МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет: Информационных технологий

Кафедра: Программной инженерии

Тестирование кодов

По дисциплине” Основы Алгоритмизации и Программирования”

***Выполнил:*** *студент 1 курса 8 группы*

*Федорович В.Г*

**Специальность:** *ПИ 1 курс.*

**Проверил:** преподаватель *Белодед Николай Иванович*

Минск

2024

Содержание:

1. [Код Тестирования с комментариями(1)](#Ш1)…………………3
2. [Код Тестирования с комментариями(2)](#Ш2)…………………12
3. [Код Тестирования с комментариями(3)](#Ш3)…………………17
4. [Код Тестирования с комментариями(4)…………………23](#Ш4)
5. [Код Тестирования с комментариями(5)…………………26](#Ш5)
6. [Код Тестирования с комментариями(6)…………………28](#Ш6)

Код Тестирования 1

#include<iostream>

using namespace std;

struct nodeVis // Структура для звена "висячей" части гирлянды

{

int elem; // Инф. поле звена висюльки

nodeVis\* vniz; // Указатель на следующий элемент "висячей" части

};

// Структура для звена гирлянды

struct nodeGir

{

int elem; // Инф. поле звена гирлянды

nodeVis\* vniz; // Указатель на "висячую" часть гирлянды

nodeGir\* sled; // Указатель на следующий элемент гирлянды

};

class GirVis // Класс, представляющий гирлянду с висячими элементами

{

private:

nodeGir\* phead; // Голова гирлянды

nodeVis\* pheadVis; // Голова "висячей" части

void VisVyvod(); // Метод для вывода содержимого "висячей" части

public:

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

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

// Для работы с висячей частью

nodeVis\* VisPostr(); // Построение "висячей" части

nodeVis\* VisPoisk(int); // Поиск элемента в висячей части

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

void VisVstav(nodeVis\*, int); // Вставка элемента в висячую часть

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

void VisUdale(nodeVis\*); // Удаление элемента из висячей части

void Vis1Udale(nodeVis\*); // Удаление указанного элемента из висячей части

// Для работы с гирляндой

void GirPostr(); // Построение гирлянды

void GirVyvod(); // Вывод содержимого гирлянды

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

// Для очистки памяти

void OCHISTKA(); // Очистка гирлянды

void OCHISTKA1(); // Очистка "висячей" части

};

void main()

{

setlocale(LC\_ALL, "Rus"); // Установка локали для корректного отображения текста на русском языке.

GirVis A; // Создание объекта класса GirVis.

int el, elGir, elVis; // Объявление переменных для хранения вводимых значений.

nodeGir\* Res; // Объявление указателя на звено гирлянды (рабочий указатель).

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

A.GirPostr(); // Вызов метода для построения гирлянды.

A.GirVyvod(); // Вызов метода для вывода содержимого гирлянды.

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

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

cin >> elGir; // Ввод значения элемента гирлянды от пользователя.

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

cout << "осуществляется вставка:\n";

cin >> elVis; // Ввод значения элемента висюльки от пользователя.

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

cin >> el; // Ввод значения нового вставляемого элемента от пользователя.

//Поиск элемента elGir в гирлянде.

Res = A.GirPoisk(elGir); // Поиск элемента гирлянды с номером elGir.

if (Res != NULL) // Если элемент найден:

{

// Устанавливаем указатель на голову висячей части гирлянды, чтобы начать поиск в висячих элементах

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

ResVis = A.VisPoisk(elVis); // Поиск элемента в висячей части с номером elVis.

if (ResVis != NULL)// Если элемент в висячей части найден:

A.VisVstav(ResVis, el); // Вставляем новый элемент el после найденного элемента в висячей части

else

cout << "Элемента в висюльке нет!\n";

}

else

cout << "Элемента в гирлянде нет\n";

A.GirVyvod(); // Выводим содержимое гирлянды после выполнения операций.

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

cin >> elGir;

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

cout << "осуществляется вставка:\n";

cin >> elVis; // вводим номер элемента висюльки от пользователя

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

cin >> el; // вводим значение вставляемого элемента от пользователя

Res = A.GirPoisk(elGir);// Поиск элемента гирлянды с номером elGir.

// Если элемент найден в гирлянде:

if (Res != NULL)

{

// Устанавливаем указатель на голову висячей части гирлянды для поиска в "висячих" элементах.

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

// Поиск элемента в висячей части с номером elVis.

ResVis = A.VisPoisk(elVis);

// Если элемент найден в висячей части:

if (ResVis != NULL)

// Выполняем вставку нового элемента el перед найденным элементом в висячей части.

A.Vis1Vstav(ResVis, el);

else

cout << "Элемента в висюльке нет!\n";

}

else

cout << "Элемента в гирлянде нет!\n";

A.GirVyvod();// Выводим содержимое гирлянды после выполнения операций.

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

cin >> elGir;// вводимномер элемента гирлянды

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

cin >> elVis; // вводим номер элемента висюльки

Res = A.GirPoisk(elGir); // Поиск элемента гирлянды с номером elGir.

if (Res != NULL)// Если элемент найден в гирлянде:

{

A.SetpheadVis((\*Res).vniz);// Устанавливаем указатель на голову висячей части гирлянды для поиска в висячих элементах.

ResVis = A.VisPoisk(elVis);// Поиск элемента в висячей части с номером elVis.

// Если элемент найден в висячей части и у него есть следующий элемент:

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

// Удаляем следующий элемент после найденного в висячей части.

A.VisUdale(ResVis);

else

cout << "Элемента в висюльке нет!\n";

}

else

cout << "Элемента в гирлянде нет!\n";

// Выводим содержимое гирлянды после выполнения операций.

A.GirVyvod();

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

cin >> elGir; // номер элемента гирлянды от пользователя.

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

cin >> elVis;// номер элемента висюльки от пользователя.

// Поиск элемента гирлянды с номером elGir.

Res = A.GirPoisk(elGir);

// Если элемент найден в гирлянде:

if (Res != NULL)

{

// Устанавливаем указатель на голову висячей части гирлянды для поиска в висячих элементах.

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

// Поиск элемента в висячей части с номером elVis.

ResVis = A.VisPoisk(elVis);

// Если элемент найден в висячей части и у него есть следующий элемент:

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

// Удаляем найденный элемент в висячей части.

A.Vis1Udale(ResVis);

else

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

}

else

cout << "Элемента в гирлянде нет!\n";

// Выводим содержимое гирлянды после выполнения операций.

A.GirVyvod();

// Очищаем память, выделенную для объектов гирлянды и висюльки.

A.OCHISTKA();

cout << "\n"; // обзац

system("PAUSE"); // остановка выполнения программы до тех пор, пока пользователь не нажмет клавишу

}

void GirVis::OCHISTKA()

{

nodeGir\* q, \* q1; // Объявление рабочих указателей

q = phead; // Инициализация указателя q головой гирлянды

q1 = (\*q).sled; // Указатель q1 "опережает" указатель q, указывая на следующий элемент после головы

// Перебираем элементы гирлянды.

while (q1 != NULL)

{

q = q1; // Перемещаем указатель q на следующий элемент

q1 = (\*q1).sled; // Перемещаем указатель q1 на следующий элемент после текущего

pheadVis = (\*q).vniz; // Устанавливаем указатель на голову висячей части текущего элемента гирлянды

OCHISTKA1(); // Вызываем метод для очистки висюльки текущего элемента

delete q; // Освобождаем память текущего элемента гирлянды

}

}

void GirVis::OCHISTKA1()

{

nodeVis\* q, \* q1; // Объявление рабочих указателей

q = pheadVis; // Установка указателя q на голову висячей части

q1 = (\*q).vniz; // Установка указателя q1 на первый элемент висячей части

// Перебираем элементы висячей части

while (q1 != NULL)

{

q = q1; // Перемещаем указатель q на следующий элемент

q1 = (\*q1).vniz; // Перемещаем указатель на следующий элемент после текущего

delete q; // Освобождаем памятьтекущего эл висячей части

}

}

void GirVis::GirPostr()

// Построение однонаправленного списка с заглавным звеном,

// заданного указателем phead (построение гирлянды).

{

nodeGir\* t; // Объявление указателя

int el; // для вводимого пользователем значения

t = phead; // Инициализация указателя t указателем на голову гирлянды

(\*t).sled = NULL; // Устанавливаем следующий элемент после головы равным NULL, чтобы указать на конец списка

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

cin >> el; // Считываем первый элемент гирлянды от пользователя

// Пока пользователь не введет 0 (окончание ввода):

while (el != 0)

{

(\*t).sled = new (nodeGir); // Создаем новый элемент гирлянды после текущего элемента

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

(\*t).elem = el; // Устанавливаем значение элемента равным введенному пользователем значению

(\*t).sled = NULL; // Устанавливаем следующий элемент после текущего равным NULL

(\*t).vniz = VisPostr(); // Рекурсивно строим висячую часть для текущего элемента

cout << " Вводите элемент гирлянды: \n"; // Выводим приглашение для ввода следующего элемента гирлянды.

cin >> el; //ввод следующий элемент гирлянды

}

}

nodeVis\* GirVis::VisPostr()

// Построение однонаправленного списка с заглавным звеном

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

{

nodeVis\* t; // Объявление указателя на текущий элемент висюльки

int el; // Переменная для вводимого пользователем значения

// Создаем заглавное звено списка.

pheadVis = new (nodeVis);

t = pheadVis; // Устанавливаем указатель t на заглавное звено.

(\*t).vniz = NULL; // Устанавливаем ссылку на следующий элемент равной NULL, чтобы указать на конец списка.

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

cin >> el; // Считываем первый элемент висюльки от пользователя

// Пока пользователь не введет 0 (окончание ввода):

while (el != 0)

{

(\*t).vniz = new (nodeVis); // Создаем новый элемент висюльки после текущего элемента

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

(\*t).elem = el; // Устанавливаем значение элемента равным введенному пользователем значению

(\*t).vniz = NULL; // Устанавливаем ссылку на следующий элемент равной NULL

cin >> el; // вводим следующий элемент висюльки от пользователя

}

return pheadVis; // Возвращаем указатель на заглавное звено висюльки

}

void GirVis::GirVyvod()

// Вывод содержимого однонаправленного списка, заданного

// указателем phead (вывод содержимого гирлянды).

{

nodeGir\* t; // Объявление указателя на текущий элемент гирлянд

t = phead; // Устанавливаем указатель t на голову гирлянды

t = (\*t).sled; // Переходим к первому элементу гирлянды, пропуская заглавное звено

cout << "Гирлянда: ";

// Пока не достигнут конец списка:

while (t != NULL)

{

cout << (\*t).elem << " "; // Выводим значение элемента гирлянды

pheadVis = (\*t).vniz; // Устанавливаем указатель на висячую часть текущего элемента

VisVyvod(); // Вызываем метод для вывода содержимого висячей части

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

}

}

nodeGir\* GirVis::GirPoisk(int el)

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

// В случае успешного поиска возвращается адрес звена списка,

// содержащего элемент el. В противном случае - NULL.

{

nodeGir\* t, \* r; // Объявление указателей для перемещения по списку и для хранения результата поиска

r = NULL; // указатель r со значением NULL (результат поиска)

t = phead; // Устанавливаем указатель t на голову списка гирлянды.

t = (\*t).sled; // Переходим к первому элементу гирлянды, пропуская заглавное звено

// Пока не достигнут конец списка и результат поиска не установлен:

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

{

if ((\*t).elem == el) // Если значение текущего элемента равно искомому значению:

r = t; // Устанавливаем указатель r на текущий элемент списка

else

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

}

return r; // Возвращаем результат поиска (указатель на найденный элемент или NULL)

}

void GirVis::VisVyvod()

// Вывод содержимого однонаправленного списка с заглавным звеном,

// заданного указателем pheadVis (вывод содержимого висюльки).

{

nodeVis\* t; // Объявление указателя на текущий элемент висюлькии

t = pheadVis; // Устанавливаем указатель t на голову висюльки

t = (\*t).vniz; // Переходим к первому элементу висюльки, пропуская заглавное звено.

cout << "(";

// Пока не достигнут конец висюльки:

while (t != NULL)

{

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

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

}

cout << ")";

}

nodeVis\* GirVis::VisPoisk(int el)

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

// В случае успешного поиска возвращается адрес звена списка,

// содержащего элемент el. В противном случае - NULL.

{

nodeVis\* t, \* r; // Объявление указателей для перемещения по списку и для хранения результата поиска.

r = NULL; //указателя r значением NULL (рез поиска)

t = pheadVis; // Устанавливаем указатель t на голову списка висюльки

t = (\*t).vniz; // Переходим к первому элементу висюльки, пропуская заглавное звено

// Пока не достигнут конец списка и результат поиска не установлен:

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

{

if ((\*t).elem == el) // Если значение текущего элемента равно искомому значению:

r = t; // Устанавливаем указатель r на текущий элемент списка

else

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

}

return r; // Возвращаем результат поиска , то есть указатель на найденный элемент или NULL

}

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; // Объявление

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

(\*q).elem = el; // Присваивание информационному полю нового звена значения el

(\*q).vniz = (\*r).vniz; // Установка ссылки нового звена на следующий элемент после r

(\*r).vniz = q; // Установка ссылки r на новый элемент

// Теперь новый элемент q вставлен после элемента, на который указывает r

}

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; // Объявление указателя на новый элемент висюльки.

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

(\*q).elem = (\*r).elem; // Копирование значения информационного поля из элемента r в элемент q.

(\*q).vniz = (\*r).vniz; // Копирование ссылки на следующий элемент после r в элемент q.

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

(\*r).vniz = q; // Установка ссылки элемента r на новый элемент q.

// Теперь новый элемент q вставлен перед элементом r.

}

void GirVis::VisUdale(nodeVis\* r)

// Удаление звена, расположенного после звена,

// на которое указывает ссылка r

// (удаление звена висюльки).

{

nodeVis\* q; // Объявление указателя на удаляемый элемент висюльки

q = (\*r).vniz; // Присваивание указателю q ссылки на элемент после r

if ((\*r).vniz != NULL) // Проверка, не является ли элемент после r последним

{

(\*r).vniz = (\*(\*r).vniz).vniz; // Перенаправление ссылки на следующий элемент после r

delete q; // Удаление элемента, на который указывает q

}

else

{

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

}

}

void GirVis::Vis1Udale(nodeVis\* r)

// Удаление звена, на которое указывает ссылка r

// (удаление звена висюльки).

{

nodeVis\* g; // Объявление указателя на элемент, который будет удален

if ((\*r).vniz != NULL) // Проверка, существует ли элемент после r

{

g = (\*r).vniz; // Присваивание указателю g ссылки на элемент после r

(\*r).elem = (\*(\*r).vniz).elem; // Копирование значения информационного поля из элемента после r в элемент

(\*r).vniz = (\*(\*r).vniz).vniz; // Перенаправление ссылки на следующий элемент после r

delete g; // Удаление элемента, на который указывает

}

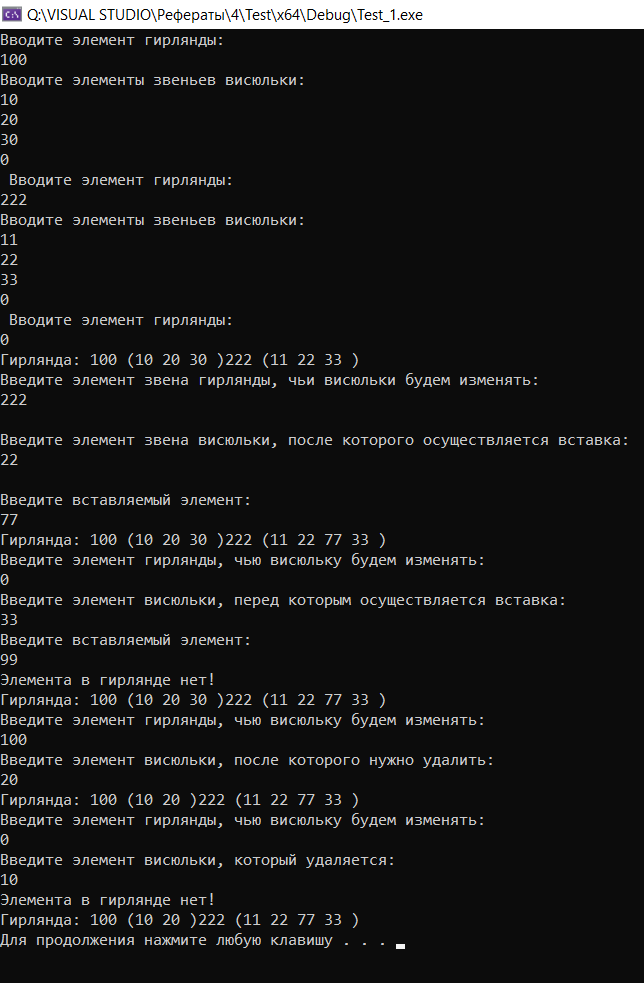
else

{

cout << "Не умею удалять последнее звено!\n"; // Вывод сообщения, если элемент после r не существует

}

}



Код Тестирования 2

#include <iostream>

using namespace std;

// Определение структуры node для хранения элементов списка

struct node {

int elem; // Хранит значение элемента списка

node\* sled; // Указатель на следующий элемент списка

};

// Определение класса Spisok для работы с односвязным списком

class Spisok {

private:

node\* phead; // Указатель на загл звено списка

node\* Res; // Рабочий указатель

public:

// Конструктор класса Spisok. Создает заглавное звено и инициализирует Res

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

// Деструктор класса Spisok. Освобождает память, выделенную для заглавного звена

~Spisok() { delete phead; }

// Методы класса Spisok для работы с односвязным списком

void POSTROENIE(); // Метод для построения списка

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

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

void InsAfter(int); // Метод для вставки элемента после указанного в списке

void InsBefore(int); // Метод для вставки элемента перед указанным в списке

void Delete(); // Метод для удаления указанного элемента из списка

void DelAfter(); // Метод для удаления элемента после указанного в списке

void OCHISTKA(); // Метод для очистки всего списка

};

void main() {

setlocale(LC\_ALL, "rus");

Spisok A; // Создаем объект класса Spisok

int el, el1; // Объявляем переменные для хранения вводимых пользователем значений

node\* Res\_Zn; // Объявляем указатель на узел для сохранения результата поиска

A.POSTROENIE(); // Вызываем метод POSTROENIE() для построения списка

A.VYVOD(); // Вызываем метод VYVOD() для вывода содержимого спискё

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

cout << "осуществляется вставка: ";

cin >> el;

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

cin >> el1;

if (A.POISK(el) != NULL) { // Проверяем, существует ли элемент в списке.

A.InsAfter(el1); // Если элемент существует, вставляем новый элемент после него.

A.VYVOD(); // После вставки выводим содержимое списка.

}

else {

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

}

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

cout << "осуществляется вставка: ";

cin >> el;

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

cin >> el1;

if (A.POISK(el) != NULL) { // Проверяем, существует ли элемент в списке

A.InsBefore(el1); // Если элемент существует, вставляем новый элемент перед ним

A.VYVOD(); // После вставки выводим содержимое списка

}

else {

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

}

cout << "\nВведите элемент удаляемого звена";

cin >> el;

if (A.POISK(el) != NULL) { // Проверяем, существует ли элемент в списке

A.Delete(); // Если элемент существует, удаляем его из списка

A.VYVOD(); // После удаления выводим содержимое списка

}

else {

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

}

cout << "Введите элемент звена ";

cout << "после которого нужно удалять: ";

cin >> el;

if (A.POISK(el) != NULL) { // Проверяем, существует ли элемент в списке

A.DelAfter(); // Если элемент существует, удаляем элемент после него

A.VYVOD(); // После удаления выводим содержимое списка

}

else {

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

}

A.OCHISTKA(); // Очищаем память, выделенную для списка

cout << "\n";

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

}

void Spisok::POSTROENIE()

// Функция для построения кольца односвязного списка

{

node\* t; // Временный указатель на тек звено списка

int el; // Переменная для эл списка

t = phead; // Установка временного указателя на начало списка (загл звено)

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

cout << "Вводите элементы кольца: ";

cin >> el; // Ввод первого элемента списка

while (el != 0) // Пока не введен нулевой элемент, продолжаем ввод

{

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

t = (\*t).sled; // Перемещение временного указателя на новое звено

(\*t).elem = el; // Присваивание введенного значения элементу списка

cin >> el; // Ввод следующего элемента списка

}

(\*t).sled = (\*phead).sled; // Связывание последнего звена с заглавным, создавая кольцо.

}

void Spisok::VYVOD()

// Метод для вывода содержимого кольца односвязного списка.

{

node\* t; // Временный указатель на текущее звено

t = (\*phead).sled; // Устанавливаем временный указатель на начало списка

cout << "Кольцо: ";

// Если список не пустой, выводим элементы кольца.

if (t != NULL)

{

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

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

// Пока не вернулись к началу кольца, выводим остальные элементы

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

{

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

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

}

}

else

{

cout << "пусто!\n";

}

}

node\* Spisok::POISK(int el)

// Метод для поиска элемента в кольце односвязного списка.

{

node\* t; // Временный указатель на текущее звено

Res = NULL; // Обнуляем указатель на найденное звено

t = (\*phead).sled; // Устанавливаем временный указатель на начало списка

// Пока не вернемся к началу кольца и не найдено звено

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

{

if ((\*t).elem == el) // Если текущий элемент равен искомому

{

Res = t; // Устанавливаем указатель на найденное звено

}

else

{

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

}

}

// Проверяем последнее звено на соответствие искомому элементу

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

{

Res = t; // Устанавливаем указатель на найденное звено

}

return Res; // Возвращаем указатель на найденное звено (или NULL)

}

void Spisok::InsAfter(int el)

// Метод для вставки элемента после указанного звена в кольце односвязного списка.

{

node\* q; // Новое звено списка

q = new(node); // Создаем новое звено

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

(\*q).sled = (\*Res).sled; // Устанавливаем связь с последующим звеном

(\*Res).sled = q; // Устанавливаем связь с новым звеном

}

void Spisok::InsBefore(int el)

// Метод для вставки элемента перед указанным звеном в кольце односвязного списка.

{

node\* q; // Новое звено списка

q = new(node); // Создаем новое звено

(\*q).elem = (\*Res).elem; // Копируем значение текущего звена в новое звено

(\*q).sled = (\*Res).sled; // Устанавливаем связь с последующим звеном

(\*Res).elem = el; // Присваиваем текущему звену новое значение

(\*Res).sled = q; // Устанавливаем связь с новым звеном

}

void Spisok::Delete()

// Метод для удаления указанного звена из кольца односвязного списка

{

node\* z, \* q; // Указатели на предыдущее и удаляемое звено

if ((\*Res).sled != (\*phead).sled) // Проверяем, что удаляемое звено не является последним в кольце

{

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

(\*Res).elem = (\*q).elem; // Копируем значение из следующего звена в текущее

(\*Res).sled = (\*q).sled; // Устанавливаем связь с последующим звеном

delete q; // Освобождаем память удаляемого звена

}

else if ((\*Res).sled == Res) // Проверяем, что удаляемое звено это единственное в кольце.

{

q = (\*phead).sled; // Устанавливаем указатель на единственное звено

(\*phead).sled = NULL; // Удаляем ссылку на единственное звено

delete q; // Освобождаем память единственного звена

cout << "Кольцо пусто!";

}

else // Если удаляемое звено находится в середине кольца

{

z = phead; // Устанавливаем указатель на заглавное звено

q = (\*phead).sled; // Устанавливаем указатель на первое звено в кольце

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

{

z = q; // Перемещаем указатель на предыдущее звено

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

}

(\*z).sled = (\*q).sled; // Устанавливаем связь между предыдущим и следующим звенами.

delete q; // Освобождаем память.

}

}

void Spisok::DelAfter()

// Метод для удаления следующего звена после указанного в кольце односвязного списка

{

node\* q; // Указатель на звено, которое нужно удалить

if ((\*Res).sled != (\*phead).sled) // Проверяем что удаляемое звено не является послдним в кольце

{

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

(\*Res).sled = (\*q).sled; // Устанавливаем связь между текущим и следующим звенами

delete q; // Освобождаем память удаляемого звена

}

else if ((\*Res).sled == Res) // Проверяем, что удаляемое звено - единственное в кольце.

{

q = (\*phead).sled; // Устанавливаем указатель на единственное звено

(\*phead).sled = NULL; // Удаляем ссылку на единственное звено.

delete q; // Освобождаем память.

cout << "Кольцо пусто!";

}

else // Если удаляемое звено находится в середине кольца.

{

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

(\*Res).sled = (\*q).sled; // Устанавливаем связь между текущим и следующим звенами

(\*phead).sled = (\*Res).sled; // Обновляем ссылку на начало кольца

delete q; // Освобождаем память

}

}

void Spisok::OCHISTKA()

// Метод для освобождения памяти, занятой всеми звеньями списка.

{

node\* q, \* q1; // Указатели на предыдущее и текущее звено.

q = phead; // Устанавливаем указатель на начало списка.

q1 = (\*q).sled; // Устанавливаем указатель на первое звено в списке

// Проходим по всем звеньям списка и освобождаем память

do {

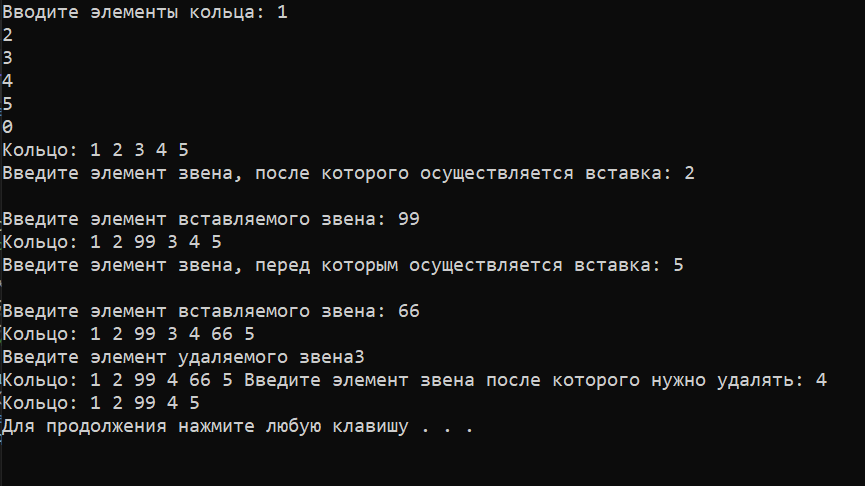
q = q1; // Переходим к следующему звену

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

delete q; // Освобождаем память предыдущего звена

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

}



Код Тестирования 3

#include <iostream>

using namespace std;

struct node

{

int elem; // Хранит значение элемента списка

node\* sled; // Указатель на следующий элемент спискаа

node\* pred; // Указатель на предыдущий элемент списка

};

class Spisok

{

private:

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

public:

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

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

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

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

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

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

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

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

node\* SearchRing(int); // Поиск звена по его элементу

void Ochistka(); // Очистка памяти, освобождение занимаемых звеньев

};

void main()

{

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

Spisok A; // Создаем объект класса Spisok.

node\* Res; // Указатель на узел списка.

int el, el1; // Переменные для хранения элементов.

A.BuiltRing(); // Создаем кольцевой список.

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

A.VyvodLeftRight();

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

A.VyvodRightLeft();

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

cout << "осуществляется вставка: ";

cin >> el;

cout << "Введите элемент вставляемого звена: ";

cin >> el1;

Res = A.SearchRing(el); // Ищем элемент в списке.

if (Res != NULL) { // Если элемент найден, выполняем вставку.

A.InsAfter(Res, el1);

A.VyvodLeftRight(); // Выводим содержимое списка.

}

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); // Ищем элемент в списке.

if (Res != NULL) { // Если элемент найден, выполняем вставку.

A.InsBefore(Res, el1);

A.VyvodLeftRight(); // Выводим содержимое списка.

}

else {

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

}

cout << "Введите элемент звена, который ";

cout << "надо удалить: ";

cin >> el;

Res = A.SearchRing(el); // Ищем элемент в списке.

if (Res != NULL) { // Если элемент найден, выполняем удаление.

A.Delete(Res);

A.VyvodLeftRight(); // Выводим содержимое списка.

}

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); // Ищем элемент в списке.

if (Res != NULL) { // Если элемент найден, выполняем удаление.

A.DelAfter(Res);

A.VyvodLeftRight(); // Выводим содержимое списка.

}

else {

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

}

A.Ochistka(); // Очищаем память, выделенную для списка.

cout << "\n";

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

}

void Spisok::BuiltRing()

// Построение двунаправленного кольцевого списка nsp

// с удаленным заглавным звеном.

// nsp - указатель на заглавное звено списка.

{

node\* r; // Объявление указателя

int el; // Объявление переменной для вводимых пользователем эл списка

//Построение заглавное звена кольцевого списка

nsp = new(node); // Выделение памяти под заглавное звено и он же есть nsp

r = nsp; // Присваивание r значение заглавного звена

(\*nsp).pred = NULL; (\*nsp).sled = NULL; // Установка указателей предыдущего и следующего звеньев заглавного звена на NULL

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

cin >> el; // Ввод пользоватедя эл списка

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

{

(\*r).sled = new (node); // Создание нового звена списка и присвоение указателю на следующий элемент адреса этого звена

(\*((\*r).sled)).pred = r; // Установка указателя на предыдущий элемент нового звена на тек элемент

r = (\*r).sled; // Перемещение указателя r на новое звено

(\*r).sled = NULL; // Установка указателя на следующий элемент нового звена на NULL

(\*r).elem = el; // Присваивание элементу нового звена значения, введеного пользователем

cin >> el; // Ввод пользоватедя эл списка

}

// образуем кольцевой список

if ((\*nsp).sled != NULL) // Проверка, не является ли список пустым

{

(\*((\*nsp).sled)).pred = r; // Установка указателя на предыдущий элемент первого звена на последний элемент списка

(\*r).sled = (\*nsp).sled; // Установка укателя на следующий элемент последнего звена на первый элемент списка

}

else // Если список пуст

cout << "Кольцевой список пуст!\n";

}

void Spisok::VyvodLeftRight()

// Вывод содержимого двунаправленного кольцевого списка

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

// nsp - указатель на заглавное звено списка.

{

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

cout << "Кольцевой список: ";

if ((\*nsp).sled != NULL) // Проверка, не является ли список пустым

{

cout << (\*((\*nsp).sled)).elem << " "; // Вывод значения элемента первого звена списка

r = (\*((\*nsp).sled)).sled; // Присваивание указателю r адреса следующего за первым звена

while (r != (\*nsp).sled) // Цикл продолжается, пока не дойдет до начального звена списка

{

cout << (\*r).elem << " "; // Вывод значения элемента текущего звена

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

}

cout << endl;

}

else cout << "пуст!";

}

void Spisok::VyvodRightLeft()

// Вывод содержимого двунаправленного кольцевого списка

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

// nsp - указатель на заглавное звено списка.

{

node\* r; // Объявление указателя

cout << "Кольцевой список: ";

if ((\*nsp).sled != NULL) // Проверка, не является ли список пустым

{

cout << (\*((\*((\*nsp).sled)).pred)).elem << " "; // Вывод значения элемента последнего звена списка

r = (\*((\*((\*nsp).sled)).pred)).pred; // Присваивание указателю r адреса предыдущего за последним звена

while (r != (\*((\*nsp).sled)).pred) // Цикл продолжается, пока не дойдет до начального звена списка

{

cout << (\*r).elem << " "; // Вывод значения элемента текущего звена

r = (\*r).pred; // Переход к предыдущему звену

}

cout << endl;

}

else cout << "пуст!";

}

node\* Spisok::SearchRing(int el)

// Поиск элемента el в кольцевом двунаправленном списке

// с удаленным заглавным звеном.

// nsp - указатель на заглавное звено списка.

{

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

node\* p; // Объявление указателя

node\* Res; // Объявление указателя для хранения результата поиска

Res = NULL; // указатель со значением NULL

p = nsp; // Присваивание указателю p адреса заглавного звена списка

if ((\*((\*p).sled)).elem == el) Res = (\*p).sled; // Проверка, является ли первый элемент списка искомым

else

{

q = (\*((\*p).sled)).sled; // Присваивание указателю q адреса второго звена списка

while (q != (\*p).sled && Res == NULL) // Цикл продолжается, пока не пройдет весь список или не найден искомый элемент

if ((\*q).elem == el) Res = q; // Если текущий элемент списка равен искомому, присваиваем его указателю Res

else q = (\*q).sled; // Переход к следующему звену списка

}

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

}

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

// Вставление в кольцевой двунаправленный список звена

// с информационным полем el после звена, на которое

// указывает ссылка Res.

{

node\* q; // Объявление указателя

q = new(node); // Выделение памяти под новое звено и присвоение адреса этой памяти указателю q

(\*q).elem = el; // Присвоение инф полю нового звена значения el

(\*q).sled = (\*Res).sled; // Присвопение указателю на следующее звено нового звена адрес следующего звена после Res

(\*q).pred = (\*(\*Res).sled).pred; // Присвоение указателю на предыдущее звено нового звена адрес предыдущего звена после Res

(\*(\*Res).sled).pred = q; // Присвоение предыдущему звену после Res адреса нового звена

(\*Res).sled = q; // Присвоение Res адреса нового звена

}

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

// Вставка в кольцевой двунаправленный список звена

// с информационным полем el перед звеном, на которое

// указывает ссылка Res.

// nsp - указатель на заглавное звено списка.

{

node\* q; // Объявление указателя на новое звено

q = new(node); // Выделение памяти под новое звено и рн жн теперь q

(\*q).elem = el; // Присвоение инфоу полю нового звена значения el

(\*q).sled = (\*(\*Res).pred).sled; // Присвоение указателю на следующее звено нового звена адрес следующего звена перед Res

(\*q).pred = (\*Res).pred; // Присвоение указателю на предыдущее звено нового звена адрес предыдущего звена перед Res

(\*(\*Res).pred).sled = q; // Присвоение следующему звену перед Res адреса нового звена

(\*Res).pred = q; // Присвоение Res адреса нового звена

if (Res == (\*nsp).sled) (\*nsp).sled = q; // Если Res указывает на первое звено в списке, то обновляем указатель списка на новое звено

}

void Spisok::Delete(node\* Res)

// Удаление из кольцевого двунаправленного списка

// звена, на которое указывает ссылка Res.

// nsp - указатель на заглавное звено списка.

{

if ((\*Res).sled == Res) // Проверка, является ли указ. Res последним звеном в списке

{

(\*nsp).sled = NULL; // Если да, то обнуляем указатель на следующее звено в заглавном звене

delete Res; // Удаляем последнее звено

}

else

{

(\*(\*Res).sled).pred = (\*Res).pred; // Присваиваем следующему звену перед Res адрес предыдущего звена перед Res

(\*(\*Res).pred).sled = (\*Res).sled; // Присваиваем предыдущему звену перед Res адрес следующего звена после Res

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

// Удаляем "первое" звено кольца.

(\*nsp).sled = (\*Res).sled; // Если Res указывает на первое звено в списке, то обновляем указатель списка на следующее звено после Res

delete Res; // Удаляем звено, на которое указывает Res

}

}

void Spisok::DelAfter(node\* Res)

// Удаление из кольцевого двунаправленного списка звена,

// расположенного после звена, на которое указывает

// ссылка Res.

// nsp - указатель на заглавное звено списка.

{

node\* q; // Объявление указателя q

if ((\*Res).sled == Res) // Проверка, является ли Res последним звеном в списке

{

(\*nsp).sled = NULL; // Если да, то обнуляем указатель на следующее звено в заглавном звене

delete Res; // Удаляем последнее звено

}

else

{

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

(\*(\*(\*Res).sled).sled).pred = (\*(\*Res).sled).pred; // Присваиваем следующему звену после Res адрес предыдущего звена перед следующим звеном после Res

(\*Res).sled = (\*(\*Res).sled).sled; // Присваиваем Res адрес следующего звена после следующего звена после Res

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

// Удаляем "последнее" звено кольца.

(\*nsp).sled = (\*Res).sled; // Если Res указывает на последнее звено в списке, то обновляем указатель списка на следующее звено после Res

delete q; // Удаляем следующее звено после Resт т есть q

}

}

void Spisok::Ochistka()

// Освобождение памяти, выделенной под кольцевой 2-направ. список

{

node\* q, \* q1; // Объявление указателей q и q1

q = (\*((\*nsp).sled)).sled; // Присваивание q адреса второго звена в списке

q1 = (\*q).sled; // Присваивание q1 адреса третьего звена в списке

while (q1 != (\*((\*nsp).sled)).sled) // Пока q1 не указывает на второе звено в списке

{

delete q; // Удаление текущего звена

q = q1; // Переход к следующему звену

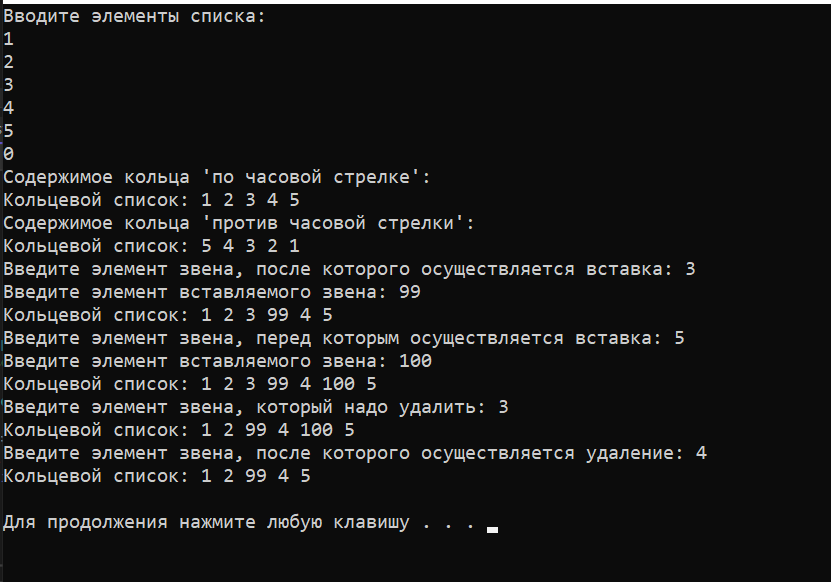
q1 = (\*q1).sled; // Обновление адреса следующего звена

}

delete q; // Удаление последнего звена

delete nsp; // Удаление заглавного звена

}



Код Тестирования 4

#include<iostream>

using namespace std;

// Структура для элемента стека

struct node

{

int elem; // Информационное поле

node\* sled; // Указатель на следующий элемент

};

// Класс для работы со стеком

class Spisok

{

private:

node\* stk; // Указатель на вершину стека

int klad; // Поле для хранения извлеченного элемента

public:

Spisok() { stk = NULL; } // Конструктор класса, инициализирует вершину стека как NULL

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

void POSTROENIE(); // Метод для построения стека

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

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

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

void OCHISTKA(); // Метод для освобождения памяти, выделенной под стек

};

// Основная функция программы

void main()

{

setlocale(LC\_ALL, "Rus"); // Установка локали для вывода на русском языке

Spisok A; // Создание объекта класса Spisok

int el; // Переменная для вводимого элемента

A.POSTROENIE(); // Построение стека

A.VYVOD(); // Вывод содержимого стека

cout << "Введите вставляемый элемент: ";

cin >> el; // Ввод элемента для вставки в стек

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

A.VYVOD(); // Вывод содержимого стека после вставки

cout << "Удалим элемент из стека.\n";

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

int t = A.Set\_Stack(); // Получение извлеченного элемента из стека

cout << "Из стека было извлечено число... " << t << endl;

A.VYVOD(); // Вывод содержимого стека после удаления

A.OCHISTKA(); // Освобождение памяти, выделенной под стек

cout << "\n";

system("PAUSE"); // Приостановка выполнения программы перед закрытием окна

}

// Метод POSTROENIE() для построения стека, заданного указателем stk.

void Spisok::POSTROENIE()

{

node\* t; // Временный указатель на узел

int el; // Переменная для вводимого элемента

cout << "Вводите элементы стека: ";

cin >> el; // Ввод первого элемента стека

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

{

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

(\*t).elem = el; // Запись значения в информационное поле узла

(\*t).sled = stk; // Присваивание указателю на следующий узел адреса вершины стека

stk = t; // Перемещение вершины стека на новый узел

cin >> el; // Считывание следующего ввода

}

}

void Spisok::VYVOD()

{

node\* t; // Временный указатель на узел

cout << "Содержимое стека: ";

t = stk; // Начальное присваивание временному указателю адреса вершины стека

while (t != NULL) // Пока временный указатель не указывает на нуль (конец стека)

{

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

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

}

cout << endl;

}

// Метод W\_S(int el) для помещения элемента el в стек stk.

void Spisok::W\_S(int el)

{

node\* q; // Временный указатель на узел

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

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

(\*q).sled = stk; // Присваивание новому узлу адреса предыдущей вершины стека

stk = q; // Обновление указателя на вершину стека

}

// Значение информационного поля удаляемого элемента помещается в параметр klad.

void Spisok::YDALENIE()

{

node\* q; // Временный указатель на узел

if (stk == NULL) // Проверка, пуст ли стек

cout << "Стек пуст!\n";

else

{

klad = (\*stk).elem; // Сохранение значения элемента для возврата

q = stk; // Присваивание временному указателю адреса вершины стека

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

delete q; // Освобождение памяти, занимаемой удаляемым узлом

}

}

// Метод OCHISTKA() для возврата выделенной памяти в "кучу"

void Spisok::OCHISTKA()

{

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

t = stk; // Присваивание временному указателю адреса вершины стека

if (t != NULL) // Проверка, пуст ли стек

{

q = (\*t).sled; // Присваивание временному указателю адреса следующего узла

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

{

delete t; // Освобождение памяти, занимаемой текущим узлом

t = q; // Переход к следующему узлу

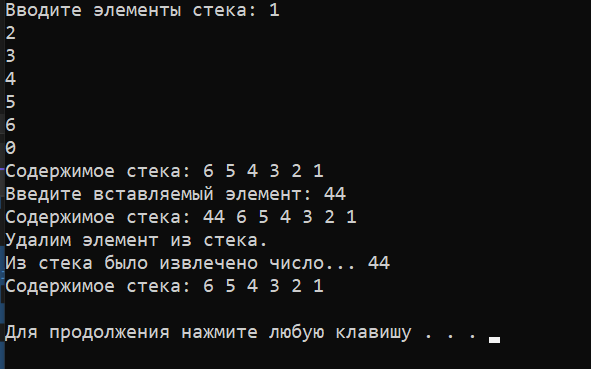
q = (\*q).sled; // Переход к следующему узлу

}

delete t; // Освобождение памяти, занимаемой последним узлом

}

}



Код Тестирования 5

#include <iostream>

#include <string.h> // Включение заголовочного файла для работы со строками

using namespace std;

const int MAXSIZE = 100; // Максимальный размер стека

// Структура для стека

struct Stack

{

char data[MAXSIZE]; // Массив данных стека

int size; // Размер стека

};

// Функция для добавления элемента в стек

void Push(Stack& S, char x)

{

if (S.size == MAXSIZE) { // Проверка на переполнение стека

printf("Стек переполнен");

return;

}

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

S.size++; // Увеличение размера стека

}

// Функция для удаления элемента из стека

char Pop(Stack& S)

{

if (S.size == 0) { // Проверка на пустоту стека

printf("Стек пуст");

return char(255); // Возврат символа, обозначающего ошибку

}

S.size--; // Уменьшение размера стека

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

}

// Основная функция программы

void main()

{

setlocale(LC\_ALL, "Rus");

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

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

char s[80], upper; // Входная строка и переменная для хранения верхнего элемента стека

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

Stack S; // Стек символов

printf("Введите выражение со скобками> ");

gets\_s(s); // Получение строки с клавиатуры

S.size = 0; // Начальный размер стека равен 0

OK = 1; // Переменная, отвечающая за корректность выражения, изначально установлена в 1

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

for (k = 0; k < 3; k++) { // Проверка каждой из трех скобок

if (s[i] == br1[k]) { // Если символ является открывающей скобкой

Push(S, s[i]); // Добавляем его в стек

break; // Прерываем цикл проверки скобок

}

if (s[i] == br2[k]) { // Если символ является закрывающей скобкой

upper = Pop(S); // Извлекаем верхний элемент стека

if (upper != br1[k]) // Если верхний элемент не соответствует соответствующей открывающей скобке

OK = 0; // Устанавливаем флаг корректности в 0

break; // Прерываем цикл проверки скобок

}

}

if (OK && (S.size == 0)) // Если проверка завершилась без ошибок и стек пуст

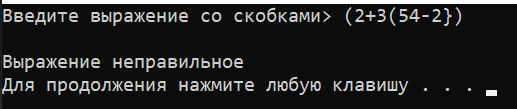
printf("\nВыражение правильное\n");

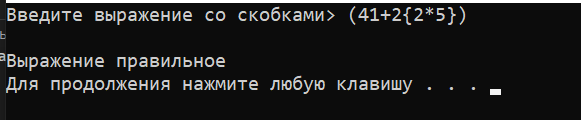
else

printf("\nВыражение неправильное \n");

system("PAUSE"); // Приостановка выполнения программы перед закрытием окна

}





Код Тестирования 6

#include<iostream>

using namespace std;

struct node

{

int elem;

node\* sled;

};

class Spisok

{

private:

node\* ld, \* rd;

int el\_left, el\_right;

public:

void POSTROENIE();

void VYVOD();

void VSTAV1(int);

void VSTAV2(int);

int SetElLeft() { return el\_left; }

int SetElRight() { return el\_right; }

void YDALE1();

void YDALE2();

void OCHISTKA();

};

void main()

{

setlocale(LC\_ALL, "Rus");

Spisok A;

int el;

A.POSTROENIE(); A.VYVOD();

cout << "Добавим звено справа.\n";

cout << "Введите элемент добавляемого звена: ";

cin >> el;

A.VSTAV1(el); A.VYVOD();

cout << "Добавим звено слева.\n";

cout << "Введите элемент добавляемого звена: ";

cin >> el;

A.VSTAV2(el); A.VYVOD();

cout << "Удалим звено справа.\n";

A.YDALE1(); A.VYVOD(); cout << A.SetElRight() << endl;

cout << "Удалим зввено слева.\n";

A.YDALE2(); A.VYVOD(); cout << A.SetElLeft() << endl;

A.OCHISTKA();

cout << "\n";

system("PAUSE");

}

// ld - указатель на левый конец дека,

// rd - Указатель на правый конец дека.

void Spisok::POSTROENIE()

{

node\* k; // Временный указатель на узел

int el; // Элемент дека, вводимый пользователем

cout << "Вводите содержимое звеньев дека: \n"; // Предложение для пользователя

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; // Инициализация нового узла

ld = k; rd = k; cin >> el; // Присваивание указателю на левый конец адреса первого узла

while (el != 0) // Цикл, пока пользователь вводит ненулевые элементы

{

VSTAV1(el); cin >> el; // Вызов метода для вставки элемента справа

}

}

else // Если первый элемент равен 0

{

rd = NULL; ld = NULL; // Установка указателей на конец дека в NULL

}

}

// Метод VYVOD() для вывода содержимого дека:

// ld - указатель на левый конец дека.

void Spisok::VYVOD()

{

node\* k; // Временный указатель на узел

k = ld;// присваивание временному указателю адреса левого конца

cout << "Дек: ";

while (k != NULL) // Цикл, пока не достигнут конец дека

{

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

}

cout << endl;

}

// ld - указатель на левый конец дека,

// rd - указатель на правый конец дека.

void Spisok::VSTAV1(int el)

{

node\* k; // Временный указатель на узел

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

(\*k).elem = el; (\*k).sled = NULL; // Инициализация нового узла

if (rd != NULL) // Если дек не пуст

{

(\*rd).sled = k; rd = k; // Присоединение нового узла к деку справа

}

else // Если дек пуст

{

rd = k; ld = k; // Установка указателей на конец дека на новый узел

}

}

// ld - указатель на левый конец дека,

// rd - указатель на правый конец дека.

void Spisok::VSTAV2(int el)

{

node\* k; // Временный указатель на узел

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

(\*k).elem = el; (\*k).sled = ld; // Инициализация нового узла

if (ld != NULL) ld = k; // Если дек не пуст, новый узел становится левым концом

else { ld = k; rd = k; } // Если дек пуст, установка указателей на конец дека на новый узел

}

// Метод YDALE2() для удаления звена из дека справа

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

// ld - указатель на левый конец дека,

// rd - указатель на правый конец дека.

void Spisok::YDALE1()

{

node\* z; // Временный указатель на узел

node\* k; // Временный указатель на узел

if (rd == ld) // Если в деке только одно звено

{

el\_right = (\*rd).elem; delete rd; // Сохранение значения и удаление звена

ld = rd = NULL;// Установка указателей на конец дека в NULL

cout << "Дек пуст!\n";

}

else // Если в деке больше одного звена

{

z = ld; k = (\*ld).sled; // Присваивание временным указателям адресов

while (k != rd) // Пока не найдено звено, предшествующее правому концу

{

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

}

el\_right = (\*rd).elem; (\*z).sled = NULL; delete rd; // Сохранение значения и удаление звена

rd = z; // Установка правого конца на предыдущее звено

}

}

// Метод YDALE2() для удаления звена из дека слева

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

// ld - указатель на левый конец дека,

// rd - указатель на правый конец дека.

void Spisok::YDALE2()

{

node\* q; // Временный указатель на узел

if (ld != NULL) // Если дек не пуст

{

el\_left = (\*ld).elem; q = ld; // Сохранение значения и присвоение временному указателю адреса левого конца

ld = (\*ld).sled; delete q; // Установка левого конца на следующее звено и удаление звена

}

else cout << "Дек пуст!\n"; // Вывод сообщения, если дек пуст

}

// Метод OCHISTKA() для возврата выделенной памяти в "кучу".

void Spisok::OCHISTKA()

{

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

k = ld; // Присваивание временному указателю адреса левого конца

if (k != NULL) // Если дек не пуст

{

q = (\*k).sled; // Присваивание временному указателю адреса следующего узла

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

{

delete k; // Освобождение памяти, занимаемой текущим узлом

k = q; // Переход к следующему узлу

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

}

delete k; // Освобождение памяти, занимаемой последним узлом

}

}

