Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Реферат

По дисциплине «Основы алгоритмизации и программирования»

На тему «Тестирование кода из лекции»

Выполнил:

Студент 1 курса 8 группы

Лужецкий Владислав Константинович

Проверил:

Белодед Николай Иванович

2024, Минск

Оглавление

[Тестирование кода с гирляндой 2](#_Toc164211947)

[Тестирование кода со списком 9](#_Toc164211948)

[Тестирование кода с кольцевым списком 15](#_Toc164211949)

[Тестирование стека со значением вершины 21](#_Toc164211950)

[Тестирование стека, проверяющего правильность постановки скобок 24](#_Toc164211951)

[Тестирование дека 26](#_Toc164211952)

# Тестирование кода с гирляндой

#include<iostream>

using namespace std;

struct nodeVis

{

int elem; // Значение элемента

nodeVis\* vniz; // Указатель на нижний узел в связном списке

};

struct nodeGir

{

int elem; // Значение элемента

nodeVis\* vniz; // Указатель на висячий узел в связном списке

nodeGir\* sled; // Указатель на следующий узел в связном списке

};

class GirVis

{

private:

nodeGir\* phead; // Указатель на начало связного списка для гирлянды

nodeVis\* pheadVis; // Указатель на начало связного списка для висячих узлов

void VisVyvod(); // Приватная функция для вывода висячих узлов

public:

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

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

nodeVis\* VisPostr(); // Функция для построения висячих узлов

nodeVis\* VisPoisk(int); // Функция для поиска висячего узла по значению

void SetpheadVis(nodeVis\* r) { pheadVis = r; } // Установка указателя на начало висячих узлов

void VisVstav(nodeVis\*, int); // Функция для вставки висячего узла

void Vis1Vstav(nodeVis\*, int); // Функция для вставки первого висячего узла

void VisUdale(nodeVis\*); // Функция для удаления висячего узла

void Vis1Udale(nodeVis\*); // Функция для удаления первого висячего узла

void GirPostr(); // Функция для построения гирлянды

void GirVyvod(); // Функция для вывода гирлянды

nodeGir\* GirPoisk(int); // Функция для поиска узла гирлянды по значению

void OCHISTKA(); // Функция для очистки гирлянды

void OCHISTKA1(); // Функция для очистки первого висячего узла

};

void main()

{

setlocale(LC\_ALL, "Rus");

GirVis A; // Создание объекта класса с именем A

int el, elGir, elVis; // Переменные для значений элементов

nodeGir\* Res; // Указатель для результата поиска узла гирлянды

nodeVis\* ResVis; // Указатель для результата поиска висячего узла

A.GirPostr(); // Построение гирлянды

A.GirVyvod(); // Вывод гирлянды на экран

cout << "\nВведите элемент звена гирлянды, чьи висюльки будем изменять:\n";

cin >> elGir; // Ввод значения

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

cin >> elVis; // Ввод значения

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

cin >> el; // Ввод значения

Res = A.GirPoisk(elGir); // Поиск узла гирлянды по значению

if (Res != NULL) // Если узел != NULL

{

A.SetpheadVis((\*Res).vniz); // Установка указателя на начало висячих узлов

ResVis = A.VisPoisk(elVis); // Поиск висячего узла по значению

if (ResVis != NULL) // Если висячий узел != NULL

A.VisVstav(ResVis, el); // Вставка висячего узла

else

cout << "Элемента в висюльке нет!\n";

}

else

cout << "Элемента в гирлянде нет\n";

A.GirVyvod(); // Вывод гирлянды на экран

A.OCHISTKA(); // Очистка гирлянды

cout << "\n";

system("PAUSE"); // Ожидание нажатия клавиши перед закрытием консоли

}

void GirVis::OCHISTKA()

{

nodeGir\* q, \* q1;

q = phead; // Установка указателя q на начало гирлянды

q1 = (\*q).sled; // Установка указателя q1 на следующий узел гирлянды

while (q1 != NULL)

{

q = q1; // Перемещение указателя q на следующий узел

q1 = (\*q1).sled; // Перемещение указателя q1 на следующий узел

pheadVis = (\*q).vniz; // Установка указателя pheadVis на начало висячих узлов текущего звена гирлянды

OCHISTKA1(); // Вызов метода для очистки висячих узлов текущего звена гирлянды

delete q; // Удаление текущего узла гирлянды

}

}

void GirVis::OCHISTKA1()

{

nodeVis\* q, \* q1;

q = pheadVis; // Установка указателя q на начало висячих узлов

q1 = (\*q).vniz; // Установка указателя q1 на следующий висячий узел

while (q1 != NULL)

{

q = q1; // Перемещение указателя q на следующий висячий узел

q1 = (\*q1).vniz; // Перемещение указателя q1 на следующий висячий узел

delete q; // Удаление текущего висячего узла

}

}

void GirVis::GirPostr()

{

nodeGir\* t;

int el;

t = phead; // Установка указателя t на начало гирлянды

(\*t).sled = NULL; // Установка указателя следующего узла равным NULL

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

cin >> el; // Ввод элемента гирлянды с клавиатуры

while (el != 0)

