(int, node\*);// функция для вставки нового элемента после заданного

void InsBefore(int, node\*); // функция для вставки нового элемента перед заданным

void Delete(node\*); // функция для удаления заданного элемента

void DelAfter(node\*); // функция для удаления элемента после заданного

node\* PoiskForward(int);// функция для поиска элемента списка в прямом направлении

node\* PoiskBack(int); // функция для поиска элемента списка в обратном направлении

};

void main() // главная функция программы

{

setlocale(LC\_ALL, "Rus"); // установка локали на русский язык

Spisok A; // создание объекта класса Spisok

node\* Res; // указатель на элемент списка

int el, el1; // переменные для ввода элементов списка

A.Postroenie(); // построение списка

A.VyvodForward(); A.VyvodBack(); // вывод списка в прямом и обратном направлениях

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

cout << "осуществляется вставка: ";

cin >> el; //ввод элемента, после которого будет производится вставка

cout << "Введите элемент вставляемого звена: ";

cin >> el1; //ввод элемента, который будем вставлять

Res = A.PoiskForward(el); // поиск элемента после которого будет производится вставка

if (Res != NULL) // если такой элемент найден

{

A.InsAfter(el1, Res); // вставка нового элемента после заданного элемента

A.VyvodForward(); A.VyvodBack(); // вывод списка в прямом и обратном направлениях

}

else cout << "Звена с заданным элементом в списке нет!\n"; // если элемент не найден

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

cout << "осуществляется вставка: ";

cin >> el; //ввод элемента, перед которым будем вставлять элемент

cout << "Введите элемент вставляемого звена: ";

cin >> el1; //ввод элемента, который будем вставлять

Res = A.PoiskBack(el); // поиск элемента, перед которым будем вставлять новый элемент

if (Res != NULL) // если такой элемент найден

{

A.InsBefore(el1, Res); //вставка нового элемента перед заданным элементом

A.VyvodForward(); A.VyvodBack(); // вывод списка в прямом и обратном направлениях

}

else cout << "Звена с заданным элементом в списке нет!\n"; // если такой элемент не найден

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

cout << "осуществляется удаление: ";

cin >> el; // ввод элемента, после которого будет производиться удаление

Res = A.PoiskForward(el); // поиск элемента, после которого будет производиться удаление

if (Res != NULL) // если такой элемент найден

{

A.DelAfter(Res); // удаление элемента после заданного элемента

A.VyvodForward(); A.VyvodBack(); // вывод списка в прямом и обратном направлениях

}

else cout << "Звена с заданным элементом в списке нет!\n"; //если такой элемент не найден

cout << "Введите элемент звена, которое ";

cout << "надо удалить: ";

cin >> el; // ввод элемента который собираемся удалять

Res = A.PoiskForward(el); // поиск элемента, который собираемся удалять

if (Res != NULL) //если такой элемент найден

{

A.Delete(Res); //удаление заданного элемента

A.VyvodForward(); A.VyvodBack(); // вывод списка в прямом и обратном направлениях

}

else cout << "Звена с заданным элементом в списке нет!\n"; //если такой элемент не найден

A.Ochistka(); //очиста списка

cout << "\n"; // переход на новую строку

system("PAUSE");//ожидание нажатия любой клавиши

}void Spisok::Postroenie()

//Построение двунаправленного списка с заглавным звеном:

// nsp - указатель на начало списка,

// ksp - указатель на конец списка.

{

node\* rsp; // указатель на текущее звено списка

int el; // переменная для ввода элементов

nsp = new(node); // создание заглавного звена

rsp = nsp; //текущее звено указывает на начало списка

(\*nsp).pred = NULL; (\*nsp).sled = NULL; //настройка указателей

cout << "Вводите последовательность:\n";

cin >> el; //ввод элемента последовательности

while (el != 0) //пока не введен ноль

{

(\*rsp).sled = new(node); // создание нового звена

(\*((\*rsp).sled)).pred = rsp; // установка указателя на предыдущее звено

rsp = (\*rsp).sled; // перемещение указателя на следующее звено

(\*rsp).sled = NULL; (\*rsp).elem = el; // установка значения элемента звена

cin >> el; //ввод элемента последовательности

}

ksp = rsp; // установка указателя на конец списка

}

void Spisok::VyvodForward()

//Вывод содержимого двунаправленного списка от его начала.

