МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Программной инженерии

Специальность Программная инженерия

**Тестирование**

По предмету

Основы алгоритмизации и программирования

Выполнил:

Студент 1 курса 10 группы, 2 подгруппы

Специальность ПИ Мандрик Алексей Иванович

Проверил:

Белодед Николай Иванович

Минск 2024

**Тестирование №1**

**Код, который необходимо протестировать:**

#include<iostream>

using namespace std;

// Структура для звена висюльки

struct nodeVis

{

int elem; // Информационное поле звена висюльки.

nodeVis\* vniz; // Указатель на следующее звено висюльки.

};

// Структура для звена гирлянды

struct nodeGir

{

int elem; // Информационное поле звена гирлянды.

nodeVis\* vniz; // Указатель на висюльку, привязанную к этому звену гирлянды.

nodeGir\* sled; // Указатель на следующее звено гирлянды.

};

// Класс, представляющий гирлянду с висюльками

class GirVis {

private:

nodeGir\* phead; // Голова гирлянды.

nodeVis\* pheadVis; // Голова висюльки.

void VisVyvod(); // Вспомогательная функция для вывода висюльки.

public:

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

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

nodeVis\* VisPostr(); // Построение висюльки.

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

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

void VisVstav(nodeVis\*, int); // Вставка элемента в висюльку.

void Vis1Vstav(nodeVis\*, int); // Вставка элемента перед указанным в висюльку.

void VisUdale(nodeVis\*); // Удаление следующего звена в висюльке.

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

void GirPostr(); // Построение гирлянды.

void GirVyvod(); // Вывод содержимого гирлянды.

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

void OCHISTKA(); // Очистка гирлянды.

void OCHISTKA1(); // Очистка висюльки.

};

void main()

{

setlocale(LC\_ALL, "Rus");

GirVis A;

int el, elGir, elVis;

nodeGir\* Res; //Рабочий указатель.

nodeVis\* ResVis; //Указатель на звено висюльки.

A.GirPostr();

A.GirVyvod();

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

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

cin >> elGir;

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

cout << "осуществляется вставка:\n";

cin >> elVis;

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

cin >> el;

//Поиск элемента elGir в гирлянде.

Res = A.GirPoisk(elGir);

if (Res != NULL)

{

//Поиск элемента elVis в висюльке.

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

ResVis = A.VisPoisk(elVis);

if (ResVis != NULL)

A.VisVstav(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, перед которым ";

cout << "осуществляется вставка:\n";

cin >> elVis;

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

cin >> el;

//Поиск элемента elGir в гирлянде.

Res = A.GirPoisk(elGir);

if (Res != NULL)

{

//Поиск элемента elVis в висюльке.

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 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);

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;

//Поиск элемента elGir в гирлянде.

Res = A.GirPoisk(elGir);

if (Res != NULL)

{

//Поиск элемента elVis в висюльке.

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();

cout << "\n";

system("PAUSE");

}

// Очистка гирлянды.

void GirVis::OCHISTKA()

{

nodeGir\* q, \* q1;//Рабочие указатели.

q = phead;

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

while (q1 != NULL)

{

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

pheadVis = (\*q).vniz;

OCHISTKA1(); //Очистка висюльки.

delete q;

}

}

// Очистка висюльки.

void GirVis::OCHISTKA1()

{

nodeVis\* q, \* q1;

q = pheadVis;

q1 = (\*q).vniz;

while (q1 != NULL)

{

q = q1; q1 = (\*q1).vniz;

delete q;

}

}

// Построение гирлянды.

void GirVis::GirPostr()

//Построение однонаправленного списка с заглавным звеном,

//заданного указателем phead (построение гирлянды).

{

nodeGir\* t;

int el;

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

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

cin >> el;

while (el != 0)

{

(\*t).sled = new (nodeGir);

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

(\*t).vniz = VisPostr();

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

cin >> el;

}

}

// Построение висюльки.

nodeVis\* GirVis::VisPostr()

//Построение однонаправленного списка с заглавным звеном

//(построение висюльки). pheadVis - указатель на висюльку.

{

nodeVis\* t;

int el;

//Создадим заглавное звено списка.

pheadVis = new (nodeVis);

t = pheadVis; (\*t).vniz = NULL;

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

cin >> el;

while (el != 0)

{

(\*t).vniz = new (nodeVis);

t = (\*t).vniz; (\*t).elem = el; (\*t).vniz = NULL;

cin >> el;

}

return pheadVis;

}

// Вывод содержимого гирлянды.

void GirVis::GirVyvod()

//Вывод содержимого однонаправленного списка, заданного

//указателем phead (вывод содержимого гирлянды).

{

nodeGir\* t;

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

cout << "Гирлянда: ";

while (t != NULL)

{

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

pheadVis = (\*t).vniz;

VisVyvod();

t = (\*t).sled;

}

}

// Поиск элемента в гирлянде.

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; t = (\*t).sled;

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

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

else t = (\*t).sled;

return r;

}

// Вспомогательная функция для вывода висюльки.

void GirVis::VisVyvod()

//Вывод содержимого однонаправленного списка с заглавным звеном,

//заданного указателем pheadVis (вывод содержимого висюльки).

{

nodeVis\* t;

t = pheadVis; t = (\*t).vniz;

cout << "(";

while (t != NULL)

{

cout << (\*t).elem << " "; t = (\*t).vniz;

}

cout << ")";

}

// Поиск элемента в висюльке.

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; t = (\*t).vniz;

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

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

else t = (\*t).vniz;

return r;

}

// Вставка элемента в висюльку.

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);

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

}

// Вставка элемента перед указанным в висюльку.

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;

q = new (nodeVis);

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

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

}

// Удаление следующего звена в висюльке.

void GirVis::VisUdale(nodeVis\* r)

//Удаление звена, расположенного после звена,

//на которое указывает ссылка r

//(удаление звена висюльки).

{

nodeVis\* q;

q = (\*r).vniz;

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

{

(\*r).vniz = (\*(\*r).vniz).vniz; delete q;

}

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

}

// Удаление указанного звена в висюльке.

void GirVis::Vis1Udale(nodeVis\* r)

//Удаление звена, на которое указывает ссылка r

//(удаление звена висюльки).

{

nodeVis\* g;

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

{

g = (\*r).vniz;

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

(\*r).vniz = (\*(\*r).vniz).vniz;

delete g;

}

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

}

**Вывод**

****

**Тестирование №2**

**Код, который необходимо протестировать:**

#include<iostream>

using namespace std;

// Структура для звена списка

struct node

{

int elem; // Информационное поле звена списка

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

};

// Класс, представляющий кольцевой список

class Spisok {

private:

node\* phead, \* Res; // Указатели на заглавное звено и рабочий звено

public:

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

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

void POSTROENIE(); // Построение кольцевого списка

void VYVOD(); // Вывод содержимого кольцевого списка

node\* POISK(int); // Поиск элемента в списке

void InsAfter(int); // Вставка элемента после указанного

void InsBefore(int); // Вставка элемента перед указанным

void Delete(); // Удаление указанного звена

void DelAfter(); // Удаление звена после указанного

void OCHISTKA(); // Очистка списка

};

// Основная функция main

void main()

{

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

Spisok A; // Создание объекта класса Spisok

int el, el1; // Переменные для ввода данных

node\* Res\_Zn; // Рабочий указатель на звено

// Построение списка и вывод содержимого

A.POSTROENIE();

A.VYVOD();

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

cout << "осуществляется вставка: ";

cin >> el;

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

cin >> el1;

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

{

A.InsAfter(el1); A.VYVOD();

}

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();

}

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();

}

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();

}

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

A.OCHISTKA();

cout << "\n";

system("PAUSE");

}

// Функция для построения кольцевого списка

void Spisok::POSTROENIE()

{

node\* t; // Рабочий указатель

int el; // Переменная для ввода данных

t = phead; (\*t).sled = NULL; // Инициализация заглавного звена

cout << "Вводите элементы кольца: ";

cin >> el;

// Ввод элементов в кольцо

while (el != 0)

{

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

t = (\*t).sled; (\*t).elem = el;

cin >> el;

}

(\*t).sled = (\*phead).sled; // Замыкаем список

}

// Функция для вывода содержимого кольцевого списка

void Spisok::VYVOD()

{

node\* t; // Рабочий указатель

t = (\*phead).sled;

cout << "Кольцо: ";

// Вывод элементов списка

if (t != NULL)

{

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

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

{

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

}

}

else cout << "пусто!\n";

}

// Функция для поиска элемента в кольцевом списке

node\* Spisok::POISK(int el)

{

node\* t; // Рабочий указатель

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

// Поиск элемента

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;

}

// Функция для вставки элемента после указанного

void Spisok::InsAfter(int el)

{

node\* q; // Новое звено

q = new (node);

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

(\*Res).sled = q;

}

// Функция для вставки элемента перед указанным

void Spisok::InsBefore(int el)

{

node\* q; // Новое звено

q = new (node);

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

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

}

// Функция для удаления указанного звена

void Spisok::Delete()

{

node\* z, \* q; // Рабочие указатели

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

{

q = (\*Res).sled;

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

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

delete q;

}

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

{

// В кольце единственное звено

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

delete q; cout << "Кольцо пусто!";

}

else

{

// Удаляется "последнее" звено кольца

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

while (q != Res)

{

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

}

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

delete q;

}

}

// Функция для удаления звена после указанного

void Spisok::DelAfter()

{

node\* q; // Рабочий указатель

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

{

// Ссылка Res не указывает на последнее звено

q = (\*Res).sled;

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

delete q;

}

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

{

// Удаляемое звено - единственное в кольце

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

delete q; cout << "Кольцо пусто!";

}

else

{

// Удаляемое звено - первое в кольце и не единственное

q = (\*phead).sled;

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

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

}

}

// Функция для очистки списка

void Spisok::OCHISTKA()

{

node\* q, \* q1; // Рабочие указатели

q = phead;

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

// Удаление всех звеньев списка

do {

q = q1;

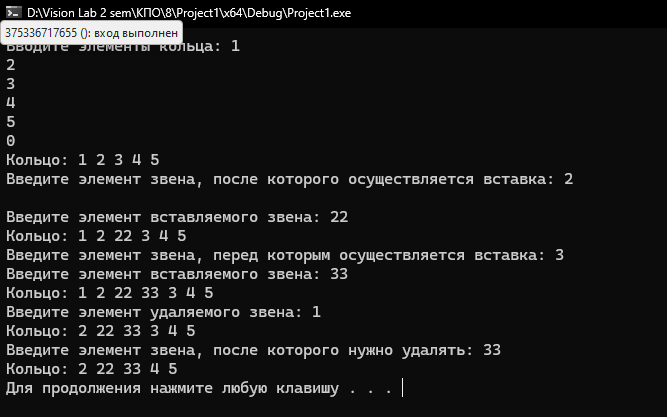
q1 = (\*q1).sled;

delete q;

} while (q1 != (\*phead).sled);

}

**Вывод**

****

**Тестирование №3**

**Код, который необходимо протестировать:**

#include <iostream>

using namespace std;

// Структура для звена списка

struct node

{

int elem; // Информационное поле звена

node\* sled; // Указатель на следующее звено

node\* pred; // Указатель на предыдущее звено

};

// Класс, представляющий кольцевой двунаправленный список

class Spisok

{

private:

node\* nsp; // Указатель на заглавное звено списка

public:

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

void BuiltRing(); // Построение списка

void VyvodLeftRight(); // Вывод списка по часовой стрелке

void VyvodRightLeft(); // Вывод списка против часовой стрелки

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

void InsBefore(node\*, int); // Вставка элемента перед указанным

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

void DelAfter(node\*); // Удаление звена после указанного

node\* SearchRing(int); // Поиск элемента в списке

void Ochistka(); // Очистка списка

};

// Основная функция main

void main()

{

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

Spisok A; // Создание объекта класса Spisok

node\* Res; // Рабочий указатель на звено

int el, el1; // Переменные для ввода данных

A.BuiltRing(); // Построение списка

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

A.VyvodLeftRight(); // Вывод списка по часовой стрелке

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

A.VyvodRightLeft(); // Вывод списка против часовой стрелки

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

cin >> el;

cout << "Введите элементы вставляемого звена:";

cin >> el1;

Res = A.SearchRing(el);

if (Res != NULL)

{

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

A.VyvodLeftRight();

}

else

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

cout << "Введите элементы 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 перед указанным звеном

A.VyvodLeftRight();

}

else

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

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

cin >> el;

Res = A.SearchRing(el);

if (Res != NULL)

{

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

A.VyvodLeftRight();

}

else

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

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

cin >> el;

Res = A.SearchRing(el);

if (Res != NULL)

{

A.DelAfter(Res); // Удаление звена после указанного

A.VyvodLeftRight();

}

else

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

A.Ochistka(); // Очистка списка

cout << "\n";

system("PAUSE");

}

// Функция для построения кольцевого двунаправленного списка

void Spisok::BuiltRing()

{

node\* r; // Рабочий указатель

int el; // Переменная для ввода данных

// Построение заглавного звена списка

nsp = new(node);

r = nsp;

(\*nsp).pred = NULL;

(\*nsp).sled = NULL;

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

cin >> el;

// Ввод элементов в список

while (el != 0)

{

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

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

r = (\*r).sled;

(\*r).sled = NULL;

(\*r).elem = el;

cin >> el;

}

// Образование кольцевого списка

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

{

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

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

}

else

cout << "Конец списка пуст!\n";

}

// Функция для вывода списка по часовой стрелке

void Spisok::VyvodLeftRight()

{

node\* r; // Рабочий указатель

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 против часовой стрелки

void Spisok::VyvodRightLeft()

{

node\* r; // Рабочий указатель

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;

// Проверка первого звена

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

Res = (\*p).sled;

else

{

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

// Поиск элемента в остальных звеньях

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

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

Res = q;

else

q = (\*q).sled;

}

return Res;

}

// Функция для вставки элемента после указанного звена

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 звеном

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 звена

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 звена 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;

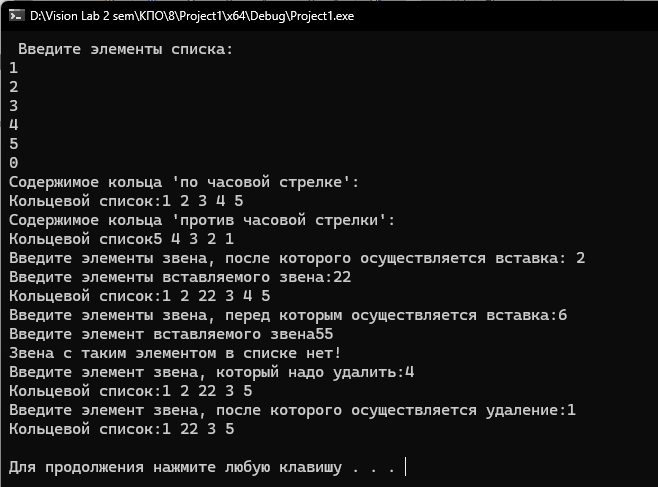
}

delete q;

delete nsp;

}

**Вывод**

****

**Тестирование №4**

**Код, который необходимо протестировать:**

#include<iostream> // Подключение библиотеки для ввода/вывода

using namespace std;

struct node // Определение структуры узла списка

{

int elem; // Элемент списка

node\* sled; // Указатель на следующий узел

};

class Spisok // Определение класса "Список"

{

private:

node\* stk; // Указатель на вершину стека

int klad; // Параметр для хранения значения удаленного элемента

public:

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

int Set\_Stack() { return klad; } // Метод для получения значения klad

void POSTROENIE(); // Метод для построения стека

void VYVOD(); // Метод для вывода содержимого стека

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

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

void OCHISTKA(); // Метод для освобождения памяти

};

void main()

{

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

Spisok A; // Создание объекта класса "Список"

int el; // Переменная для хранения вводимого элемента

int t; // Переменная для хранения значения вершины стека

A.POSTROENIE(); A.VYVOD(); // Вызов метода POSTROENIE для построения стека и метода VYVOD для вывода содержимого стека

cout << "Введите вставляемый элемент: "; // Вывод запроса на ввод элемента

cin >> el; // Ввод элемента

A.W\_S(el); A.VYVOD(); // Добавление элемента в стек и вывод содержимого стека

cout << "Удалим элемент из стека.\n"; // Вывод сообщения о удалении элемента из стека

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

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

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

A.VYVOD(); // Вывод содержимого стека

A.OCHISTKA(); // Освобождение памяти

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

system("PAUSE"); // Приостановка выполнения программы

}

void Spisok::POSTROENIE() // Метод для построения стека

// Построение стека, заданного указателем stk.

{

node\* t; // Временный указатель на узел

int el; // Временная переменная для хранения вводимого элемента

cout << "Вводите элементы стека: "; // Вывод запроса на ввод элементов

cin >> el; // Ввод первого элемента

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

{

t = new (node); // Выделение памяти под новый узел

(\*t).elem = el; (\*t).sled = stk; // Заполнение полей узла и указание на следующий узел

stk = t; cin >> el; // Обновление указателя на вершину стека и ввод следующего элемента

}

}

void Spisok::VYVOD() // Метод для вывода содержимого стека

// Вывод содержимого стека, заданного указателем stk.

{

node\* t; // Временный указатель на узел

cout << "Содержимое стека: "; t = stk; // Вывод сообщения о содержимом стека и инициализация временного указателя

while (t != NULL) // Пока не достигнут конец стека

{

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

} // Вывод элемента и переход к следующему узлу

cout << endl; // Переход на новую строку

}

void Spisok::W\_S(int el) // Метод для добавления элемента в стек

// Помещение элемента el в стек stk.

{

node\* q; // Временный указатель на узел

q = new (node); // Выделение памяти под новый узел

(\*q).elem = el; (\*q).sled = stk; stk = q; // Заполнение полей узла и обновление указателя на вершину стека

}

void Spisok::YDALENIE() // Метод для удаления элемента из стека

// Удаление элемента из стека, заданного указателем stk.

// Значение информационного поля удаляемого элемента

// помещается в параметр klad.

{

node\* q; // Временный указатель на узел

if (stk == NULL) // Если стек пуст

cout << "Стек пуст!\n"; // Вывод сообщения о пустом стеке

else

{

klad = (\*stk).elem; q = stk; // Сохранение значения вершины стека и временное сохранение указателя на вершину

stk = (\*stk).sled; delete q; // Обновление указателя на вершину стека и удаление узла

}

}

void Spisok::OCHISTKA()// Метод для освобождения памяти

// Возврат выделенной памяти в "кучу".

{

node\* t, \* q; // Временные указатели на узлы

t = stk; // Инициализация временного указателя

if (t != NULL) // Если стек не пуст

{

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

while (q != NULL) // Пока не достигнут конец списка

{

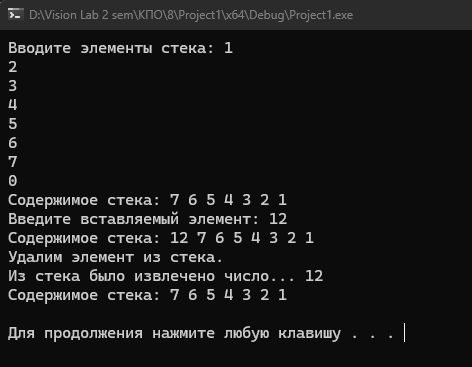
delete t; t = q; q = (\*q).sled; // Удаление узла, переход к следующему узлу

}

}

}

**Вывод**



**Тестирование №5**

**Код, который необходимо протестировать:**

#include <iostream> // Подключение библиотеки для ввода и вывода через потоки

#include <cstring> // Подключение библиотеки для работы с строками

using namespace std; // Использование пространства имен std

const int MAXSIZE = 100; // Определение максимального размера стека

struct Stack // Определение структуры стека

{

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

int size; // Переменная для хранения текущего размера стека

};

void Push(Stack& S, char x) // Функция для добавления элемента в стек

{

if (S.size == MAXSIZE) { // Если стек полон

printf("Стек переполнен"); // Вывод сообщения о переполнении

return; // Выход из функции

}

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

S.size++; // Увеличение размера стека

}

char Pop(Stack& S) // Функция для удаления элемента из стека

{

if (S.size == 0) { // Если стек пуст

printf("Стек пуст"); // Вывод сообщения о пустом стеке

return char(255); // Возврат символа, обозначающего пустой стек

}

S.size--; // Уменьшение размера стека

return S.data[S.size]; // Возврат удаленного элемента стека

}

int main() // Основная функция программы

{

setlocale(LC\_ALL, "Rus"); // Установка локали для корректного отображения русских символов

char br1[3] = { '(', '[', '{' }; // Массив открывающих скобок

char br2[3] = { ')', ']', '}' }; // Массив закрывающих скобок

char s[80], upper; // Массив для ввода выражения и переменная для хранения верхней скобки из стека

int i, k, OK; // Переменные для циклов и проверки корректности скобок

Stack S; // Объявление стека

printf("Введите выражение со скобками> "); // Вывод приглашения на ввод выражения

cin.getline(s, 80); // Считывание выражения

S.size = 0; // Инициализация размера стека

OK = 1; // Инициализация переменной для проверки корректности скобок

for (i = 0; OK && (s[i] != '\0'); i++) // Цикл по символам выражения

for (k = 0; k < 3; k++) { // Цикл для проверки каждого вида скобок

if (s[i] == br1[k]) { // Если текущий символ - открывающая скобка

Push(S, s[i]); // Добавить ее в стек

break; // Выход из цикла

}

if (s[i] == br2[k]) { // Если текущий символ - закрывающая скобка

upper = Pop(S); // Удалить верхнюю скобку из стека

if (upper != br1[k]) // Если закрывающая скобка не соответствует открывающей

OK = 0; // Установить флаг некорректности

break; // Выход из цикла

}

}

if (OK && (S.size == 0)) // Если скобки расставлены корректно и стек пуст

printf("\nВыражение правильное\n"); // Вывод сообщения о корректности выражения

else

printf("\nВыражение неправильное\n"); // Вывод сообщения о некорректности выражения

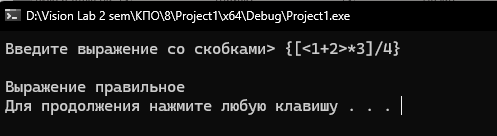
system("PAUSE"); // Ожидание нажатия клавиши перед завершением программы

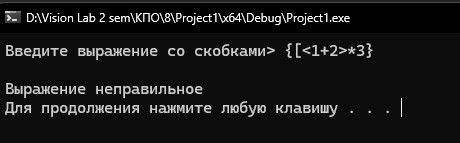
return 0; // Возврат нулевого значения, указывающего на успешное завершение программы

}

Пришлось заменить конструкцию gets, т.к. она устарела.

**Вывод**





**Тестирование №6**

**Код, который необходимо протестировать:**

#include<iostream> // Подключение библиотеки для ввода и вывода через потоки

using namespace std; // Использование пространства имен std

struct node // Определение структуры узла

{

int elem; // Элемент узла

node\* sled; // Указатель на следующий узел

};

class Spisok // Определение класса "Список"

{

private:

node\* ld, \* rd; // Указатели на левый и правый концы списка

int el\_left, el\_right; // Переменные для хранения удаляемых элементов

public:

void POSTROENIE(); // Метод для построения списка

void VYVOD(); // Метод для вывода списка

void VSTAV1(int); // Метод для вставки элемента справа

void VSTAV2(int); // Метод для вставки элемента слева

int SetElLeft() { return el\_left; } // Метод для получения левого элемента

int SetElRight() { return el\_right; } // Метод для получения правого элемента

void YDALE1(); // Метод для удаления элемента справа

void YDALE2(); // Метод для удаления элемента слева

void OCHISTKA(); // Метод для очистки памяти

};

void main()

{

setlocale(LC\_ALL, "Rus"); // Установка локали для корректного отображения русских символов

Spisok A; // Создание объекта класса Spisok

int el; // Переменная для ввода элементов

A.POSTROENIE(); A.VYVOD(); // Вызов методов построения и вывода списка

cout << "Добавим звено справа.\n"; // Вывод сообщения о добавлении элемента справа

cout << "Введите элемент добавляемого звена: "; // Приглашение на ввод элемента

cin >> el; // Ввод элемента

A.VSTAV1(el); A.VYVOD(); // Вызов метода вставки элемента справа и вывод списка

cout << "Добавим звено слева.\n"; // Вывод сообщения о добавлении элемента слева

cout << "Введите элемент добавляемого звена: "; // Приглашение на ввод элемента

cin >> el; // Ввод элемента

A.VSTAV2(el); A.VYVOD(); // Вызов метода вставки элемента слева и вывод списка

cout << "Удалим звено справа.\n"; // Вывод сообщения об удалении элемента справа

A.YDALE1(); A.VYVOD(); cout << A.SetElRight() << endl; // Вызов метода удаления элемента справа, вывод списка и удаленного элемента

cout << "Удалим зввено слева.\n"; // Вывод сообщения об удалении элемента слева

A.YDALE2(); A.VYVOD(); cout << A.SetElLeft() << endl; // Вызов метода удаления элемента слева, вывод списка и удаленного элемента

A.OCHISTKA(); // Вызов метода очистки памяти

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

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

}

void Spisok::POSTROENIE()

// Построение списка:

// ld - указатель на левый конец дека,

// rd - Указатель на правый конец дека.

{

node\* k; // Указатель на текущий узел

int el; // Переменная для ввода элемента

cout << "Вводите содержимое звеньев дека: \n"; // Вывод приглашения на ввод элементов

cin >> el; // Ввод первого элемента

if (el != 0) // Если элемент не равен 0

{

k = new (node); // Выделение памяти под новый узел

(\*k).elem = el; (\*k).sled = NULL; // Присвоение значения элементу и установка указателя на следующий узел

ld = k; rd = k; cin >> el; // Установка указателей на первый узел и считывание следующего элемента

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

{

VSTAV1(el); cin >> el; // Вызов метода вставки элемента справа и считывание следующего элемента

}

}

else

{

rd = NULL; ld = NULL; // Если элемент равен 0, обнуляем указатели

}

}

void Spisok::VYVOD()

// Вывод содержимого списка:

// ld - указатель на левый конец дека.

{

node\* k; // Указатель на текущий узел

k = ld; cout << "Дек: "; // Установка указателя на левый конец и вывод сообщения о списке

while (k != NULL) // Пока указатель не станет NULL

{

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

}

cout << endl; // Переход на новую строку

}

void Spisok::VSTAV1(int el)

// Помещение узла, содержащего элемент el, в дек справа.

// ld - указатель на левый конец дека,

// rd - указатель на правый конец дека.

{

node\* k; // Указатель на новый узел

k = new (node); // Выделение памяти под новый узел

(\*k).elem = el; (\*k).sled = NULL; // Присвоение значения элементу и установка указателя на следующий узел

if (rd != NULL) // Если правый конец не пустой

{

(\*rd).sled = k; rd = k; // Присвоение указателю на следующий узел адрес нового узла и обновление правого конца

}

else

{

rd = k; ld = k; // Иначе, установка указателей на новый узел

}

}

void Spisok::VSTAV2(int el)

// Помещение узла, содержащего элемент el, в дек слева.

// ld - указатель на левый конец дека,

// rd - указатель на правый конец дека.

{

node\* k; // Указатель на новый узел

k = new (node); // Выделение памяти под новый узел

(\*k).elem = el; (\*k).sled = ld; // Присвоение значения элементу и установка указателя на следующий узел

if (ld != NULL) ld = k; // Если левый конец не пустой, установка указателя на новый узел

else { ld = k; rd = k; } // Иначе, установка указателей на новый узел

}

void Spisok::YDALE1()

// Удаление узла из дека справа

// с сохранением удаляемого узла в переменной el\_right.

// ld - указатель на левый конец дека,

// rd - указатель на правый конец дека.

{

node\* z; // Вспомогательный указатель

node\* k; // Указатель на текущий узел

if (rd == ld) // Если правый и левый концы совпадают

{

el\_right = (\*rd).elem; delete rd; // Сохраняем элемент в переменную, удаляем его и обнуляем указатели

ld = rd = NULL; cout << "Дек пуст!\n"; // Выводим сообщение о пустом списке

}

else

{

z = ld; k = (\*ld).sled; // Устанавливаем указатель на предыдущий узел и на следующий после левого конца

while (k != rd) // Пока не дойдем до правого конца

{

z = k; k = (\*k).sled; // Перемещаемся по узлам

}

el\_right = (\*rd).elem; (\*z).sled = NULL; delete rd; // Сохраняем элемент в переменную, удаляем его и обнуляем указатель на следующий узел

rd = z; // Обновляем правый конец

}

}

void Spisok::YDALE2()

// Удаление узла из дека слева

// с сохранением удаляемого узла в переменной el\_left.

// ld - указатель на левый конец дека,

// rd - указатель на правый конец дека.

{

node\* q; // Вспомогательный указатель

if (ld != NULL) // Если левый конец не пустой

{

el\_left = (\*ld).elem; q = ld; // Сохраняем элемент в переменную, устанавливаем указатель на удаляемый узел

ld = (\*ld).sled; delete q; // Обновляем левый конец и удаляем узел

}

else cout << "Дек пуст!\n"; // Выводим сообщение о пустом списке

}

void Spisok::OCHISTKA()

{

node\* k, \* q; // Указатели на текущий и следующий узлы

k = ld; // Устанавливаем указатель на левый конец

if (k != NULL) // Если указатель не пустой

{

q = (\*k).sled; // Устанавливаем указатель на следующий узел

while (q != NULL) // Пока следующий узел не пустой

{

delete k; k = q; q = (\*k).sled; // Удаляем текущий узел, переходим к следующему

}

delete k; // Удаляем последний узел

}

}

**Вывод**