{

(\*t).sled = new (nodeGir); // Выделение памяти под следующий узел

t = (\*t).sled; // Перемещение указателя t на новый узел

(\*t).elem = el; // Запись значения элемента в узел

(\*t).sled = NULL; // Установка указателя следующего узла равным NULL

(\*t).vniz = VisPostr(); // Вызов метода для построения висячих узлов текущего звена гирлянды

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

cin >> el; // Ввод следующего элемента

}

}

nodeVis\* GirVis::VisPostr()

{

nodeVis\* t;

int el;

pheadVis = new (nodeVis); // Выделение памяти для узла висячих узлов

t = pheadVis;

(\*t).vniz = NULL; // Установка указателя на следующий висячий узел равным NULL

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

cin >> el; // Ввод элемента звена висячих узлов

while (el != 0)

{

(\*t).vniz = new (nodeVis); // Выделение памяти для висячего узла

t = (\*t).vniz; // Перемещение указателя t на новый висячий узел

(\*t).elem = el; // Запись значения элемента в узел

(\*t).vniz = NULL; // Установка указателя на следующий висячий узел равным NULL

cin >> el; // Ввод следующего элемента звена висячих узлов

}

return pheadVis; // Возвращение указателя на начало висячих узлов

}

void GirVis::GirVyvod()

{

nodeGir\* t;

t = phead;

t = (\*t).sled;

cout << "Гирлянда: ";

while (t != NULL)

{

cout << (\*t).elem << " "; // Вывод значения элемента гирлянды

pheadVis = (\*t).vniz; // Установка указателя pheadVis на начало висячих узлов текущего звена гирлянды

VisVyvod(); // Вызов метода для вывода висячих узлов текущего звена гирлянды

t = (\*t).sled; // Переход к следующему узлу гирлянды

}

}

nodeGir\* GirVis::GirPoisk(int el)

{

nodeGir\* t, \* r;

r = NULL;

t = phead;

t = (\*t).sled; // Переход к первому узлу гирлянды, пропуская заголовочный узел

while (t != NULL && r == NULL)

{

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

r = t; // Если найден узел с искомым элементом, устанавливаем r на этот узел

else

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

}

return r; // Возвращаем указатель на найденный узел или NULL, если не найден

}

void GirVis::VisVyvod()

{

nodeVis\* t;

t = pheadVis;

t = (\*t).vniz; // Переход к первому висячему узлу, пропуская заголовочный узел

cout << "(";

while (t != NULL)

{

cout << (\*t).elem << " "; // Вывод значения элемента висячего узла

t = (\*t).vniz; // Переход к следующему висячему узлу

}

cout << ")";

}

nodeVis\* GirVis::VisPoisk(int el)

{

nodeVis\* t, \* r;

r = NULL;

t = pheadVis;

t = (\*t).vniz; // Переход к первому висячему узлу, пропуская заголовочный узел

while (t != NULL && r == NULL)

{

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

r = t; // Если найден узел с искомым элементом, устанавливаем r на этот узел

else

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

}

return r; // Возвращаем указатель на найденный узел или NULL, если не найден

}

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

{

nodeVis\* q;

q = new (nodeVis); // Выделение памяти для нового висячего узла

(\*q).elem = el; // Присваивание новому узлу значения элемента

(\*q).vniz = (\*r).vniz; // Установка указателя нового узла на следующий висячий узел после r

(\*r).vniz = q; // Установка указателя r на новый висячий узел

}

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

{

nodeVis\* q;

q = new (nodeVis); // Выделение памяти для нового висячего узла

(\*q).elem = (\*r).elem; // Присваивание новому узлу значения элемента r

(\*q).vniz = (\*r).vniz; // Установка указателя нового узла на следующий висячий узел после r

(\*r).elem = el; // Присваивание r нового значения элемента

(\*r).vniz = q; // Установка указателя r на новый висячий узел

}

void GirVis::VisUdale(nodeVis\* r)

{

nodeVis\* q;

q = (\*r).vniz; // Сохранение указателя на следующий висячий узел после r

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

{

(\*r).vniz = (\*(\*r).vniz).vniz; // Установка указателя r на второй висячий узел после r

delete q; // Освобождение памяти

}

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

}

void GirVis::Vis1Udale(nodeVis\* r)

{

nodeVis\* g;

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

{

g = (\*r).vniz; // Сохранение указателя на следующий висячий узел после r

(\*r).elem = (\*(\*r).vniz).elem; // Присваивание r значения элемента следующего висячего узла после r

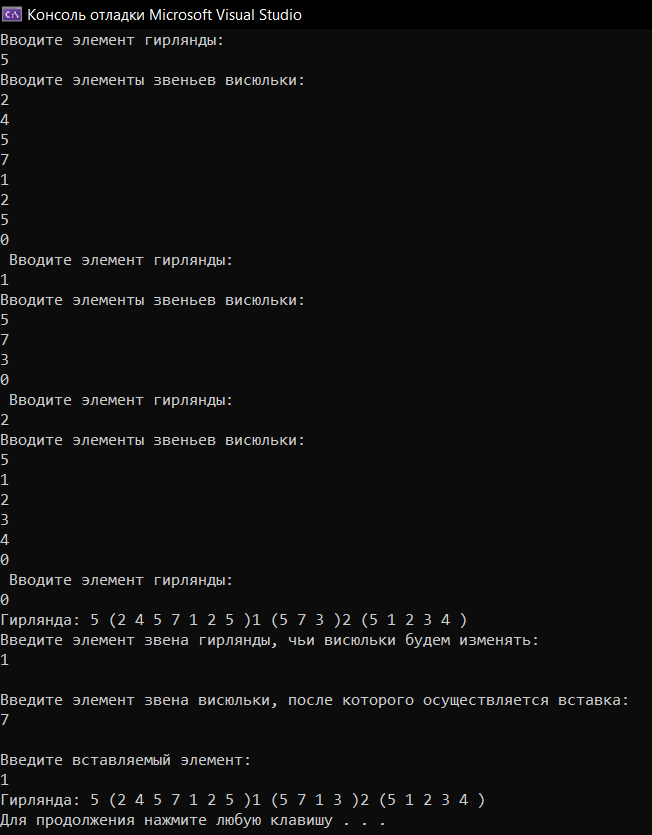
(\*r).vniz = (\*(\*r).vniz).vniz; // Установка указателя r на второй висячий узел после r

delete g; // Освобождение памяти

}

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

}



# Тестирование кода со списком

#include <iostream>

using namespace std;

struct node {

int elem;

node\* sled;

};

class Spisok {

private:

node\* phead; // Указатель на заголовочный узел списка

node\* Res; // Дополнительный указатель

public:

Spisok() { phead = new(node); Res = NULL; } // Конструктор класса, инициализирует заголовочный узел

~Spisok() { delete phead; } // Деструктор класса, освобождает память, выделенную для заголовочного узла

void POSTROENIE(); // Функция для построения структуры данных

void VYVOD(); // Функция для вывода информации

node\* POISK(int); // Функция для поиска узла с заданным значением

void InsAfter(int); // Функция для вставки нового узла после элемента с заданным значением

void InsBefore(int); // Функция для вставки нового узла перед элементом с заданным значением

void Delete(); // Функция для удаления текущего узла

void DelAfter(); // Функция для удаления узла, следующего за текущим узлом

void OCHISTKA(); // Функция для очистки и освобождения памяти, занятой структурой данных

};

void main()

{

setlocale(LC\_ALL, "rus");

Spisok A; // Создание объекта класса Spisok

int el, el1; // Объявление переменных el и el1

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

A.POSTROENIE(); // Вызов функции POSTROENIE

A.VYVOD(); // Вызов функции для вывода данных

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

cout << "осуществляется вставка: ";

cin >> el; // Ввод значения

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

cin >> el1; // Ввод значения

if (A.POISK(el) != NULL) // Проверка, элемент != NULL

{

A.InsAfter(el1); // Вызов функции для вставки нового узла после указанного элемента

A.VYVOD(); // Вызов функции для вывода данных

}

else

{

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

}

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

cout << "осуществляется вставка: ";

cin >> el; // Ввод значения

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

cin >> el1; // Ввод значения

if (A.POISK(el) != NULL) // Проверка, элемент != NULL

{

A.InsBefore(el1); // Вызов функции для вставки нового узла перед указанным элементом

A.VYVOD(); // Вызов функции для вывода обновленных данных

}

else

{

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

}

cout << "\nВведите элемент удаляемого звена";

cin >> el; // Ввод значения

if (A.POISK(el) != NULL) // Проверка, элемент != NULL

{

A.Delete(); // Вызов функции для удаления указанного узла

A.VYVOD(); // Вызов функции для вывода обновленных данных

}

else

{

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

}

cout << "Введите элемент звена ";

cout << "после которого нужно удалять: ";

cin >> el; // Ввод значения

if (A.POISK(el) != NULL) // Проверка, элемент != NULL

{

A.DelAfter(); // Вызов функции для удаления узла, следующего за указанным элементом

A.VYVOD(); // Вызов функции для вывода обновленных данных

}

else

{

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

}

A.OCHISTKA(); // Вызов функции для очистки памяти, занятой списком

cout << "\n";

system("PAUSE"); // Ожидание нажатия клавиши перед завершением программы

}

void Spisok::POSTROENIE()

{

node\* t;

int el;

t = phead; // Указатель t указывает на головн. узел

(\*t).sled = NULL; // Устанавливаем следующий указатель равным NULL

cout << "Вводите элементы кольца: ";

cin >> el;

while (el != 0)

{

(\*t).sled = new(node); // Выделяем память для следующего указателя

t = (\*t).sled; // Перемещаем указатель t на новый узел

(\*t).elem = el; // Присваиваем элементу нового узла значение el

cin >> el;

}

(\*t).sled = (\*phead).sled; // Устанавливаем следующий указатель на головн. узел, чтобы закольцевать список

}

void Spisok::VYVOD()

{

node\* t;

t = (\*phead).sled; // Указатель t указывает на первый узел после головн.

cout << "Кольцо: ";

if (t != NULL)

{

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

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

while (t != (\*phead).sled)

{

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

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

}

}

else

{

cout << "пусто!\n";

}

}

node\* Spisok::POISK(int el)

{

node\* t;

Res = NULL; // Переменной Res присваивается значение NULL

t = (\*phead).sled; // Указатель t указывает на первый узел (после головн.)

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

{

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

{

Res = t; // Если значение элемента текущего узла равно el, присваиваем Res значение t

}

else

{

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

}

}

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

{

Res = t; // Если Res равно NULL и значение элемента текущего узла равно el, присваиваем Res значение t

}

return Res; // Возвращаем значение Res

}

void Spisok::InsAfter(int el)

{

node\* q;

q = new(node); // Выделение памяти для указателя q

(\*q).elem = el; // Присваиваем элементу узла значение el

(\*q).sled = (\*Res).sled; // Устанавливаем следующий указатель нового узла равным следующему указателю узла Res

(\*Res).sled = q; // Устанавливаем следующий указатель узла Res равным q

}

void Spisok::InsBefore(int el)

{

node\* q;

q = new(node); // Выделение памяти для указателя q

(\*q).elem = (\*Res).elem; // Присваиваем элементу нового узла значение элемента узла Res

(\*q).sled = (\*Res).sled; // Устанавливаем следующий указатель нового узла равным следующему указателю узла Res

(\*Res).elem = el; // Присваиваем элементу узла Res значение el

(\*Res).sled = q; // Устанавливаем следующий указатель узла Res равным q

}

void Spisok::Delete()

{

node\* z, \* q;

if ((\*Res).sled != (\*phead).sled) // Если следующий указатель узла Res != следующему указателю головн. узла

{

q = (\*Res).sled; // Указатель q указывает на следующий узел после Res

(\*Res).elem = (\*((\*Res).sled)).elem; // Значению элемента узла Res присваивается значение элемента следующего узла

(\*Res).sled = (\*((\*Res).sled)).sled; // Устанавливаем следующий указатель узла Res равным следующему указателю следующего узла

delete q; // Удаляем следующий узел

}

else if ((\*Res).sled == Res) // Иначе если следующий указатель узла Res равен Res

{

q = (\*phead).sled; // Указатель q указывает на следующий узел после головн. узла

(\*phead).sled = NULL; // Устанавливаем следующий указатель головн. узла равным NULL

delete q; // Удаляем следующий узел

cout << "Кольцо пусто!";

}

else {

z = phead; // Указатель z указывает на головной узел

q = (\*phead).sled; // Указатель q на следующий узел после головн. узла

while (q != Res)

{

z = q; // Указатель z переходит к следующему узлу

q = (\*q).sled; // Указатель q переходит к следующему узлу

}

(\*z).sled = (\*((\*z).sled)).sled; // Устанавливаем следующий указатель предыдущего узла равным следующему указателю следующего узла

delete q; // Удаляем узел Res

}

}

void Spisok::DelAfter()

{

node\* q;

if ((\*Res).sled != (\*phead).sled) // Если следующий указатель узла Res != следующему указателю головного узла

{

q = (\*Res).sled; // Указатель q указывает на следующий узел после Res

(\*Res).sled = (\*((\*Res).sled)).sled; // Устанавливаем следующий указатель узла Res = следующему указателю следующего узла

delete q; // Удаляем следующий узел

}

else if ((\*Res).sled == Res) // Иначе, если следующий указатель узла Res равен Res

{

q = (\*phead).sled; // Указатель q указывает на следующий узел после головного узла

(\*phead).sled = NULL; // Устанавливаем следующий указатель головного узла = NULL

delete q; // Удаляем следующий узел

cout << "Кольцо пусто!";

}

else

{

q = (\*phead).sled; // Указатель q указывает на следующий узел после головного узла

(\*Res).sled = (\*((\*Res).sled)).sled; // Устанавливаем следующий указатель узла Res = следующему указателю следующего узла

(\*phead).sled = (\*Res).sled; // Устанавливаем следующий указатель головного узла = следующему указателю узла Res

delete q; // Удаляем узел, следующий за Res

}

}

void Spisok::OCHISTKA()

{

node\* q, \* q1;

q = phead;

q1 = (\*q).sled;

do {

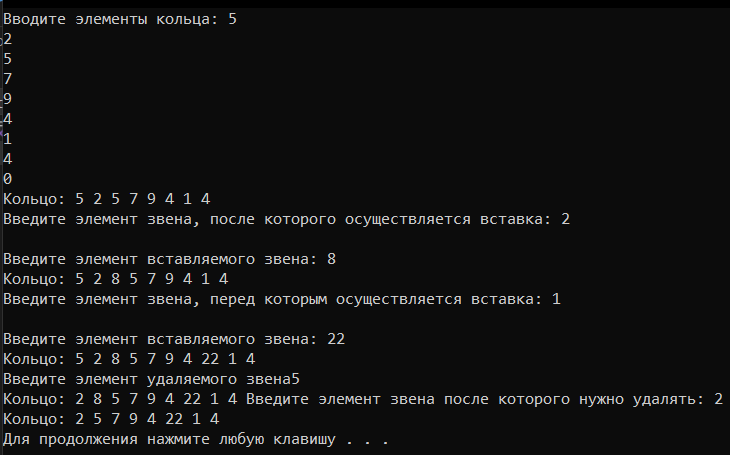
q = q1; // Указатель q переходит к следующему узлу

q1 = (\*q1).sled; // Указатель q1 переходит к следующему узлу

delete q; // Удаляем текущий узел

} while (q1 != (\*phead).sled); // Повторяем, пока q1 не станет равным следующему указателю головного узла

}



# Тестирование кода с кольцевым списком

#include <iostream>

using namespace std;

struct node

{

int elem;

node\* sled;

node\* pred;

};

class Spisok

{

private:

node\* nsp; // Указатель на начальный узел (начало списка)

public:

Spisok() { nsp = NULL; } //Конструктор класса, инициализирует указатель nsp значением NULL

void BuiltRing(); // функция для построения кольцевого списка

void VyvodLeftRight(); // функция для вывода элементов списка слева направо

void VyvodRightLeft(); // функция для вывода элементов списка справа налево

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

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

void Delete(node\*); // функция для удаления указанного узла

void DelAfter(node\*); // функция для удаления узла, следующего за указанным узлом

node\* SearchRing(int); // функция для поиска узла с указанным значением в кольцевом списке

void Ochistka(); // функция для очистки кольцевого списка

};

void main()

{

setlocale(LC\_ALL, "Rus");

Spisok A;

node\* Res;

int el, el1;

A.BuiltRing(); // Построение кольцевого списка

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

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

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

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

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

cout << "осуществляется вставка: ";

cin >> el;

cout << "Введите элемент вставляемого звена: ";

cin >> el1;

Res = A.SearchRing(el);

if (Res != NULL)

{

A.InsAfter(Res, el1); // Вставка нового звена после указанного звена

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

}

else

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

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

cout << "осуществляется вставка: ";

cin >> el;

cout << "Введите элемент вставляемого звена: ";

cin >> el1;

Res = A.SearchRing(el);

if (Res != NULL)

{

A.InsBefore(Res, el1); // Вставка нового звена перед указанным звеном

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

}

else

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

cout << "Введите элемент звена, который ";

cout << "надо удалить: ";

cin >> el;

Res = A.SearchRing(el);

if (Res != NULL)

{

A.Delete(Res); // Удаление указанного звена

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

}

else

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

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

cout << "осуществляется удаление: ";

cin >> el;

Res = A.SearchRing(el);

if (Res != NULL)

{

A.DelAfter(Res); // Удаление звена, следующего за указанным звеном

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

}

else

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

A.Ochistka(); // Очистка кольцевого списка

cout << "\n";

system("PAUSE");

}

void Spisok::BuiltRing()

{

node\* r;

int el;

nsp = new(node); // Выделения памяти для нового звена

r = nsp;

(\*nsp).pred = NULL; (\*nsp).sled = NULL; // Инициализация указателей предыдущего и следующего звена

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

cin >> el;

while (el != 0)

{

(\*r).sled = new (node); // Выделения памяти для нового звена

(\*((\*r).sled)).pred = r; // Установка указателя предыдущего звена

r = (\*r).sled;

(\*r).sled = NULL; (\*r).elem = el; // Инициализация указателя следующего звена и значения элемента

cin >> el;

}

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

{

(\*((\*nsp).sled)).pred = r; // Установка указателя предыдущего звена последнего звена на текущее звено

(\*r).sled = (\*nsp).sled; // Установка указателя следующего звена текущего звена на первое звено

}

else

cout << "Кольцевой список пуст!\n";

}

void Spisok::VyvodLeftRight()

{

node\* r;

cout << "Кольцевой список: ";

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

{

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

r = (\*((\*nsp).sled)).sled; // Устанавливается указатель на следующее звено

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

{

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

r = (\*r).sled; // Переход к следующему звену

}

cout << endl;

}

else

cout << "пуст!";

}

void Spisok::VyvodRightLeft()

{

node\* r;

cout << "Кольцевой список: ";

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

{

cout << (\*((\*((\*nsp).sled)).pred)).elem << " "; // Выводится значение элемента последнего звена

r = (\*((\*((\*nsp).sled)).pred)).pred; // Устанавливается указатель на предыдущее звено, указатель r, будет использоваться для перехода по звеньям.

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

{

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

}

cout << endl;

}

else cout << "пуст!";

}

node\* Spisok::SearchRing(int el)

{

node\* q; // Указатель для перебора звеньев

node\* p; // Указатель на начало списка

node\* Res; // Указатель на найденное звено

Res = NULL; p = nsp;

if ((\*((\*p).sled)).elem == el) Res = (\*p).sled; // Проверка первого звена списка

else

{

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

while (q != (\*p).sled && Res == NULL) // Поиск элемента в остальных звеньях

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

else q = (\*q).sled;

}

return Res;

}

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

{

node\* q; // Новое звено

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

(\*q).elem = el; (\*q).sled = (\*Res).sled; // Установка значения элемента и указателя следующего звена

(\*q).pred = (\*(\*Res).sled).pred; // Установка указателя предыдущего звена

(\*(\*Res).sled).pred = q; (\*Res).sled = q; // Установка указателя предыдущего звена и следующего звена

}

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

{

node\* q; // Новое звено

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

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

(\*q).sled = (\*(\*Res).pred).sled; (\*q).pred = (\*Res).pred; // Установка указателя следующего и предыдущего звеньев

(\*(\*Res).pred).sled = q; (\*Res).pred = q; // Установка указателя следующего и предыдущего звена для существующих звеньев

if (Res == (\*nsp).sled) (\*nsp).sled = q; // Установка указателя следующего звена для начала списка

}

void Spisok::Delete(node\* Res)

{

if ((\*Res).sled == Res) // Если Res является единственным звеном в списке

{

(\*nsp).sled = NULL; delete Res; // Удаление Res и установка указателя начала списка в NULL

}

else

{

(\*(\*Res).sled).pred = (\*Res).pred; // Обновление указателя предыдущего звена для следующего звена после Res

(\*(\*Res).pred).sled = (\*Res).sled; // Обновление указателя следующего звена для предыдущего звена перед Res

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

(\*nsp).sled = (\*Res).sled; // Если Res является началом списка, обновление указателя начала списка

delete Res; // Удаление Res

}

}

void Spisok::DelAfter(node\* Res)

{

node\* q;

if ((\*Res).sled == Res) // Если Res является единственным звеном в списке

{

(\*nsp).sled = NULL; delete Res; // Удаление Res и установка указателя начала списка в NULL

}

else

{

q = (\*Res).sled; // q = следующее звено после Res

(\*(\*(\*Res).sled).sled).pred = (\*(\*Res).sled).pred; // Обновление указателя предыдущего звена для второго следующего звена после Res

(\*Res).sled = (\*(\*Res).sled).sled; // Обновление указателя следующего звена для Res

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

(\*nsp).sled = (\*Res).sled; // Если следующее звено после начала списка является Res, обновление указателя начала списка

delete q; // Удаление следующего звена после Res

}

}

void Spisok::Ochistka()

{

node\* q, \* q1;

q = (\*((\*nsp).sled)).sled; // q = второе звено списка

q1 = (\*q).sled; // q1 = третье звено списка

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

{

delete q; // Удаление текущего звена

q = q1; // Перемещение q на следующее звено

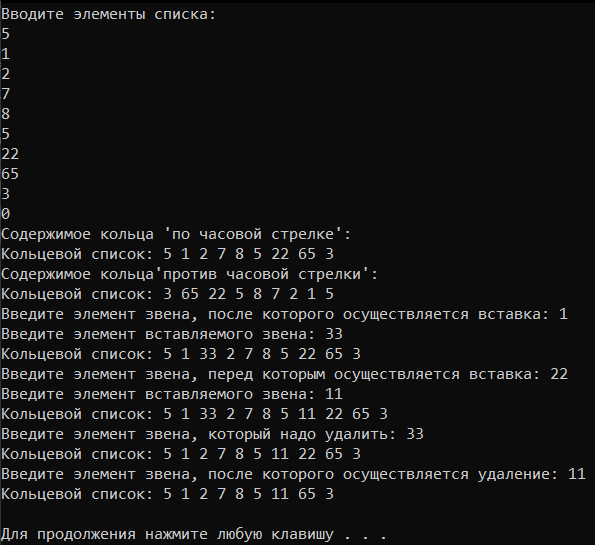
q1 = (\*q1).sled; // Перемещение q1 на следующее звено

}

delete q; // Удаление последнего звена

delete nsp; // Удаление начала списка

}



# Тестирование стека со значением вершины

#include<iostream>

using namespace std;

struct node

{

int elem;

node\* sled;

};

class Spisok

{

private:

node\* stk; // Указатель на вершину стека

int klad; // Значение вершины стека

public:

Spisok() { stk = NULL; } // Конструктор класса, инициализирует stk значением NULL

int Set\_Stack() { return klad; } // Возвращает значение klad

void POSTROENIE(); // Функция для построения стека

void VYVOD(); // Функция для вывода стека

void W\_S(int); // Функция для добавления элемента в стек

void YDALENIE(); // Функция для удаления элемента из стека

void OCHISTKA(); // Функция для очистки стека

};

void main()

{

setlocale(LC\_ALL, "Rus");

Spisok A; // Создание объекта класса Spisok

int el; // Переменная для ввода элемента

int t; // Переменная для хранения извлеченного элемента

A.POSTROENIE(); // Построение стека

A.VYVOD(); // Вывод стека

cout << "Введите вставляемый элемент: ";

cin >> el; // Ввод элемента

A.W\_S(el); // Добавление элемента в стек

A.VYVOD(); // Вывод стека

cout << "Удалим элемент из стека.\n";

A.YDALENIE(); // Удаление элемента из стека

t = A.Set\_Stack(); // Получение значения klad

cout << "Из стека было извлечено число... " << t << endl; // Вывод извлеченного элемента

A.VYVOD(); // Вывод стека

A.OCHISTKA(); // Очистка стека

cout << "\n";

system("PAUSE");

}

void Spisok::POSTROENIE()

{

node\* t;

int el;

cout << "Вводите элементы стека: ";

cin >> el; // Ввод элемента

while (el != 0) // Цикл продолжается, пока не будет введен 0

{

t = new (node); // Выделение памяти для нового узла

(\*t).elem = el; (\*t).sled = stk; // Присваивание элементу значения и установка указателя следующего звена на предыдущий узел

stk = t; // Обновление указателя вершины стека

cin >> el;

}

}

void Spisok::VYVOD()

{

node\* t;

cout << "Содержимое стека: ";

t = stk; // Присваивание t указателя на вершину стека

while (t != NULL)

{

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

t = (\*t).sled; // Переход к следующему узлу

}

cout << endl;

}

void Spisok::W\_S(int el)

{

node\* q;

q = new (node); // Выделение памяти для нового узла

(\*q).elem = el; (\*q).sled = stk; // Присваивание элементу значения и установка указателя следующего звена на предыдущий узел

stk = q; // Обновление указателя вершины стека

}

void Spisok::YDALENIE()

{

node\* q;

if (stk == NULL)

cout << "Стек пуст!\n";

else

{

klad = (\*stk).elem; // Сохранение значения вершины стека

q = stk; // Присваивание q указателя на вершину стека

stk = (\*stk).sled; // Обновление указателя вершины стека

delete q; // Удаление предыдшего узла

}

}

void Spisok::OCHISTKA()

{

node\* t, \* q;

t = stk; // Присваивание t указателя на вершину стека

if (t != NULL)

{

q = (\*t).sled; // Присваивание q указателя на следующий узел

while (q != NULL)

{

delete t; // Удаление текущего узла

t = q; // Переход к следующему узлу

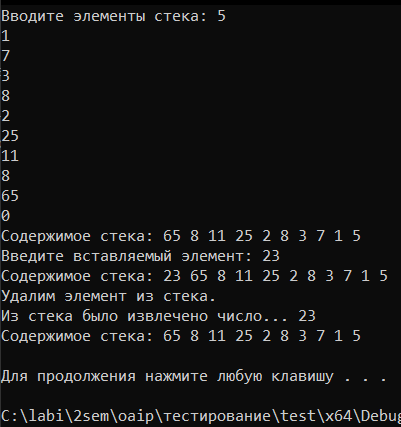
q = (\*q).sled; // Переход к следующему узлу

}

delete t; // Удаление последнего узла

}

}



# Тестирование стека, проверяющего правильность постановки скобок

#include <iostream>

#include <string.h>

using namespace std;

const int MAXSIZE = 100;

struct Stack

{

char data[MAXSIZE]; // Массив для хранения элементов стека

int size; // Текущий размер стека

};

void Push(Stack& S, char x)

{

if (S.size == MAXSIZE) {

printf("Стек переполнен");

return;

}

S.data[S.size] = x; // Добавление элемента в стек

S.size++; // Увеличение размера стека

}

char Pop(Stack& S)

{

if (S.size == 0) {

printf("Стек пуст");

return char(255); // Возвращение значения 255

}

S.size--; // Уменьшение размера стека

return S.data[S.size]; // Возвращение удаленного элемента

}

void main()

{

setlocale(LC\_ALL, "Rus");

char br1[3] = { '(', '[', '{' }; // открывающие скобки

char br2[3] = { ')', ']', '}' }; // закрывающие скобки

char s[80], upper;

int i, k, OK;

Stack S; //стек символов

printf("Введите выражение со скобками> ");

fgets(s, MAXSIZE, stdin); // Здесь была ошибка, так как gets устарела, то используем fgets и передаём ей 3 параметра

S.size = 0; //сначала стек пуст

OK = 1;

for (i = 0; OK && (s[i] != '\0'); i++)

for (k = 0; k < 3; k++) { //проверить 3 вида скобок

if (s[i] == br1[k]) { // открывающая скобка

Push(S, s[i]);

break;

}

if (s[i] == br2[k]) { // закрывающая скобка

upper = Pop(S);

if (upper != br1[k])

OK = 0;

break;

}

}

if (OK && (S.size == 0))

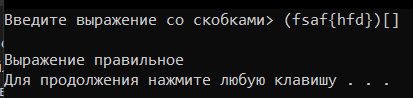
printf("\nВыpажение пpавильное\n");

else

printf("\nВыpажение непpавильное \n");

system("PAUSE");

}



# Тестирование дека

#include<iostream>

using namespace std;

struct node

{

int elem;

node\* sled;

};

class Spisok

{

private:

node\* ld, \* rd; // указатели на левое и правое звено

int el\_left, el\_right; // элементы в левом и правом звене

public:

void POSTROENIE(); // Функция для построения дека

void VYVOD(); // Функция для вывода содержимого дека

void VSTAV1(int); // Функция для вставки звена справа

void VSTAV2(int); // Функция для вставки звена слева

int SetElLeft() { return el\_left; } // Функция для получения элемента в левом звене

int SetElRight() { return el\_right; } // Функция для получения элемента в правом звене

void YDALE1(); // Функция для удаления звена справа

void YDALE2(); // Функция для удаления звена слева

void OCHISTKA(); // Функция для очистки дека

};

void main()

{

setlocale(LC\_ALL, "Rus");

Spisok A;

int el;

A.POSTROENIE(); A.VYVOD(); // Вызов Функции для построения дека и Функции для его вывода

cout << "Добавим звено справа.\n";

cout << "Введите элемент добавляемого звена: ";

cin >> el;

A.VSTAV1(el); A.VYVOD(); // Вызов Функции для вставки звена справа и Функции для вывода дека

cout << "Добавим звено слева.\n";

cout << "Введите элемент добавляемого звена: ";

cin >> el;

A.VSTAV2(el); A.VYVOD(); // Вызов Функции для вставки звена слева и Функции для вывода дека

cout << "Удалим звено справа.\n";

A.YDALE1(); A.VYVOD(); cout << A.SetElRight() << endl; // Вызов Функции для удаления звена справа, Функция для вывода дека и вывод элемента в правом звене

cout << "Удалим звено слева.\n";

A.YDALE2(); A.VYVOD(); cout << A.SetElLeft() << endl; // Вызов Функции для удаления звена слева, Функция для вывода дека и вывод элемента в левом звене

A.OCHISTKA(); // Вызов Функции для очистки дека

cout << "\n";

system("PAUSE");

}

void Spisok::POSTROENIE()

{

node\* k;

int el;

cout << "Вводите содержимое звеньев дека: \n";

cin >> el;

if (el != 0)

{

k = new (node); // Выделение памяти для нового звена дека

(\*k).elem = el; (\*k).sled = NULL; // Запись значения элемента и установка указателя на следующее звено

ld = k; rd = k; cin >> el; // Инициализация указателей ld и rd, чтение следующего значения элемента

while (el != 0)

{

VSTAV1(el); cin >> el; // Вставка нового звена справа и чтение следующего значения элемента

}

}

else

{

rd = NULL; ld = NULL; // Если введенное значение = 0, то дек пустой, поэтому обнуляем указатели ld и rd

}

}

void Spisok::VYVOD()

{

node\* k;

k = ld; cout << "Дек: ";

while (k != NULL)

{

cout << (\*k).elem << " "; k = (\*k).sled; // Вывод значения элемента и переход к следующему звену

}

cout << endl;

}

void Spisok::VSTAV1(int el)

{

node\* k;

k = new (node); // Выделение памяти для нового звена дека

(\*k).elem = el; (\*k).sled = NULL; // Запись значения элемента и установка указателя на следующее звено

if (rd != NULL)

{

(\*rd).sled = k; rd = k; // Если rd != NULL, то устанавливаем указатель следующего звена текущего последнего звена на новое звено и обновляем указатель rd

}

else

{

rd = k; ld = k; // Если rd == NULL, то обновляем указатели ld и rd

}

}

void Spisok::VSTAV2(int el)

{

node\* k;

k = new (node); // Выделение памяти для нового звена дека

(\*k).elem = el; (\*k).sled = ld; // Запись значения элемента и установка указателя на следующее звено равным текущему левому звену

if (ld != NULL) ld = k; // Если ld != NULL, то обновляем указатель ld

else { ld = k; rd = k; } // Если ld == NULL, то обновляем указатели ld и rd

}

void Spisok::YDALE1()

{

node\* z;

node\* k;

if (rd == ld) // Если rd и ld указывают на одно и то же звено

{

el\_right = (\*rd).elem; delete rd; // Сохраняем значение элемента в правом звене, удаляем звено

ld = rd = NULL; cout << "Дек пуст!\n"; // Обнуляем указатели ld и rd

}

else

{

z = ld; k = (\*ld).sled; // Инициализация z и k

while (k != rd)

{

z = k; k = (\*k).sled; // Переход к следующему звену

}

el\_right = (\*rd).elem; (\*z).sled = NULL; delete rd; // Сохраняем значение элемента в правом звене, обновляем указатель на следующее звено у предыдущего звена, удаляем последнее звено

rd = z; // Обновляем указатель rd

}

}

void Spisok::YDALE2()

{

node\* q;

if (ld != NULL) // Если ld != NULL

{

el\_left = (\*ld).elem; q = ld; // Сохраняем значение элемента в левом звене, инициализируем временный указатель q

ld = (\*ld).sled; delete q; // Обновляем указатель ld, удаляем первое звено

}

else cout << "Дек пуст!\n"; // Если ld == NULL

}

void Spisok::OCHISTKA()

{

node\* k, \* q;

k = ld; // Инициализация k

if (k != NULL) // Если k != NULL

{

q = (\*k).sled; // Инициализация временного указателя q на следующее звено

while (q != NULL) // Пока есть следующее звено

{

delete k; k = q; q = (\*k).sled; // Удаляем текущее звено, переходим к следующему звену

}

delete k; // Удаляем последнее звено

}

}