// nsp - указатель на начало списка, ksp - указатель на конец списка.

{

node\* rsp; // Объявление указателя на звено списка

rsp = (\*nsp).sled; // Инициализация указателя на первый элемент списка (после заголовочного звена)

cout << "Двунаправленный список содержит: ";

while (rsp != NULL)// Пока указатель не указывает на нулевой указатель

{

cout << (\*rsp).elem << " ";// Вывод значения элемента на экран

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

}

cout << endl; // Вывод символа перевода строки

}

void Spisok::VyvodBack()

//Вывод содержимого двунаправленного списка от его конца.

// nsp - указатель на начало списка, ksp - указатель на конец списка.

{

node\* rsp; // Объявление указателя на звено списка

rsp = ksp; // Инициализация указателя на последний элемент списка

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

while ((\*rsp).pred != NULL) // Пока указатель не указывает на нулевой указатель

{

cout << (\*rsp).elem << " "; ;// Вывод значения элемента на экран

rsp = (\*rsp).pred; // Перемещение указателя на следующий элемент

}

cout << endl; // Вывод символа перевода строки

}

node\* Spisok::PoiskForward(int el)

//Функция возвращает указатель на найденный элемент el

//двунаправленного списка, заданного указателями nsp

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

{

node\* q; //рабочий указатель

node\* Res; //указатель для найденног элемента

Res = NULL; q = (\*nsp).sled; //Инициализация указателя на NULL и начало поиска с первого элемента списка

while (q != NULL && Res == NULL) //Пока не дошли до конца списка и не нашли элемент el, идем по списку

if ((\*q).elem == el) Res = q; // есил элемент найден, записываем результат в Res

else q = (\*q).sled; // переходим к следующему элементу

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

}

node\* Spisok::PoiskBack(int el)

//Функция возвращает указатель на найденный элемент el

//двунаправленного списка, заданного указателями nsp

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

{

node\* q; //рабочий указатель

node\* Res; //указатель для найденног элемента

Res = NULL; q = ksp; //Инициализация указателя на NULL и начало поиска с первого элемента списка

while (q != NULL && Res == NULL) //Пока не дошли до конца списка и не нашли элемент el, идем по списку

if ((\*q).elem == el) Res = q; // есил элемент найден, записываем результат в Res

else q = (\*q).pred; // переходим к следующему элементу

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

}

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; //рабочий указатель

q = new(node); // создание нового звена

(\*q).elem = el; // инициализация информационного поля звена новым значением

if ((\*Res).sled != NULL) // если Res не является последним звеном списка

{

(\*q).sled = (\*Res).sled; // установка указателя следующего звена нового звена на следующий за Res звеньев

(\*q).pred = (\*(\*Res).sled).pred; // установка указателя предыдущего звена нового звена на предыдущий за Res звеньев

(\*(\*Res).sled).pred = q; // установка указателя предыдущего звена следующего за Res звена на q

(\*Res).sled = q; // установка указателя следующего звена Res на q

}

else // если Res является последним звеном списка

{

(\*q).sled = NULL; // установка указателя следующего звена нового звена на NULL

(\*q).pred = Res; // установка указателя предыдущего звена нового звена на Res

ksp = q; // изменение указателя на последний элемент списка на q

(\*Res).sled = q; // установка указателя следующего звена Res на 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; //рабочий указатель

q = new(node); // создание нового звена

(\*q).elem = el; // инициализация информационного поля звена новым значением

(\*q).sled = (\*(\*Res).pred).sled; // новый элемент "соединяется" со следующим элементом после Res

(\*q).pred = (\*Res).pred; // предыдущий элемент перед Res становится предыдущим для нового элемента

(\*(\*Res).pred).sled = q; // следующим элементом после предыдущего элемента становится новый элемент

(\*Res).pred = q; // элемент Res теперь предыдущий для нового элемента

}

void Spisok::Delete(node\* Res)

//Удаление звена из двунаправленного списка.

// nsp - указатель на начало списка,

// ksp - указатель на конец списка,

// Res - указатель на удаляемое звено.

{

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

{

(\*(\*Res).sled).pred = (\*Res).pred; // предыдущим для следующего элемента после Res становится предыдущий элемент Res

(\*(\*Res).pred).sled = (\*Res).sled; // следующим за предыдущим элементом становится следующий элемент Res

delete Res; // удаление элемента

}

else // если элемент последний

{

(\*(\*Res).pred).sled = NULL; // предыдущий элемент становится последним

ksp = (\*ksp).pred; // конечный элемент списка становится предпоследним

delete Res; // удаление элемента

}

}

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; //рабочий указатель

if ((\*Res).sled == NULL) // Проверяем, является ли удаляемое звено последним звеном в списке.

{

cout << "Указано последнее звено!\n";

}

else

{

if ((\*(\*Res).sled).sled != NULL) // Если удаляемое звено не является последним, то перенаправляем указатели вокруг него.

{

q = (\*Res).sled; // сохраняем указатель на удаляемое звено.

(\*(\*(\*Res).sled).sled).pred = Res; // перенаправляем указатель на предыдущее звено.

(\*Res).sled = (\*(\*Res).sled).sled; // перенаправляем указатель на следующее звено.

delete q; // удаляем звено.

}

else // Если удаляемое звено является последним, то просто удаляем его и перенаправляем указатель на последнее звено в списке на предыдущее звено.

{

q = (\*Res).sled; // сохраняем указатель на удаляемое звено.

(\*Res).sled = NULL; // перенаправляем указатель на следующее звено на NULL.

ksp = (\*ksp).pred; // перенаправляем указатель на последнее звено на предыдущее звено.

delete q; // удаляем звено.

}

}

}

void Spisok::Ochistka()

//Удаление двунаправленного списка из памяти.

// nsp - указатель на заглавное звено списка,

// ksp - указатель на последнее звено списка.

{

node\* q, \* q1; // рабочие указатели

q = nsp; // устанавливаем указатель q на заглавное звено списка.

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

while (q1 != NULL) // пока не достигнем конца списка.

{

q = q1; // перемещаем указатель q на следующее звено.

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

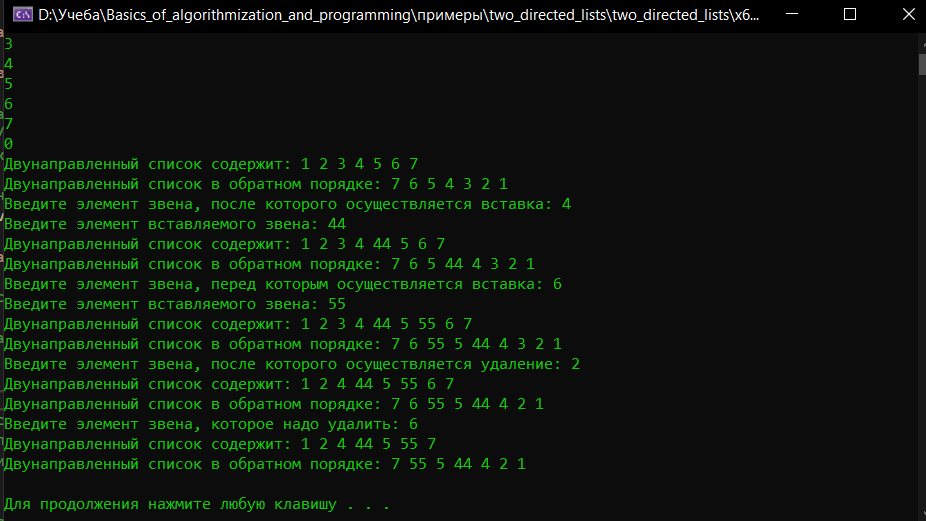
delete q; // удаляем звено.

}

delete nsp; // удаляем заглавное звено списка.

nsp = ksp = NULL; // устанавливаем указатели на NULL.

}



# Построение двунаправленного кольца с удаленным заглавным звеном, вывод содержимого кольцевого списка, вставка звена, удаление звена.

#include <iostream> //Подключаем библиотеку iostream, которая позволяет вводить и выводить данные

using namespace std; //Используем стандартное пространство имён

struct node//Определяем структуру node, представляющую звено списка

{

int elem; //Значение элемента звена

node\* sled; //Указатель на следующее звено

node\* pred; //Указатель на предыдущее звено

};

class Spisok //Определяем класс Spisok

{

private:

node\* nsp; //Указатель на начало списка

public:

Spisok() { nsp = NULL; } //Конструктор класса

//Методы класса

void BuiltRing(); //Построение кольца

void VyvodLeftRight(); //Вывод содержимого кольца по часовой стрелке

void VyvodRightLeft(); //Вывод содержимого кольца против часовой стрелки

void InsAfter(node\*, int); //Вставка нового звена после указанного звена

void InsBefore(node\*, int); //Вставка нового звена перед указанным звеном

void Delete(node\*); //Удаление указанного звена

void DelAfter(node\*); //Удаление звена, следующего за указанным звеном

node\* SearchRing(int); //Поиск звена по указанному значению элемента

void Ochistka(); //Удаление списка из памяти

};

void main() //Основная функция программы

{

setlocale(LC\_ALL, "Rus"); //Устанавливаем локаль для корректного отображения символов на кириллице

Spisok A; //Создаем объект класса Spisok

node\* Res; //Объявляем указатель на звено

int el, el1; // Объявляем две переменные для хранения значений элементов звенов

A.BuiltRing(); //Вызываем метод BuiltRing() для построения кольца

cout << "Содержимое кольца 'по часовой стрелке': \n";

A.VyvodLeftRight(); // Выводм содержимое кольца по часовой стрелке

cout << "Содержимое кольца 'против часовой стрелки': \n";

A.VyvodRightLeft(); // Выводим содержимое кольца против часовой стрелки

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

cin >> el; //Вводим элемент, после которого будет осуществлятся вставка

cout << "Введите элемент вставляемого звена: ";

cin >> el1; // Вводи элемент, который будем втавлять

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

cout << "осуществляется вставка: ";

cin >> el;// вводим элемент, перед которым будем вставлять

cout << "Введите элемент вставляемого звена: ";

cin >> el1; //вводим элемент, который будем втавлять

Res = A.SearchRing(el); //Вызываем метод SearchRing() для поиска звена по указанному значению элемента

if (Res != NULL) // Если звено найдено

{

A.InsBefore(Res, el1); A.VyvodLeftRight(); // вставляем новое звено перед ним и выводим содержимое кольца по часовой стрелке

}

else cout << "Звена с таким элементом в списке нет!\n"; // Иначе выводим сообщение об ошибке

Res = A.SearchRing(el); //Вызываем метод SearchRing() для поиска звена по указанному значению элемента

if (Res != NULL) // Если звено найдено

{

A.InsBefore(Res, el1); A.VyvodLeftRight(); // вставляем новое звено после него и выводим содержимое кольца по часовой стрелке

}

else cout << "Звена с таким элементом в списке нет!\n"; // Иначе выводим сообщение об ошибке

// Запрашиваем у пользователя элемент звена, которое необходимо удалить

cout << "Введите элемент звена, который ";

cout << "надо удалить: ";

cin >> el; // ввод элемента, который будем удалять

Res = A.SearchRing(el); // Ищем звено, которое необходимо удалить

if (Res != NULL) // Если звено найдено

{

A.Delete(Res); A.VyvodLeftRight(); // удаляем его и выводим содержимое кольца по часовой стрелке

}

else cout << "Звена с таким элементом в списке нет!\n"; // Иначе выводим сообщение об ошибке

// Запрашиваем у пользователя элемент звена после которого необходимо удалить элемент

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

cout << "осуществляется удаление: ";

cin >> el; // ввод элемента, после которого будет производиться удаление

Res = A.SearchRing(el); // Ищем звено, после которого будем удалять

if (Res != NULL) // Если звено найдено

{

A.DelAfter(Res); A.VyvodLeftRight(); // удаляем его и выводим содержимое кольца по часовой стрелке

}

else cout << "Звена с таким элементом в списке нет!\n"; // Иначе выводим сообщение об ошибке

A.Ochistka(); //очищам список

cout << "\n"; //вводим символ перехода на новую строку

system("PAUSE"); //ожидания нажатия любой клавиши

}

void Spisok::BuiltRing()

// Построение двунаправленного кольцевого списка nsp

// с удаленным заглавным звеном.

// nsp - указатель на заглавное звено списка.

{

node\* r; // указатель на текущее звено списка

int el; // переменная для ввода элементов списка

// Построим заглавное звено кольцевого списка.

nsp = new(node); // выделяем память под заглавное звено

r = nsp; // устанавливаем указатель на текущее звено на заглавное

(\*nsp).pred = NULL; (\*nsp).sled = NULL; // устанавливаем предыдущий и следующий элементы заглавного звена в NULL

cout << "Вводите элементы списка: \n";

cin >> el; //ввод элемента списка

while (el != 0) // пока не введен ноль

{

(\*r).sled = new (node); // выделяем память под следующее звено

(\*((\*r).sled)).pred = r; // устанавливаем указатель на предыдущее звено

r = (\*r).sled; // переносим указатель на текущее звено на следующее

(\*r).sled = NULL; (\*r).elem = el; // устанавливаем указатель на следующее звено в NULL и записываем элемент списка

cin >> el; // считываем следующий элемент

}

// А теперь - образуем кольцевой список!

if ((\*nsp).sled != NULL) // если список не пуст

{

(\*((\*nsp).sled)).pred = r; // устанавливаем указатель на предыдущее звено последнего элемента списка на текущий

(\*r).sled = (\*nsp).sled; // устанавливаем указатель на следующее звено последнего элемента на первый элемент списка

}

else // если список пуст

cout << "Кольцевой список пуст!\n";

}

void Spisok::VyvodLeftRight()

// Вывод содержимого двунаправленного кольцевого списка

// с удаленным заглавным звеном "по часовой стрелке".

// nsp - указатель на заглавное звено списка.

{

node\* r; // указатель на текущее звено списка.

cout << "Кольцевой список: ";

if ((\*nsp).sled != NULL) // Если список не пуст

{

cout << (\*((\*nsp).sled)).elem << " "; // Выводим значение первого элемента списка

r = (\*((\*nsp).sled)).sled; // Устанавливаем указатель на следующий элемент списка

while (r != (\*nsp).sled) // Проходим по всем элементам списка до возврата к первому элементу

{

cout << (\*r).elem << " "; // Выводим значение текущего элемента

r = (\*r).sled; // Устанавливаем указатель на следующий элемент списка

}

cout << endl;

}

else // Если список пуст

{

cout << "пуст!";

}

}

void Spisok::VyvodRightLeft()

// Вывод содержимого двунаправленного кольцевого списка

// с удаленным заглавным звеном "против часовой стрелки".

// nsp - указатель на заглавное звено списка.

{

node\* r; //указатель на текущее звено списка.

cout << "Кольцевой список: ";

if ((\*nsp).sled != NULL) // если список не пуст

{

cout << (\*((\*((\*nsp).sled)).pred)).elem << " "; // выводим последний элемент

r = (\*((\*((\*nsp).sled)).pred)).pred; // переходим к следующему элементу

while (r != (\*((\*nsp).sled)).pred)

{

cout << (\*r).elem << " "; r = (\*r).pred; // выводим оставшиеся элементы в обратном порядке

}

cout << endl; // символ перехода на новую строку

}

else cout << "пуст!"; // Если список пуст, то выводим соответствующее сообщение.

}

node\* Spisok::SearchRing(int el)

// Поиск элемента el в кольцевом двунаправленном списке

// с удаленным заглавным звеном.

// nsp - указатель на заглавное звено списка.

{

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

node\* p; // указатель на заглавное звено списка

node\* Res; // указатель на узел, содержащий элемент "el"

Res = NULL; // инициализация указателя результата

p = nsp; // инициализация указателя заглавного звена списка

if ((\*((\*p).sled)).elem == el) // проверяем первый элемент списка

Res = (\*p).sled; // сохраняем указатель на узел

else

{

q = (\*((\*p).sled)).sled; // указатель на следующий узел списка

// ищем элемент в списке, начиная со второго элемента

while (q != (\*p).sled && Res == NULL)

{

if ((\*q).elem == el) // если элемент найден

Res = q; // сохраняем указатель на узел

else

q = (\*q).sled; // иначе переходим к следующему узлу

}

}

return Res; // возвращаем указатель на узел с искомым элементом

}

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

// Вставление в кольцевой двунаправленный список звена

// с информационным полем el после звена, на которое

// указывает ссылка Res.

{

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

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

(\*q).elem = el; // устанавливаем информационное поле нового звена

(\*q).sled = (\*Res).sled; // настраиваем связи между узлами

(\*q).pred = (\*(\*Res).sled).pred; // настраиваем связи между узлами

(\*(\*Res).sled).pred = q; (\*Res).sled = q; // настраиваем связи между узлами

}

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

// Вставка в кольцевой двунаправленный список звена

// с информационным полем el перед звеном, на которое

// указывает ссылка Res.

// nsp - указатель на заглавное звено списка.

{

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

q = new(node); // выделение памяти для нового звена списка

(\*q).elem = el; // сохранение значения нового элемента в информационное поле нового звена

(\*q).sled = (\*(\*Res).pred).sled; // связывание нового звена с предыдущим звеном в списке

(\*q).pred = (\*Res).pred; // связывание нового звена с звеном, на которое указывает ссылка Res

(\*(\*Res).pred).sled = q; // связывание предыдущего звена со вставляемым

(\*Res).pred = q; // связывание звена, на которое указывает ссылка Res со вставляемым

if (Res == (\*nsp).sled) // Если Res указывает на первый элемент списка, то необходимо обновить указатель на первый элемент

{

(\*nsp).sled = q; // новый элемент становится первым в списке

}

}

void Spisok::Delete(node\* Res)

// Удаление из кольцевого двунаправленного списка

// звена, на которое указывает ссылка Res.

// nsp - указатель на заглавное звено списка.

{

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

{

// Если удаляем единственное звено в списке:

(\*nsp).sled = NULL; delete Res;

}

else

{

// Если удаляем звено, которое не единственное в списке:

(\*(\*Res).sled).pred = (\*Res).pred; // Перестраиваем связи между звеньями

(\*(\*Res).pred).sled = (\*Res).sled; // Перестраиваем связи между звеньями

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

// Если удаляем первое звено кольца:

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

delete Res;

}

}

void Spisok::DelAfter(node\* Res)

// Удаление из кольцевого двунаправленного списка звена,

// расположенного после звена, на которое указывает

// ссылка Res.

// nsp - указатель на заглавное звено списка.

{

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

if ((\*Res).sled == Res) // проверка, что Res - последнее звено кольца

{

(\*nsp).sled = NULL; // кольцо разрушено, список стал пустым

delete Res; // удаляем Res

}

else // Res - не последнее звено кольца

{

q = (\*Res).sled; // сохраняем указатель на удаляемое звено

(\*(\*(\*Res).sled).sled).pred = (\*(\*Res).sled).pred; // связываем предыдущее звено со следующим

(\*Res).sled = (\*(\*Res).sled).sled; // связываем текущее звено со следующим

if ((\*(\*nsp).sled).pred == Res)

// Удаляем "последнее" звено кольца.

(\*nsp).sled = (\*Res).sled; // если Res - предпоследнее звено, то следующее звено станет последним

delete q; // удаляем звено, которое находилось после Res

}

}

void Spisok::Ochistka()

{

node\* q, \* q1;

q = (\*((\*nsp).sled)).sled; // Изначально q указывает на первый элемент списка.

q1 = (\*q).sled; // q1 указывает на следующий элемент после q.

while (q1 != (\*((\*nsp).sled)).sled) // Проходим по списку и удаляем каждый элемент.

{

delete q;

q = q1;

q1 = (\*q1).sled;

}

delete q; // Удаляем последний элемент.

delete nsp; // Удаляем заглавное звено списка.

}

