БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

**Динамические структуры данных. Списки. Тестирование кода**

**По:** Основы алгоритмизации и программирования

**Выполнил:** Корнелюк

Валентин Владимирович

1 курс 4 группа ПОИТ

**Преподаватель:**

Белодед Николай Иванович

**г. Минск**

2023 г.

Пример программы, осуществляющей удаление звена из однонаправленного списка с заглавным звеном.

#include<iostream>

using namespace std;

struct node

{

int elem; //элемент списка

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

};

class Spisok {

private:

node\* phead, \* Res; //указатель на заглавное звено и рабочий указатель Res для операций поиска и удаления.

public:

Spisok() { phead = new(node); Res = NULL; } // конструктор класса, создающий заглавное звено и инициализирующий рабочий указатель

~Spisok() { delete phead; } // деструктор класса, удаляющий заглавное звено

void POSTROENIE(); // метод построения списка

void VYVOD(); // метод вывода содержимого списка

node\* POISK(int); // метод поиска звена с заданным элементом

void YDALE(); // метод удаления звена, расположенного после заданного элемента

void YDALE1(); // метод удаления звена с заданным элементом

void OCHISTKA(); // метод очистки памяти, выделенной под список

};

void main()

/\*В функции `main()` создается объект класса `Spisok`, вызываются его методы для работы со списком, а затем происходит очистка памяти.\*/

{

setlocale(LC\_ALL, "Rus");

Spisok A; // создание объекта класса Spisok

int el; //переменная для ввода элементов списка

node\* Res\_Zn; //указатель на найденный элемент списка

A.POSTROENIE(); //вызов метода построения списка

A.VYVOD(); //вызов метода вывода списка на экран

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

cout << "осуществляется удаление:\n";

cin >> el; //ввод элемента звена

Res\_Zn = A.POISK(el); //поиск звена с заданным элементом в списке

if (Res\_Zn != NULL && (\*Res\_Zn).sled != NULL)

{

A.YDALE(); A.VYVOD(); //удаление звена и вывод списка на экран

}

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

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

cin >> el;

Res\_Zn = A.POISK(el);

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

{

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

}

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

A.OCHISTKA();

cout << "\n";

system("PAUSE");

}

void Spisok::POSTROENIE()

/\*Метод `POSTROENIE()` позволяет построить однонаправленный список, начиная с заглавного звена.

При этом происходит динамическое выделение памяти под новые элементы списка. \*/

//phead -указатель на заглавное звено

{

node\* t; // рабочий указатель для создания новых звеньев

int el; // переменная для ввода элементов списка

t = phead; (\*t).sled = NULL; // указатель указывает на начало списка, заглавное звено не содержит элементов

cout << "Вводите элементы звеньев списка: ";

cin >> el; // вводим первый элемент списка

while (el != 0) // пока не введен 0

{

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

t = (\*t).sled; (\*t).elem = el; (\*t).sled = NULL; // инициализируем новое звено и перемещаем указатель на него

cin >> el; // вводим следующий элемент списка

}

}

void Spisok::VYVOD()

/\*Метод `VYVOD()` выводит содержимое списка, начиная с первого элемента. Перемещение по списку осуществляется

с помощью указателя на текущий элемент (`t`), который изменяется на указатель на следующий элемент

при переходе к следующему звену.\*/

//phead - указатель на заглавное звено.

{

node\* t; //рабочий указатель на текущий элемент списка

t = phead; t = (\*t).sled; //устанавливаем указатель на первый элемент списка, ереходим на первый элемент списка

cout << "Список: ";

while (t != NULL) //цикл перебора всех элементов списка

{

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

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

}

}

node\* Spisok::POISK(int el)

/\*Метод `POISK()` осуществляет поиск звена с заданным элементом в списке, начиная с первого элемента.

Перемещение по списку осуществляется с помощью указателя на текущий элемент (`t`), который изменяется

на указатель на следующий элемент при переходе к следующему звену. Если звено с заданным элементом найдено,

его адрес сохраняется в переменной `Res`, иначе `Res` остается равным `NULL`.\*/

//списка. В противном случае Res содержит NULL.

{

node\* t; //рабочий указатель на текущий элемент списка

Res = NULL; t = phead; t = (\*t).sled; //обнуляем указатель на найденный элемент, //устанавливаем указатель на первый элемент списка, //переходим на первый элемент списка

while (t != NULL && Res == NULL) //цикл перебора всех элементов списка до нахождения нужного элемента или конца списка

if ((\*t).elem == el) Res = t; //если элемент найден, сохраняем его адрес в Res

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

return Res; //возвращаем указатель на найденный элемент списка

}

void Spisok::YDALE()

/\*Метод `YDALE()` удаляет звено, следующее за заданным элементом в списке.

Если звено не является последним, указатель на следующий элемент списка перенаправляется

на элемент, следующий за удаляемым, а затем удаляется удаляемый элемент. Если звено является

последним, выводится сообщение об ошибке.\*/

{

node\* q; //рабочий указатель на текущий элемент списка

q = (\*Res).sled; //указатель на следующий элемент после удаляемого звена

if (q != NULL)

{

//Если звено, после которого нужно удалять,

//не является последним, то:

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

}

else

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

}

void Spisok::YDALE1()

/\*Метод `YDALE1()` удаляет звено с заданным элементом из списка. Если звено существует, его элемент копируется

в заданное звено, указатель на следующий элемент списка перенаправляется на следующий элемент после удаляемого

звена, а затем удаляется удаляемое звено. Если звено не существует, указатель на предыдущий элемент списка

перенаправляется на `NULL`, удаляемое звено удаляется, а указатель на следующее звено после удаленного звена

устанавливается в `NULL`.\*/

{

node\* q, \* q1, \* q2;

q = (\*Res).sled; //указатель на следующий элемент после удаляемого звена

if (q != NULL) //если следующий элемент существует

{

(\*Res).elem = (\*q).elem; (\*Res).sled = (\*q).sled; //копируем элемент из следующего звена в заданное звено, перенаправляем указатель на следующий элемент списка,

delete q; //удаляем следующее звено

}

else //если следующего элемента нет (удаляемый элемент последний в списке)

{

q1 = phead; q2 = (\*q1).sled; //устанавливаем указатель на заглавное звено, //устанавливаем указатель на первый элемент списк

while (q2 != Res) //ищем предыдущий элемент перед удаляемым звеном

{

q1 = q2; q2 = (\*q2).sled;

}

(\*q1).sled = NULL; q2 = NULL; delete Res; //перенаправляем указатель на предыдущий элемент списка на NULL, устанавливаем указатель на следующее звено после удаленного звена в NULL, удаляем удаляемое звено

}

}

void Spisok::OCHISTKA()

/\*Метод `OCHISTKA()` очищает память, выделенную под список. Перемещение по списку осуществляется с помощью указателей

на текущий и следующий элементы списка (`q` и `q1`). При удалении текущего звена указатель `q` перемещается на следующий

элемент списка, а `q1` перемещается на следующее за ним звено.\*/

// phead - указатель на заглавное звено списка.

{

node\* q, \* q1;// Рабочие указатели.

q = phead;

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

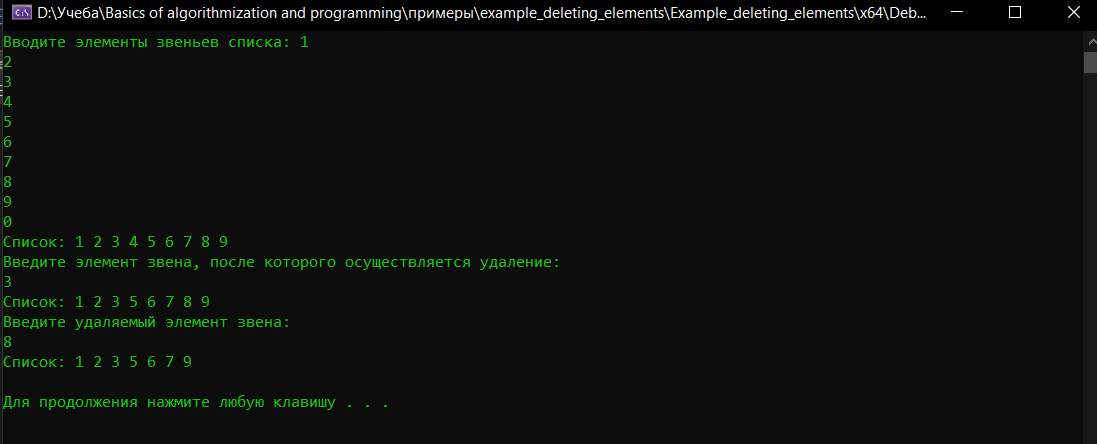
while (q1 != NULL) //цикл перебора всех элементов списка

{

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

}

}



Пример программы, демонстрирующей работу с “гирляндо-висюлечной” структурой.

#include<iostream>

using namespace std;

struct nodeVis

{

int elem; //Информационное поле звена висюльки.

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

};

struct nodeGir

{

int elem;//Информационное поле звена гирлянды.

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

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

};

class GirVis {

private:

nodeGir\* phead;//Голова гирлянды.

nodeVis\* pheadVis;//Голова висюльки.

void VisVyvod();

public:

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

~GirVis() { delete phead; }

nodeVis\* VisPostr();

nodeVis\* VisPoisk(int);

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

void VisVstav(nodeVis\*, int);

void Vis1Vstav(nodeVis\*, int);

void VisUdale(nodeVis\*);

void Vis1Udale(nodeVis\*);

void GirPostr();

void GirVyvod();

nodeGir\* GirPoisk(int);

void OCHISTKA();

void OCHISTKA1();

};

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");

}

void GirVis::OCHISTKA()

{

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

q = phead;

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

while (q1 != NULL) // Пока список не пустой.

{

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

pheadVis = (\*q).vniz;

OCHISTKA1(); //Очистка висюльки.

delete q;

}

}

void GirVis::OCHISTKA1()

{

nodeVis\* q, \* q1;

q = pheadVis;

//Удаляем звенья висюльки по очереди.

q1 = (\*q).vniz;

while (q1 != NULL)

{

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

delete q;

}

}

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;

}

}

// Функция VisPostr() строит однонаправленный список для висюльки,

// начиная с заглавного звена pheadVis.

nodeVis\* GirVis::VisPostr()

{

nodeVis\* t;

int el;

// Создаем заглавное звено списка.

pheadVis = new (nodeVis);

t = pheadVis;

(\*t).vniz = NULL;

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

cin >> el;

// Вводим элементы звеньев и добавляем их в список.

while (el != 0)

{

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

t = (\*t).vniz;

(\*t).elem = el;

(\*t).vniz = NULL;

cin >> el;

}

return pheadVis;

}

// Функция GirVyvod() выводит содержимое гирлянды,

// включая содержимое висюльки каждого звена.

void GirVis::GirVyvod()

{

nodeGir\* t;

// Начинаем с первого звена после заглавного.

t = phead;

t = (\*t).sled;

cout << "Гирлянда: ";

// Выводим содержимое каждого звена и его висюльки.

while (t != NULL)

{

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

pheadVis = (\*t).vniz;

VisVyvod();

t = (\*t).sled;

}

}

// Функция GirPoisk() осуществляет поиск элемента el в списке гирлянды.

// Начинается поиск с первого элемента после заглавного звена.

// Если элемент найден, возвращает адрес звена списка, содержащего элемент el.

// Если элемент не найден, возвращает NULL.

nodeGir\* GirVis::GirPoisk(int el) {

// создаем два указателя на звенья списка

nodeGir\* t, \* r;

r = NULL; // инициализируем r значением NULL

t = phead; // t указывает на заглавное звено гирлянды

t = (\*t).sled; // t теперь указывает на первый элемент гирлянды (после заглавного)

// проходим по всем элементам гирлянды

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

if ((\*t).elem == el) { // если нашли нужный элемент

r = t; // запоминаем адрес звена, содержащего элемент el

}

else {

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

}

}

return r; // возвращаем адрес звена, содержащего элемент el (или NULL, если элемент не найден)

}

// Функция VisVyvod() выводит содержимое однонаправленного списка для висюльки,

// заданного указателем pheadVis. Выводится содержимое висюльки в скобках.

void GirVis::VisVyvod() {

nodeVis\* t;

t = pheadVis; // t указывает на заглавное звено списка для висюльки

t = (\*t).vniz; // t теперь указывает на первый элемент списка для висюльки

cout << "(";

while (t != NULL) { // пока не достигли конца списка

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

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

}

cout << ")";

}

// Функция VisPoisk() осуществляет поиск элемента el в списке для висюльки,

// заданном указателем pheadVis. Если элемент найден, возвращает адрес звена списка,

// содержащего элемент el. Если элемент не найден, возвращает NULL.

nodeVis\* GirVis::VisPoisk(int el) {

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;

}

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;

}

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;

}

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";

}

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";

}

