Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

Тестирование кода

По дисциплине «Основы алгоритмизации и программирования»

На тему «Виды списков»

Выполнил:

Студент 1 курса 7 группы

Ленкевич Павел Андреевич

Преподаватель: Белодед Н. И.

2024, Минск

# Введение

В этом документе размещено тестирование кода трёх разных типов списка: ортогонального списка, кольцевого списка и линейного списка. Большинство строк имеют пояснение в комментариях, а после кода представлен результат запуска и работы программы. Для того, чтоб быстро увидеть определённый код или его результат работы воспользуйтесь навигацией.

# Тестирование кода с лекции

## Ортогональный список

#include <iostream>

using namespace std;

// Cтруктура звена Висюльки

struct nodeVis

{

int elem; // Элемент звена

nodeVis\* vniz; // Указатель на следующее звено

};

// Структура звена Гирлянды

struct nodeGir

{

int elem; // Элемент звена

nodeVis\* vniz; // Указатель на висюльку текущего звена

nodeGir\* sled; // Указатель на следующее звено Гирлянды

};

// Объявление класса для гирлянды и висюльки

class GirVis {

private:

nodeGir\* phead; // Голова гирлянды

nodeVis\* pheadVis; // Голова висюльки

void VisVyvod(); // Вывод висюльки

public:

GirVis() { phead = new(nodeGir); } // Конструктор

~GirVis() { delete phead; } // Деструктор

nodeVis\* VisPostr(); // Создание висюльки

nodeVis\* VisPoisk(int); // Поиск звена в висюльке

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

void VisVstav(nodeVis\*, int); // Вставка звена в висюльку после заданного звена

void Vis1Vstav(nodeVis\*, int); // Вставка звена в висюльку перед заданным звеном

void VisUdale(nodeVis\*); // Удаление звена в висюльке после заданного звена

void Vis1Udale(nodeVis\*); // Удаление заданного звена в висюльке

void GirPostr(); // Создание гирлянды

void GirVyvod(); // Вывод гирлянды

nodeGir\* GirPoisk(int); // Поиск звена в гирлянде

void OCHISTKA(); // Очистка гирлянды

void OCHISTKA1(); // Очистка висюльки

};

void main()

{

setlocale(LC\_ALL, "Rus"); // Установка локали для поддержки русского языка

GirVis A; // Создание объекта гирлянды

int el, elGir, elVis; // Объявление переменных для элемента, звена гирлянды и звена висюльки

nodeGir\* Res; // Объявление указателя на звено гирлянды

nodeVis\* ResVis; // Объявление указателя на звено висюльки

A.GirPostr(); // Создание гирлянды

A.GirVyvod(); // Вывод гирлянды

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

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

cin >> elGir; // Ввод элемента звена гирлянды

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

cout << "осуществляется вставка:\n";

cin >> elVis; // Ввод элемента звена висюльки

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

cin >> el; // Ввод вставляемого элемента

Res = A.GirPoisk(elGir); // Поиск звена гирлянды

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

{

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

ResVis = A.VisPoisk(elVis); // Поиск звена висюльки

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

A.VisVstav(ResVis, el); // Вставка нового звена после найденного

else cout << "Элемента в висюльке нет!\n"; // Если звено не найдено

}

else cout << "Элемента в гирлянде нет!\n"; // Если звено не найдено

A.GirVyvod(); // Вывод гирлянды

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

cin >> elGir;

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

cin >> elVis;

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

cin >> el;

Res = A.GirPoisk(elGir);

if (Res != NULL)

{

A.SetpheadVis((\*Res).vniz);

ResVis = A.VisPoisk(elVis);

if (ResVis != NULL);

A.SetpheadVis((\*Res).vniz);

ResVis = A.VisPoisk(elVis);

if (ResVis != NULL)

A.Vis1Vstav(ResVis, el);

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

}

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

A.GirVyvod();

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

cin >> elGir;

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

cin >> elVis;

Res = A.GirPoisk(elGir);

if (Res != NULL)

{

A.SetpheadVis((\*Res).vniz);

ResVis = A.VisPoisk(elVis);

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

A.VisUdale(ResVis);

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

}

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

A.GirVyvod();

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

cin >> elGir;

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

cin >> elVis;

Res = A.GirPoisk(elGir);

if (Res != NULL)

{

A.SetpheadVis((\*Res).vniz);

ResVis = A.VisPoisk(elVis);

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

A.Vis1Udale(ResVis);

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

}

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

A.GirVyvod();

A.OCHISTKA(); // Очистка гирлянды

cout << "\n";

system("PAUSE"); // Пауза перед завершением программы

}

void GirVis::OCHISTKA()

{

nodeGir\* q, \* q1;

q = phead; // Начинаем с головы гирлянды

q1 = (\*q).sled; // Следующее звено гирлянды

while (q1 != NULL) // Пока не достигнем конца гирлянды

{

q = q1; // Переходим к следующему звену

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

pheadVis = (\*q).vniz; // Устанавливаем голову висюльки

OCHISTKA1(); // Очищаем висюльку

delete q; // Удаляем текущее звено гирлянды

}

}

void GirVis::OCHISTKA1()

{

nodeVis\* q, \* q1;

q = pheadVis; // Начинаем с головы висюльки

q1 = (\*q).vniz; // Следующее звено висюльки

while (q1 != NULL) // Пока не достигнем конца висюльки

{

q = q1; // Переходим к следующему звену

q1 = (\*q1).vniz; // Обновляем указатель на следующее звено

delete q; // Удаляем текущее звено висюльки

}

}

void GirVis::GirPostr()

{

nodeGir\* t; // Объявление указателя на звено гирлянды

int el; // Объявление переменной для элемента

t = phead; // Начинаем с головы гирлянды

(\*t).sled = NULL; // Устанавливаем следующее звено гирлянды в NULL

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

cin >> el; // Ввод элемента

while (el != 0) // Пока введенный элемент не равен 0

{

(\*t).sled = new(nodeGir); // Создаем новое звено гирлянды

t = (\*t).sled; // Переходим к новому звену

(\*t).elem = el; // Устанавливаем элемент звена

(\*t).sled = NULL; // Устанавливаем следующее звено в NULL

(\*t).vniz = VisPostr(); // Создаем висюльку для текущего звена

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

cin >> el; // Ввод следующего элемента

}

}

nodeVis\* GirVis::VisPostr()

{

nodeVis\* t; // Объявление указателя на звено висюльки

int el; // Объявление переменной для элемента

pheadVis = new (nodeVis); // Создание новой висюльки

t = pheadVis; // Начинаем с головы висюльки

(\*t).vniz = NULL; // Устанавливаем следующее звено висюльки в NULL

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

cin >> el; // Ввод элемента

while (el != 0) // Пока введенный элемент не равен 0

{

(\*t).vniz = new(nodeVis); // Создаем новое звено висюльки

t = (\*t).vniz; // Переходим к новому звену

(\*t).elem = el; // Устанавливаем элемент звена

(\*t).vniz = NULL; // Устанавливаем следующее звено в NULL

cin >> el; // Ввод следующего элемента

}

return pheadVis; // Возвращаем голову висюльки

}

void GirVis::GirVyvod()

{

nodeGir\* t; // Объявление указателя на звено гирлянды

t = phead; // Начинаем с головы гирлянды

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

cout << "Гирлянда: "; // Выводим начальный текст

while (t != NULL) // Пока не достигнем конца гирлянды

{

cout << (\*t).elem << " "; // Выводим элемент текущего звена

pheadVis = (\*t).vniz; // Устанавливаем голову висюльки

VisVyvod(); // Выводим висюльку

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

}

}

nodeGir\* GirVis::GirPoisk(int el)

{

nodeGir\* t, \* r; // Объявление указателей на звено гирлянды

r = NULL; // Инициализация указателя r как NULL

t = phead; // Начинаем с головы гирлянды

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

while (t != NULL && r == NULL) // Пока не достигнем конца гирлянды и не найдем искомый элемент

if ((\*t).elem == el) r = t; // Если элемент текущего звена равен искомому, устанавливаем r на текущее звено

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

return r; // Возвращаем указатель на найденное звено или NULL, если звено не найдено

}

void GirVis::VisVyvod()

{

nodeVis\* t; // Объявление указателя на звено висюльки

t = pheadVis; // Начинаем с головы висюльки

t = (\*t).vniz; // Переходим к следующему звену

cout << "("; // Выводим открывающую скобку

while (t != NULL) // Пока не достигнем конца висюльки

{

cout << (\*t).elem << " "; // Выводим элемент текущего звена

t = (\*t).vniz; // Переходим к следующему звену

}

cout << ")"; // Выводим закрывающую скобку

}

nodeVis\* GirVis::VisPoisk(int el)

{

nodeVis\* t, \* r; // Объявление указателей на звено висюльки

r = NULL; // Инициализация указателя r как NULL

t = pheadVis; // Начинаем с головы висюльки

t = (\*t).vniz; // Переходим к следующему звену

while (t != NULL && r == NULL) // Пока не достигнем конца висюльки и не найдем искомый элемент

if ((\*t).elem == el) r = t; // Если элемент текущего звена равен искомому, устанавливаем r на текущее звено

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

return r; // Возвращаем указатель на найденное звено или NULL, если звено не найдено

}

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

{

nodeVis\* q; // Объявление указателя на звено висюльки

q = new(nodeVis); // Создание нового звена

(\*q).elem = el; // Установка элемента нового звена

(\*q).vniz = (\*r).vniz; // Установка следующего звена нового звена как следующее звено заданного звена

(\*r).vniz = q; // Установка следующего звена заданного звена как новое звено

}

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

{

nodeVis\* q; // Объявление указателя на звено висюльки

q = new(nodeVis); // Создание нового звена

(\*q).elem = (\*r).elem; // Копирование элемента из заданного звена в новое звено

(\*q).vniz = (\*r).vniz; // Установка следующего звена нового звена как следующее звено заданного звена

(\*r).elem = el; // Замена элемента заданного звена на новый элемент

(\*r).vniz = q; // Установка следующего звена заданного звена как новое звено

}

void GirVis::VisUdale(nodeVis\* r)

{

nodeVis\* q; // Объявление указателя на звено висюльки

q = (\*r).vniz; // Установка указателя на следующее звено после заданного

if ((\*r).vniz != NULL) // Если следующее звено существует

{

(\*r).vniz = (\*(\*r).vniz).vniz; // Установка следующего звена заданного звена как следующее звено для следующего звена

delete q; // Удаление следующего звена

}

else cout << "Звено с заданным элементом последнее!\n"; // Если следующего звена нет, выводим сообщение об ошибке

}

void GirVis::Vis1Udale(nodeVis\* r)

{

nodeVis\* g; // Объявление указателя на звено висюльки

if ((\*r).vniz != NULL) // Если следующее звено существует

{

g = (\*r).vniz; // Установка указателя на следующее звено

(\*r).elem = (\*(\*r).vniz).elem; // Копирование элемента из следующего звена в заданное звено

(\*r).vniz = (\*(\*r).vniz).vniz; // Установка следующего звена заданного звена как следующее звено для следующего звена

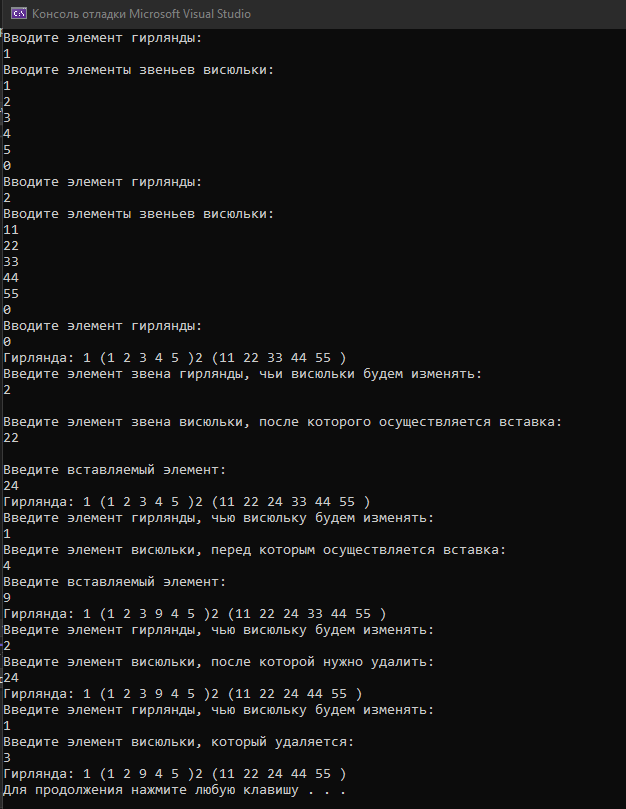
delete g; // Удаление следующего звена

}

else cout << "Не умею удалять последнее звено\n"; // Если следующего звена нет, выводим сообщение об ошибке

}

### Результат работы программы:



## Кольцевой список

#include <iostream>

using namespace std;

struct node // Определение структуры "узел"

{

int elem; // Элемент узла

node\* sled; // Указатель на следующий узел

};

class Spisok { // Определение класса "Список"

private:

node\* phead, \* Res; // Указатели на голову списка и результат поиска

public:

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

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

void POSTROENIE(); // Метод для построения списка

void VYVOD(); // Метод для вывода списка

node\* POISK(int); // Метод для поиска элемента в списке

void InsAfter(int); // Метод для вставки элемента после заданного

void InsBefore(int); // Метод для вставки элемента перед заданным

void Delete(); // Метод для удаления элемента

void DelAfter(); // Метод для удаления элемента после заданного

void OCHISTKA(); // Метод для очистки списка

};

void main()

{

setlocale(LC\_ALL, "Rus"); // Установка локали

Spisok A; // Создание объекта класса "Список"

int el, el1; // Объявление переменных для элементов

node\* Res\_Zn; // Объявление указателя на результат

A.POSTROENIE(); // Построение списка

A.VYVOD(); // Вывод списка

// Ввод элемента, после которого осуществляется вставка

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

cin >> el;

// Ввод элемента вставляемого звена

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

cin >> el1;

// Если элемент найден в списке

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

{

A.InsAfter(el1); // Вставка элемента после найденного

A.VYVOD(); // Вывод списка

}

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

// Ввод элемента, перед которым осуществляется вставка

cout << "\nВведите элемент звена, перед которы осуществляется вставка:\n";

cin >> el;

// Ввод элемента вставляемого звена

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

cin >> el1;

// Если элемент найден в списке

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

{

A.InsBefore(el1); // Вставка элемента перед найденным

A.VYVOD(); // Вывод списка

}

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

// Ввод элемента удаляемого звена

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

cin >> el;

// Если элемент найден в списке

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

{

A.Delete(); // Удаление найденного элемента

A.VYVOD(); // Вывод списка

}

else cout << "Звена с заданным элементом в кольце нет!"; // Если элемент не найден

// Ввод элемента, после которого нужно удалить

cout << "\nВведите элемент звена, после которого нужно удалить: ";

cin >> el;

// Если элемент найден в списке

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

{

A.DelAfter(); // Удаление элемента после найденного

A.VYVOD(); // Вывод списка

}

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

A.OCHISTKA(); // Очистка списка

cout << "\n";

system("PAUSE"); // Пауза перед завершением программы

}

void Spisok::POSTROENIE()

{

node\* t; // Объявление указателя на узел

int el; // Объявление переменной для элемента

t = phead; (\*t).sled = NULL; // Инициализация головы списка

cout << "Вводите элементы кольца: "; // Приглашение к вводу элементов

cin >> el; // Ввод первого элемента

while (el != 0) // Пока введенный элемент не равен 0

{

(\*t).sled = new(node); // Создание нового узла

t = (\*t).sled; (\*t).elem = el; // Присвоение элемента новому узлу

cin >> el; // Ввод следующего элемента

}

(\*t).sled = (\*phead).sled; // Замыкание списка в кольцо

}

void Spisok::VYVOD()

{

node\* t; // Объявление указателя на узел

t = (\*phead).sled; // Установка указателя на начало списка

cout << "Кольцо "; // Вывод начала сообщения

if (t != NULL) { // Если список не пуст

cout << (\*t).elem << " "; t = (\*t).sled; // Вывод первого элемента

while (t != (\*phead).sled) { // Пока не вернемся к началу списка

cout << (\*t).elem << " "; t = (\*t).sled; // Вывод элемента

}

}

else cout << "пусто\n"; // Если список пуст

}

node\* Spisok::POISK(int el) // Реализация метода поиска элемента в списке

{

node\* t; // Объявление указателя на узел

Res = NULL; t = (\*phead).sled; // Инициализация результата и установка указателя на начало списка

while ((\*t).sled != (\*phead).sled && Res == NULL) // Пока не вернемся к началу списка и не найдем элемент

if ((\*t).elem == el) Res = t; // Если элемент найден, сохраняем указатель на него

else t = (\*t).sled; // Если элемент не найден, переходим к следующему узлу

if (Res == NULL && (\*t).elem == el) // Если дошли до конца списка и элемент найден

Res = t; // Сохраняем указатель на него

return Res; // Возвращаем результат поиска

}

void Spisok::InsAfter(int el)

{

node\* q; // Объявление указателя на узел

q = new(node); // Создание нового узла

(\*q).elem = el; (\*q).sled = (\*Res).sled; // Присвоение элемента новому узлу и установка указателя на следующий узел

(\*Res).sled = q; // Установка указателя на новый узел в найденном узле

}

void Spisok::InsBefore(int el)

{

node\* q; // Объявление указателя на узел

q = new(node); // Создание нового узла

(\*q).elem = (\*Res).elem; (\*q).sled = (\*Res).sled; // Присвоение элемента новому узлу и установка указателя на следующий узел

(\*Res).elem = el; (\*Res).sled = q; // Присвоение нового элемента найденному узлу и установка указателя на новый узел

}

void Spisok::Delete()

{

node\* z, \* q;

if ((\*Res).sled != (\*phead).sled) // Если удаляемый элемент не является последним в списке

{

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

(\*Res).elem = (\*((\*Res).sled)).elem; // Копируем данные следующего элемента в текущий

(\*Res).sled = (\*((\*Res).sled)).sled; // Устанавливаем указатель на следующий элемент после следующего

delete q; // Удаляем следующий элемент

}

else if ((\*Res).sled == Res) // Если удаляемый элемент является единственным в списке

{

q = (\*phead).sled; (\*phead).sled = NULL; // Удаляем ссылку на элемент из головы списка

delete q; cout << "Кольцо пусто!"; // Удаляем элемент и выводим сообщение о том, что список пуст

}

else // Если удаляемый элемент является последним в списке, но не единственным

{

z = phead; q = (\*phead).sled; // Сохраняем указатели на голову списка и первый элемент

while (q != Res) // Пока не дойдем до удаляемого элемента

{

z = q; // Сохраняем указатель на текущий элемент

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

}

(\*z).sled = (\*((\*z).sled)).sled; // Устанавливаем указатель на следующий элемент после удаляемого

delete q; // Удаляем элемент

}

}

void Spisok::DelAfter()

{

node\* q;

if ((\*Res).sled != (\*phead).sled) // Если следующий за заданным элемент не является последним в списке

{

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

(\*Res).sled = (\*((\*Res).sled)).sled; // Устанавливаем указатель на следующий элемент после следующего

delete q; // Удаляем следующий элемент

}

else

if ((\*Res).sled == Res) // Если следующий за заданным элемент является единственным в списке

{

q = (\*phead).sled; (\*phead).sled = NULL; // Удаляем ссылку на элемент из головы списка

delete q; cout << "Кольцо пусто!"; // Удаляем элемент и выводим сообщение о том, что список пуст

}

else // Если следующий за заданным элемент является последним в списке, но не единственным

{

q = (\*phead).sled; // Сохраняем указатель на первый элемент

(\*Res).sled = (\*((\*Res).sled)).sled; // Устанавливаем указатель на следующий элемент после следующего

(\*phead).sled = (\*Res).sled; // Устанавливаем указатель на новый первый элемент в голове списка

delete q; // Удаляем элемент

}

}

void Spisok::OCHISTKA()

{

node\* q, \* q1;

q = phead; // Сохраняем указатель на голову списка

q1 = (\*q).sled; // Сохраняем указатель на первый элемент

do

{

q = q1; // Сохраняем указатель на текущий элемент

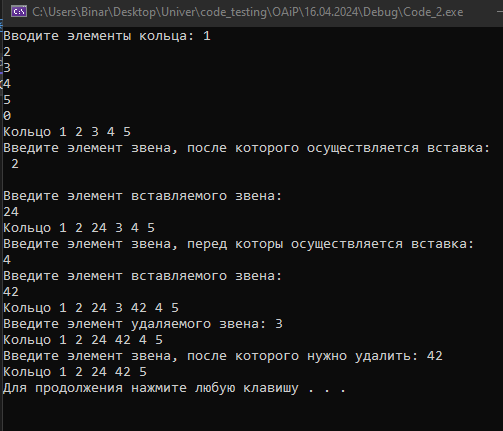
q1 = (\*q1).sled; // Переходим к следующему элементу

delete q; // Удаляем текущий элемент

} while (q1 != (\*phead).sled); // Повторяем, пока не вернемся к началу списка

}

### Результат работы программы:



## Кольцевой список №2

#include <iostream>

using namespace std;

// Определение структуры "node" для элементов списка

struct node

{

int elem; // Элемент списка

node\* sled; // Указатель на следующий элемент

node\* pred; // Указатель на предыдущий элемент

};

// Определение класса "Spisok" для работы со списком

class Spisok

{

private:

node\* nsp; // Указатель на начало списка

public:

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

void BuiltRing(); // Метод для создания кольцевого списка

void VyvodLeftRight(); // Метод для вывода списка слева направо

void VyvodRightLeft(); // Метод для вывода списка справа налево

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

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

void Delete(node\*); // Метод для удаления указанного элемента

void DelAfter(node\*); // Метод для удаления элемента после указанного

node\* SearchRing(int); // Метод для поиска элемента в списке

void Ochistka(); // Метод для очистки списка

};

// Главная функция

void main()

{

setlocale(LC\_ALL, "Rus");

Spisok A; // Создание объекта класса Spisok

node\* Res; // Указатель для работы с элементами списка

int el, el1;

A.BuiltRing(); // Создание кольцевого списка

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

A.VyvodLeftRight(); // Вывод списка слева направо

cout << "Содержимое кольца простив часовой стелки:\n";

A.VyvodRightLeft(); // Вывод списка справа налево

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

cin >> el;

cout << "Введите элемент вставляемого звена: ";

cin >> el1;

Res = A.SearchRing(el); // Поиск элемента в списке

if (Res != NULL)

{

A.InsAfter(Res, el1); A.VyvodLeftRight(); // Вставка элемента после найденного и вывод списка

}

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

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

cin >> el;

cout << "Введите элемент вставляемого звена: ";

cin >> el1;

Res = A.SearchRing(el); // Поиск элемента в списке

if (Res != NULL) {

A.InsBefore(Res, el1); A.VyvodLeftRight(); // Вставка элемента перед найденным и вывод списка

}

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

cout << "Введите элемент звена, который надо удалить: ";

cin >> el;

Res = A.SearchRing(el); // Поиск элемента в списке

if (Res != NULL)

{

A.Delete(Res); A.VyvodLeftRight(); // Удаление найденного элемента и вывод списка

}

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

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

cin >> el;

Res = A.SearchRing(el); // Поиск элемента в списке

if (Res != NULL)

{

A.DelAfter(Res); A.VyvodLeftRight(); // Удаление элемента после найденного и вывод списка

}

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

A.Ochistka(); // Очистка списка

cout << "\n";

system("PAUSE");

}

// Функция для создания кольцевого списка

void Spisok::BuiltRing()

{

node\* r;

int el;

nsp = new(node);

r = nsp; (\*nsp).pred = NULL; (\*nsp).sled = NULL;

cout << "Вводите элементы списка:\n";

cin >> el;

while (el != 0)

{

(\*r).sled = new(node);

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

(\*r).sled = NULL; (\*r).elem = el;

cin >> el;

}

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

{

(\*((\*nsp).sled)).pred = r; (\*r).sled = (\*nsp).sled;

}

else cout << "Кольцевой список пуст!\n";

}

// Функция для вывода списка слева направо

void Spisok::VyvodLeftRight()

{

node\* r;

cout << "Кольцевой список: ";

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

{

cout << (\*((\*nsp).sled)).elem << " ";

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

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

{

cout << (\*r).elem << " "; r = (\*r).sled;

}

cout << endl;

}

else cout << "пуст!";

}

// Функция для вывода списка справа налево

void Spisok::VyvodRightLeft()

{

node\* r;

cout << "Кольцевой список: ";

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

{

cout << (\*((\*((\*nsp).sled)).pred)).elem << " ";

r = (\*((\*((\*nsp).sled)).pred)).pred;

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

{

cout << (\*r).elem << " "; r = (\*r).pred;

}

cout << endl;

}

else cout << "пуст!";

}

// Функция для поиска элемента в списке

node\* Spisok::SearchRing(int el)

{

node\* q;

node\* p;

node\* Res;

Res = NULL;

p = nsp;

if ((\*((\*p).sled)).elem == el) Res = (\*p).sled;

else

{

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

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

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

else q = (\*q).sled;

}

return Res;

}

// Функция для вставки элемента после указанного

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

{

node\* q;

q = new(node);

(\*q).elem = el; (\*q).sled = (\*Res).sled;

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

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

}

// Функция для вставки элемента перед указанным

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

{

node\* q;

q = new(node);

(\*q).elem = el;

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

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

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

}

// Функция для удаления указанного элемента

void Spisok::Delete(node\* Res)

{

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

{

(\*nsp).sled = NULL;

delete Res;

}

else

{

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

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

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

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

delete Res;

}

}

// Функция для удаления элемента после указанного

void Spisok::DelAfter(node\* Res)

{

node\* q;

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

{

(\*nsp).sled = NULL;

delete Res;

}

else

{

q = (\*Res).sled;

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

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

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

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

delete q;

}

}

// Функция для очистки списка

void Spisok::Ochistka()

{

node\* q, \* q1;

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

q1 = (\*q).sled;

while (q1 != (\*((\*nsp).sled)).sled)

{

delete q; q = q1; q1 = (\*q1).sled;

}

delete q;

delete nsp;

}

### Результат работы программы:

