Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Реферат

По дисциплине «Основы алгоритмизации и программирования»

На тему «Тестирование кода структур данных»

Выполнила:

Студент(ка) 1 курса 7 группы

Подшиваленко Диана Игоревна

Проверил:

Белодед Николай Иванович

2024, Минск

1. **Тестирование кода “Гирлянды”**

#include <iostream>

#include <Windows.h>

using namespace std;

struct nodeVis { // структра звена висюльки

int elem; // информационное поле звена висюльки

nodeVis\* vniz; // указатель на звено висюльки

};

struct nodeGir { // структура звена гирлянды

int elem; // информационное поле звена гирлянды

nodeVis\* vniz; // указатель на звено висюльки

nodeGir\* sled; // указатедб на звено гирлянды

};

class GirVis { // класс для гирлянды

private: // приватная секция

nodeGir\* phead; // голова гирлянды

nodeVis\* pheadVis; // голова висюльки

void VisVyvod(); // функция для вывода висюльки

public: // пубичная секция

GirVis() { // конструктор

phead = new nodeGir; // выделение памяти для гирлянды

}

~GirVis() { // деструктор

delete phead; // очищение(освобождение) памяти

}

nodeVis\* VisPostr(); // прототип функции построения висюльки

nodeVis\* VisPoisk(int); // прототип функции поиска элемента в висюльке

void SetpheadVis(nodeVis\* r) { // функция для установки головы висюльки

pheadVis = r;

}

void VisVstav(nodeVis\*, int); // прототип функции вставки элемента в висюльку после заданного

void Vis1Vstav(nodeVis\*, int);// прототип функции вставки элемента в висюльку перед заданным

void VisUdale(nodeVis\*); // прототип функции удаления элемента в висюльке после введенного

void Vis1Udale(nodeVis\*); // прототип удаления заданного элемента в висюльке

void GirPostr(); // прототип функции построения гирлянды

void GirVyvod(); // прототип функции вывода гирлянды

nodeGir\* GirPoisk(int); // прототип функции поиска элемента гирлянды

void OCHISTKA(); // прототип функции освобождения памяти, занятой гирляндой

void OCHISTKA1(); // прототип функции освобождения памяти, занятой висюлькой

};

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

GirVis A; // создаем объект класса GirVis

int el, elGir, elVis; // буферные переменные для хранения значения элемента, звена гирлянды, звена висюльки соответственно

nodeGir\* Res; // буферный указатель на элемент гирлянды

nodeVis\* ResVis; // буферный указатель на элемент висюльки

A.GirPostr();

A.GirVyvod();

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

cin >> elGir;

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

cin >> elVis;

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

cin >> el;

Res = A.GirPoisk(elGir);

if (Res != NULL) {

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";

}

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

cin >> elGir;

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

cin >> elVis;

cout << "Введите вставляемый элемент\n";

cin >> el;

Res = A.GirPoisk(elGir);

if (Res != NULL) {

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

ResVis = A.VisPoisk(elVis);

if (ResVis != NULL) {

A.Vis1Vstav(ResVis, el);

}

else {

cout << "Элемента нет в висюльке\n";

}

}

else {

cout << "Элемента нет в гирлянде\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;

Res = A.GirPoisk(elGir);

if (Res != NULL) {

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

ResVis = A.VisPoisk(elVis);

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

A.VisUdale(ResVis);

}

else {

cout << "Элемента нет в висюльке\n";

}

}

else {

cout << "Элемента нет в гирлянде\n";

}

A.GirVyvod();

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

cin >> elGir;

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

cin >> elVis;

Res = A.GirPoisk(elGir);

if (Res != NULL) {

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

ResVis = A.VisPoisk(elVis);

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

A.Vis1Udale(ResVis);

}

else {

cout << "Элемента нет в висюльке\n";

}

}

else {

cout << "Элемента нет в гирлянде\n";

}

A.GirVyvod();

A.OCHISTKA();

cout << '\n';

}

void GirVis::OCHISTKA() { // функция освобождения памяти, занятой гирляндой

nodeGir\* q, \* q1; // буферные указатели

q = phead; // присваиваем указателю q адрес головы гирлянды

q1 = (\*q).sled; // присваиваем указателю q1 адрес следующего элемента гирлянды

while (q1 != NULL) { // пока q1 не будет NULL

q = q1; // присваиваем указателю q адрес, на который ссылается q1

q1 = (\*q1).sled; // присваиваем указателю q1 адрес следующего за ним элемента

pheadVis = (\*q).vniz; // передвигаем голову висюльки на следующий элемент

OCHISTKA1(); // вызываем функцию для очистки висюльки

delete q; // освобождаем память, на которую ссылается q

}

}

void GirVis::OCHISTKA1() { // функция очистки висюльки

nodeVis\* q, \* q1;// буферные указатели

q = pheadVis; // присваиваем указателю q адрес головы висюльки

q1 = (\*q).vniz; // присваиваем указателю q1 адрес следующего элемента висюльки

while (q1 != NULL) {// пока q1 не будет NULL

q = q1;// присваиваем указателю q адрес, на который ссылается q1

q1 = (\*q1).vniz;// присваиваем указателю q1 адрес следующего за ним элемента

delete q;// освобождаем память, на которую ссылается q

}

}

void GirVis::GirPostr() { // функция построения гирлянды

nodeGir\* t; // буферный указатель

int el; // буферная переменная для вводимых элементов

t = phead; // присваиваем указателю t адрес головы гирлянды

(\*t).sled = NULL; // присваиваем указателю на следующий элемент NULL

cout << "Вводите элементы гирлянды\n";

cin >> el; // считываем элементы, вводимые пользователем

while (el != 0) { // делаем это до тех пор, пока не будет введен признак окончания(значение 0)

(\*t).sled = new nodeGir; // выделяем помять под следующий элемент

t = (\*t).sled; // присваиваем буферному указателю адрес следующего элемента

(\*t).elem = el; // записываем в информационное поле текущее считанное значение

(\*t).sled = NULL; // присваиваем указателю на следующий элемент NULL

(\*t).vniz = VisPostr(); // вызываем функцию построения висюльки

cout << "Вводите элемент гирлянды\n";

cin >> el; // считываем следующий элемент

}

}

nodeVis\* GirVis::VisPostr() { // функция построения висюльки

nodeVis\* t;// буферный указатель

int el;// буферная переменная для вводимых элементов

pheadVis = new nodeVis; // выделяем память для головы висюльки

t = pheadVis; // присваиваем указателю t адрес головы висюльки

(\*t).vniz = NULL; // присваиваем указателю на следующий элемент NULL

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

cin >> el; // считываем новый элемент висюльки

while (el != 0) {// делаем это до тех пор, пока не будет введен признак окончания(значение 0)

(\*t).vniz = new nodeVis;// выделяем помять под следующий элемент

t = (\*t).vniz;// присваиваем буферному указателю адрес следующего элемента

(\*t).elem = el;// записываем в информационное поле текущее считанное значение

(\*t).vniz = NULL; // присваиваем указателю на следующий элемент NULL

cin >> el;// считываем следующий элемент

}

return pheadVis; // возвращаем указатель на построенную висюльку

}

void GirVis::GirVyvod() { // функция вывода гирлянды

nodeGir\* t;// буферный указатель

t = phead; // присваиваем указателю t адрес головы гирлянды

t = (\*t).sled; // перемещаемся на следующий элемент

cout << "Гирлянда\n";

while (t != NULL) { // пока не дошли до конца гирлянды

cout << (\*t).elem << ' '; // выводим текущий элемент

pheadVis = (\*t).vniz; // передвигаем голову висюльки на следующий элемент

VisVyvod(); // вызываем функцию вывода элементов висюльки

t = (\*t).sled; // передвигаемся на следующий элемент

}

}

nodeGir\* GirVis::GirPoisk(int el) { // функция поиска элемента в гирлянде

nodeGir\* t, \* r; // буферные указатели

r = NULL; // присваиваем указателю r значение NULL

t = phead; // присваиваем указателю t адрес головы гирлянды

t = (\*t).sled; // передвигаемся на следующий элемент

while (t != NULL && r == NULL) { // пока не дошли до конца и пока не нашли нужный элемент

if ((\*t).elem == el) { // проверяем элемент на соответствие нужному

r = t; // если элемент найден, то присваиваем соответствующему указателю адрес этого элемента

}

else { // иначе

t = (\*t).sled; // переходим на следующий элемент

}

}

return r; // возвращаем адрес элемента

}

void GirVis::VisVyvod() { // функция вывода элементов висюльки

nodeVis\* t; // буферный указатель

t = pheadVis;// присваиваем указателю t адрес головы висюльки

t = (\*t).vniz; // переходим н аследующий элемент

cout << '(';

while (t != NULL) { // пока не дошли до конца висюльки

cout << (\*t).elem << ' '; // выводим значение элемента

t = (\*t).vniz; // переходим на следующий

}

cout << ')';

}

nodeVis\* GirVis::VisPoisk(int el) { // функция поиска элемента висюльки

nodeVis\* t, \* r; // буферные указатели

r = NULL; // присваиваем указателю r значение NULL

t = pheadVis;// присваиваем указателю t адрес головы висюльки

t = (\*t).vniz;// передвигаемся на следующий элемент

while (t != NULL && r == NULL) {// пока не дошли до конца и пока не нашли нужный элемент

if ((\*t).elem == el) {// проверяем элемент на соответствие нужному

r = t;// если элемент найден, то присваиваем соответствующему указателю адрес этого элемента

}

else {// иначе

t = (\*t).vniz;// переходим на следующий элемент

}

}

return r; // возвращаем адрес элемента

}

void GirVis::VisVstav(nodeVis\* r, int el) { // функция для вставки элемента в висюльку после указанного

nodeVis\* q; // буферный указатель

q = new nodeVis; // выделяем память для нового элемента

(\*q).elem = el; // записываем в информационное поле заданное значение

(\*q).vniz = (\*r).vniz; // присваиваем указателю нового элемента на следующий элемент адрес следующего элемента, за которым мы вставляем

(\*r).vniz = q; // присваиваем указателю элемента, за которым мы вставляем, на следующий элемент адрес нового элемента

}

void GirVis::Vis1Vstav(nodeVis\* r, int el) { // функция вставки элемента в висюльку перед указанным

nodeVis\* q;// буферный указатель

q = new nodeVis; // выделяем память для нового элемента

(\*q).elem = (\*r).elem; // присваиваем информационному полю нового элемента значение того, перед которым мы должны вставить

(\*q).vniz = (\*r).vniz; // присваиваем указателю нового элемента на следующий элемент адрес следующего элемента, перед которым мы вставляем

(\*r).elem = el; // присваиваем тому элементу новое значение

(\*r).vniz = q; // устанавливаем адрес следующего элемента

}

void GirVis::VisUdale(nodeVis\* r) { // функция удаления элемента после заданного

nodeVis\* q;// буферный указатель

q = (\*r).vniz; // переходим на следующий элемент после заданного

if ((\*r).vniz != NULL) { // если это не последний элемент

(\*r).vniz = (\*(\*r).vniz).vniz; // то переносим связи на следующий после удаляемого

delete q; // освобождаем память

}

else {// иначе выводим сообщение, что данный элемент последний

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

}

}

void GirVis::Vis1Udale(nodeVis\* r) { // функция удаления заданного элемента

nodeVis\* g;// буферный указатель

if ((\*r).vniz != NULL) { // если это не последний элемент

g = (\*r).vniz; // присваиваем буферному указателю адрес следующего за удаляемым элементом элемента

(\*r).elem = (\*(\*r).vniz).elem; // переносим информационное поле

(\*r).vniz = (\*(\*r).vniz).vniz; // переносим связи на следующий за удаляемым элемент

delete g; // освобождаем память

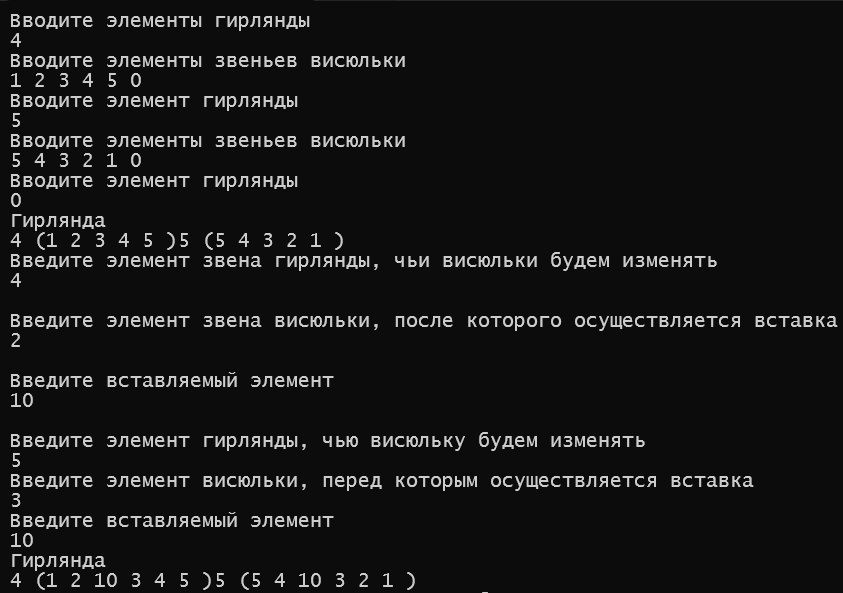
}

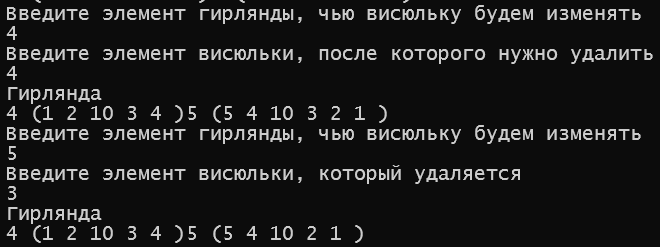
else { // иначе выводим сообщение

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

}

}

Результат выполнения:



1. **Тестирование кода “Кольцевого списка”**

#include <iostream>

#include <Windows.h>

using namespace std;

struct node { // структура узла списка

int elem; // инофрмационное поле

node\* sled; // указатель на следующий элемент

};

class Spisok { // клсс списка

private: // приватная секция

node\* phead, \* Res; // указатель на голову и дополнительный

public: // публичная секция

Spisok() { // конструктор

phead = new node; // выделяем память для головы

Res = NULL; // присваиваем Res NULL

}

~Spisok() { // деструктор

delete phead; // очищаем память

}

void POSTROENIE(); // прототип функции построения списка

void VYVOD(); // прототип функции вывода

node\* POISK(int); // прототип функции поиска

void InsAfter(int); // прототип функции вставки после заданного элемента

void InsBefore(int); // прототип функции вставки до заданного элемента

void Delete(); // прототип функции удаления хданного элемента

void DelAfter(); // прототип функции удаления после заданного элемента

void OCHISTKA(); // прототип функции очистки

};

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

Spisok A; // создаем объект класса Spisok

int el, el1; // буферные переменные для хранения вводимых элементов

A.POSTROENIE();

A.VYVOD();

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

cin >> el;

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

cin >> el1;

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

A.InsAfter(el1);

A.VYVOD();

}

else {

cout << "Звена с заданным элементом в кольце нет\n";

}

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

cin >> el;

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

cin >> el1;

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

A.InsBefore(el1);

A.VYVOD();

}

else {

cout << "Звена с заданным элементом в кольце нет\n";

}

cout << "Введите элемент удаляемого звена\n";

cin >> el;

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

A.Delete();

A.VYVOD();

}

else {

cout << "Звена с заданным элементом в кольце нет\n";

}

cout << "Введите элемент звена, после которого нужно удалять\n";

cin >> el;

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

A.DelAfter();

A.VYVOD();

}

else {

cout << "Звена с заданным элементом в кольце нет\n";

}

A.OCHISTKA();

cout << '\n';

}

void Spisok::POSTROENIE() { // функция построения

node\* t; // буферный указатель

int el; // буферная переменная

t = phead; // устанавливаем буферный указатель на голову

(\*t).sled = NULL; // указатель на следующий элемент устанавливаем NULL

cout << "Вводите элементы кольца\n";

cin >> el; // считываем текущий новый элемент

while (el != 0) { // пока вводимый элемент не 0(признак окончания ввода)

(\*t).sled = new node; // выделяем память под следующий элемент

t = (\*t).sled; // переходим на него

(\*t).elem = el; // записываем в информационное поле текущее введенное значение

cin >> el; // считываем следующий элемент

}

(\*t).sled = (\*phead).sled; // присваиваем указателю на следующий элемент адрес головы для замыкания списка

}

void Spisok::VYVOD() { // функция вывода

node\* t; // буферный указатель

t = (\*phead).sled; // ссылаем его на следующий после головы элемент

cout << "Кольцо\n";

if (t != NULL) { // пока не дошли до конца

cout << (\*t).elem << ' '; // выводим значение текущего элемента

t = (\*t).sled; // переходим на следующий

while (t != (\*phead).sled) { // пока не вернулись в начало списка

cout << (\*t).elem << ' '; // выводим элементы

t = (\*t).sled; // переходим на следующий

}

}

else {// иначе выводим сообщение

cout << "пусто\n";

}

}

node\* Spisok::POISK(int el) { // функция поиска

node\* t; // буферный указатель

Res = NULL; // пока элемент не найден(NULL)

t = (\*phead).sled; // буферный указатель ссылаем на следующий элемент после головы

while ((\*t).sled != (\*phead).sled && Res == NULL) { // пока не вернулись в начало списка и пока не нашли искомый элемент

if ((\*t).elem == el) { // сравниваем текущий с искомым

Res = t; // если нашли, то записываем в указатель для искомого элемента адрес найденного элемента

}

else { // иначе

t = (\*t).sled; // переходим на следующий

}

}

if (Res == NULL && (\*t).elem == el) { // если это оказался последний элемент

Res = t; //то записываем в указатель для искомого элемента адрес найденного элемента

}

return Res; // возвращаем адрес найденного элемента

}

void Spisok::InsAfter(int el) { // функция вставки после заданного

node\* q; // буферный указатель

q = new node; // выделяем память

(\*q).elem = el; // записываем в информационное поле требуемое значение

(\*q).sled = (\*Res).sled; // присваиваем указателю на следующий элемент адрес следующего за ссылаемым элементом

(\*Res).sled = q; // присваиваем указателю на следующий элемент, после которого должны вставить, адрес нового элемента

}

void Spisok::InsBefore(int el) { // функция вставки перед заданным

node\* q;// буферный указатель

q = new node; // выделяем память

(\*q).elem = (\*Res).elem;// записываем в информационное поле значение предыдущего

(\*q).sled = (\*Res).sled; // присваиваем адресу следующего значение следующего элемента предыдущего

(\*Res).elem = el; // меняем местами

(\*Res).sled = q; // присваиваем указателю на следующий элемент, перед которым должны вставить, адрес нового элемента

}

void Spisok::Delete() { // функция для удаления заданного элемента

node\* z, \* q; // буферные указатели

if ((\*Res).sled != (\*phead).sled) { // если не дошли до начала списка

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

(\*Res).elem = (\*((\*Res).sled)).elem;// записываем в информационное поле значение следующего

(\*Res).sled = (\*((\*Res).sled)).sled; // меняем связи

delete q; // очищаем память

}

else if ((\*Res).sled == Res) { // если у нас один элемент

q = (\*phead).sled; // присваиваем q адрес следующего за головой элемента

(\*phead).sled = NULL; // и присваиваем NULL

delete q;// очищаем память

cout << "Кольцо пусто\n";

}

else { // иначе

z = phead; // присваиваем буферному указателю адрес головы

q = (\*phead).sled; // присваиваем указателю q адрес следующего за головой элемента

while (q != Res) { // пока не дошли до начала

z = q; // переходим на следующий элемент

q = (\*q).sled;// переходим на следующий элемент

}

(\*z).sled = (\*((\*z).sled)).sled; // переходим на следующий элемент

delete q;// очищаем память

}

}

void Spisok::DelAfter() {

node\* q;// буферный указатель

if ((\*Res).sled != (\*phead).sled) {// пока не дошли до начала списка

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

(\*Res).sled = (\*((\*Res).sled)).sled;// меняем связи

delete q;// очищаем память

}

else if ((\*Res).sled == Res) {// если у нас один элемент

q = (\*phead).sled;// присваиваем указателю q адрес следующего за головой элемента

(\*phead).sled = NULL; // и присваиваем NULL

delete q;// очищаем память

cout << "Кольцо пусто\n";

}

else { // иначе

q = (\*phead).sled;// присваиваем указателю q адрес следующего за головой элемента

(\*Res).sled = (\*((\*Res).sled)).sled;// переходим на следующий элемент

(\*phead).sled = (\*Res).sled;// переходим на следующий элемент

delete q;// очищаем память

}

}

void Spisok::OCHISTKA() { // функция очистки

node\* q, \* q1;// буферные указатели

q = phead; // присваиваем адрес головы

q1 = (\*q).sled; // присваиваем адрес следующего за q элемента

do { // делаем

q = q1; // переходим на следующий элемент

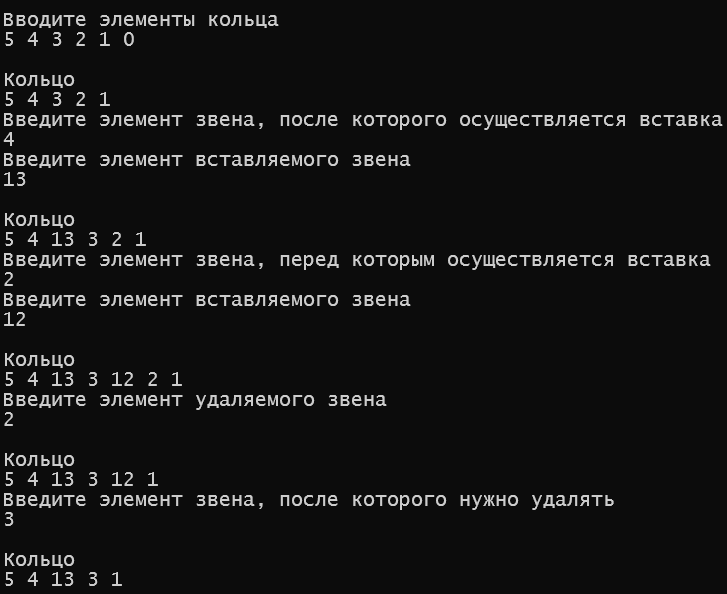
q1 = (\*q1).sled; // переходим на следующий элемент

delete q;// очищаем память

} while (q1 != (\*phead).sled); // пока не вернулись в начало списка

}

Результат выполнения:



1. **Тестирование кода “Двунаправленного кольцевого списка”**

#include <iostream>

using namespace std;

struct node // структура узла списка

{

int elem; // информационное поле

node\* sled; // указатель на следующий элемент

node\* pred; // указатель на предыдущий

};

class Spisok // класс списка

{

private: // приватная секция

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

public: // публичная секция

Spisok() { //Конструктор конструктор списка

nsp = NULL; // присваиваем указатель на начало NULL

}

void BuiltRing(); // прототип функции для построения списка

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

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

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

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

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

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

node\* SearchRing(int); //прототип функции для поиска узла с указанным значением

void Ochistka(); //прототип функции для очистки списка

};

void main()

{

setlocale(LC\_ALL, "Rus");

Spisok A; // создаем объкт класса Spisok

node\* Res; // буферный указатель

int el, el1; // буферные переменнные для считывания занчений с клавиатуры

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() // функция для построения списка

{

node\* r; // буферный указатель

int el; // буферная переменнная

nsp = new(node); // Выделения памяти для начального элемента

r = nsp; // присваиваем буферному указателю адрес начала списка

(\*nsp).pred = NULL; (\*nsp).sled = NULL; // инициализируем указатели предыдущего и следующего звена значением NULL

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

cin >> el; // считываем новый элемент списка

while (el != 0) // пока не встретился 0(признак окончания ввода)

{

(\*r).sled = new (node); // выделяем память для нового звена

(\*((\*r).sled)).pred = r; // перенастраиваем указатель предыдущего звена

r = (\*r).sled; // переходим на следующий элемент

(\*r).sled = NULL; // присваиваем указателю на следующий элемент NULL

(\*r).elem = el; // записываем в информационное поле текущее значение

cin >> el; // считываем следующий элемент

}

if ((\*nsp).sled != NULL) //если только начальный элемент

{

(\*((\*nsp).sled)).pred = r; // устанавливаем указатель на предыдущий следующего за начальным звена на r

(\*r).sled = (\*nsp).sled; // устанавливаем указатель следующего на следующее за начальным звено

}

else

cout << "Кольцевой список пуст!\n";

}

void Spisok::VyvodLeftRight() // функция для вывода элементов слева направо

{

node\* r; // буферный указатель

cout << "Кольцевой список: ";

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

{

cout << (\*((\*nsp).sled)).elem << " "; // выводим значение следующего за заглавным звена

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

while (r != (\*nsp).sled) // пока мы не вернулись в начало списка

{

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

r = (\*r).sled; // переходим на следующий элемент

}

cout << endl;

}

else

cout << "пуст!";

}

void Spisok::VyvodRightLeft() // функция для вывода списка справа налево

{

node\* r; // буферный указатель

cout << "Кольцевой список: ";

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

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

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

while (r != (\*((\*nsp).sled)).pred) // пока мы не вернулись к последнему

{

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

r = (\*r).pred;// переходим на предыдущий элемент

}

cout << endl;

}

else cout << "пуст!";

}

node\* Spisok::SearchRing(int el) // функция для поиска элемента

{

node\* q; // буферный указатель

node\* p; // указатель на начало списка

node\* Res; // указатель на найденное звено

Res = NULL; // изначально элемент не найден

p = nsp; // присваиваем указателю p адрес начального звена

if ((\*((\*p).sled)).elem == el) Res = (\*p).sled; // если первый элемент является искомым, то присваиваем указателю Res его адрес

else { // иначе

q = (\*((\*p).sled)).sled; // переходим на следующий элемент

while (q != (\*p).sled && Res == NULL) // перебираем все остальные звенья

if ((\*q).elem == el) Res = q; // если нашли, то присваиваем указателю Res его адрес

else q = (\*q).sled; // иначе переходим на следующий элемент

}

return Res; // возвращаем адрес найденного элемента

}

void Spisok::InsAfter(node\* Res, int el) // функция для вставки элемента после заданного

{

node\* q; // буферный указатель

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

(\*q).elem = el; // записываем в информационное поле нового элемента наше значение

(\*q).sled = (\*Res).sled; (\*q).pred = (\*(\*Res).sled).pred;// устанавливаем связи

(\*(\*Res).sled).pred = q; (\*Res).sled = q; // устанавливаем связи

}

void Spisok::InsBefore(node\* Res, int el) // функция вставки элемента перед заданным

{

node\* q; // буферный указатель

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

(\*q).elem = el; // записываем в информационное поле нового элемента наше значение

(\*q).sled = (\*(\*Res).pred).sled; (\*q).pred = (\*Res).pred; // устанавливаем связи

(\*(\*Res).pred).sled = q; (\*Res).pred = q; // устанавливаем связи

if (Res == (\*nsp).sled) (\*nsp).sled = q; // если вставка перед первым элементом после заглавного, то вставляем его после заглавного

}

void Spisok::Delete(node\* Res) // функция для удаления заданного элемента

{

if ((\*Res).sled == Res){// если у нас только один элемент в списке

(\*nsp).sled = NULL; delete Res; // устанавливаем указатель на начало в NULL и освобождаем память

}

else { // иначе

(\*(\*Res).sled).pred = (\*Res).pred; // переустанавливаем связи

(\*(\*Res).pred).sled = (\*Res).sled; // переустанавливаем связи

if ((\*nsp).sled == Res) // если это первый элемент после заглавного

(\*nsp).sled = (\*Res).sled; // обновляем указатель начала списка

delete Res; // очищаем память

}

}

void Spisok::DelAfter(node\* Res) // функция для удаления элемента после заданного

{

node\* q; // буфрный указатель

if ((\*Res).sled == Res){ // если у нас один элемент в списке

(\*nsp).sled = NULL; delete Res; // устанавливаем указатель на начало в NULL и освобождаем память

}

else{ // иначе

q = (\*Res).sled; // устанавливаем указатель q на следующее за нужным звеном

(\*(\*(\*Res).sled).sled).pred = (\*(\*Res).sled).pred; // переустанавливаем связи

(\*Res).sled = (\*(\*Res).sled).sled; // переустанавливаем связи

if ((\*(\*nsp).sled).pred == Res) // если это последнее звено

(\*nsp).sled = (\*Res).sled; // обновляем указатель на предыдущий элемент первого элемента после заглавного

delete q; // очищаем память

}

}

void Spisok::Ochistka() // функция для очистки списка

{

node\* q, \* q1; // буферные указатели

q = (\*((\*nsp).sled)).sled; //устанавливаем указатель на второе звено

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

while (q1 != (\*((\*nsp).sled)).sled) // пока мы не вернулись в начало

{

delete q; // очищаем память

q = q1; // переходим на следующее звено

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

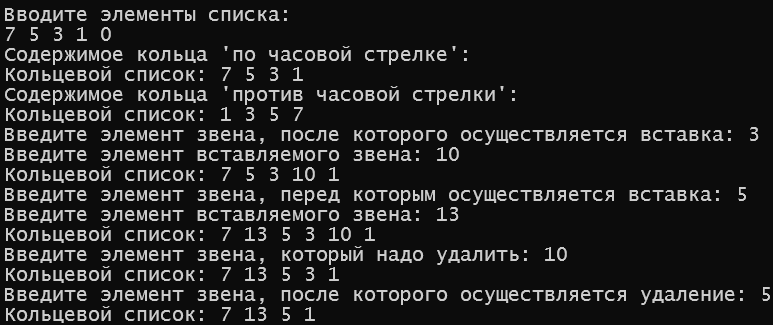
}

delete q; // очищаем память

delete nsp; // очищаем память

}

Результат выполнения:



1. **Тестирование кода “Стека, реализованного на основе списка”**

#include<iostream>

using namespace std;

struct node // структура узла

{

int elem; // информационное поле

node\* sled; // указатель на следующий элемент

};

class Spisok // класс списка

{

private: // приватная секция

node\* stk; // указатель на вершину стека

int klad; // информационное поле вершины стека

public:

Spisok() { // конструктор

stk = NULL; // устанавливаем указатель на вершину в NULL

}

int Set\_Stack() { // получение элемента вершины стека

return klad;

}

void POSTROENIE(); // прототип функции для построения стека

void VYVOD(); // прототип функции для вывода стека

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

void YDALENIE(); //прототип функции для удаления элемента из стека

void OCHISTKA(); // прототип функции для очистки стека

};

void main()

{

setlocale(LC\_ALL, "Rus");

Spisok A; // Создание объекта класса Spisok

int el, t; // буферные переменные

A.POSTROENIE();

A.VYVOD();

cout << "Введите вставляемый элемент: ";

cin >> el;

A.W\_S(el);

A.VYVOD();

cout << "Удалим элемент из стека.\n";

A.YDALENIE();

t = A.Set\_Stack();

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

A.VYVOD();

A.OCHISTKA();

cout << "\n";

system("PAUSE");

}

void Spisok::POSTROENIE() // функция построения стека

{

node\* t; // буферный указатель

int el; // буферная переменная

cout << "Вводите элементы стека: ";

cin >> el; // считываем новый элемент

while (el != 0) // пока не встретился 0(признак окончания ввода)

{

t = new (node); // выделяем память для нового узла

(\*t).elem = el; // записываем в информационное поле текущее значение

(\*t).sled = stk; // устанавливаем связи

stk = t; // вершиной стека становится новый элемент

cin >> el; // считываем следующий элемент

}

}

void Spisok::VYVOD() // функция для вывода элементов стека

{

node\* t; // буферный указатель

cout << "Содержимое стека: ";

t = stk; // ссылаем указатель t на вершину стека

while (t != NULL) // пока не дошли до последнего элемента

{

cout << (\*t).elem << " "; // выводим значение текущего

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

}

cout << endl;

}

void Spisok::W\_S(int el) // функция для добавления нового элемента

{

node\* q; // буферный указатель

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

(\*q).elem = el; // записываем в информационное поле текущее значение

(\*q).sled = stk; // устанавливаем связи

stk = q; // вершиной стека становится новый элемент

}

void Spisok::YDALENIE() // функция для удаления элемента

{

node\* q;// буферный указатель

if (stk == NULL) // если стек пуст

cout << "Стек пуст!\n";

else

{

klad = (\*stk).elem; // сохраняем значение вершины стека

q = stk; // устанавливаем q на вершину

stk = (\*stk).sled; // вершина становится следующим элементом

delete q; // освобождаем память

}

}

void Spisok::OCHISTKA() // функция очистки стека

{

node\* t, \* q; // буферные указатели

t = stk; // ссылаем t на вершину

if (t != NULL){ // пока не дошли до конца

q = (\*t).sled; // ссылаем q на следующий элемент

while (q != NULL){ // пока не конец

delete t; // освобождаем память

t = q; // переходим на следующий узел

q = (\*q).sled; // переходим на следующий узел

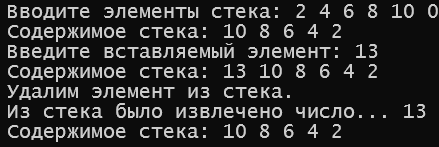
}

delete t; // очищаем память

}

}

Результат выполнения:



1. **Тестирование кода “Стека, реализованного на основе массива. Проверка скобочной последовательности”**

#include <iostream>

#include <string.h>

#include <stdio.h>

using namespace std;

const int MAXSIZE = 100; // максимальный размер стека

struct Stack // структура стека

{

char data[MAXSIZE]; // массив для элементов

int size; // текущий размер стека

};

void Push(Stack& S, char x) // функция для добавления элемента

{

if (S.size == MAXSIZE) { // если превышен размер

printf("Стек переполнен"); // вывод сообщения об этом

return;

}

// иначе

S.data[S.size] = x; // добавляем элемент в стек

S.size++; // увеличиваем размера стека

}

char Pop(Stack& S) // функция для извлечения элемента

{

if (S.size == 0) { // если стек пуст

printf("Стек пуст"); // выводим сообщение об этом

return char(255);

}

//иначе

S.size--; // уменьшаем размер стека

return S.data[S.size]; // возвращаем удаленный элемент

}

void main()

{

setlocale(LC\_ALL, "Rus");

char br1[3] = { '(', '[', '{' }; // массив открывающихся скобок

char br2[3] = { ')', ']', '}' }; // массив закрывающихся скобок

char s[80], upper;

int i, k, OK;

Stack S; //стек символов

printf("Введите выражение со скобками> ");

fgets(s, MAXSIZE, stdin); // считываем входную строку

S.size = 0; // изначально стек пуст

OK = 1; // и все хорошо

for (i = 0; OK && (s[i] != '\0'); i++) // в цикле проверяем строку

for (k = 0; k < 3; k++) { // проверяем скобки

if (s[i] == br1[k]) { // если нашли открывающуюся

Push(S, s[i]); // добавляем в стек

break;

}

if (s[i] == br2[k]) { // если нашли закрывающуюся

upper = Pop(S); // извлекаем первый элемент стека

if (upper != br1[k]) // если скобки разные

OK = 0; // то последовательность неверная

break; // выходим

}

}

if (OK && (S.size == 0)) // если все хорошо и стек пуст

printf("\nВыpажение пpавильное\n");

else // иначе

printf("\nВыpажение непpавильное \n");

system("PAUSE");

}

Результат выполнения:



