#include <iostream>

using namespace std;

struct Node

{

int value;

Node\* p\_next; // Нодовые указатели нна след и пред элемент

Node\* p\_prev;

};

Node\* init(int value);

void print\_list(Node\* p\_first);

void addTo(Node\*p\_first);

void del\_pos(Node\* p\_cur, int index);

int get\_size(Node\* p\_first);

void del\_end(Node\* p\_cur);

void del2(Node\* p\_first, int new\_size);

int find(Node\* p\_first, int value);

int main()

{

setlocale(LC\_ALL, "rus");

cout << " Двусвязный список - добавление нового элемента, удаление" << endl;

cout << " элемента, удаление элемента и всей последующей части списка, поиск" << endl;

cout << "Двусвязный список - добавление нового элемента, удаление" << endl;

Node\* list = init(0); //создание первой ноды

print\_list(list); //вывод всего списка

cout << " Добавление однной ноды: ";

addTo(list);// добавление нового элемента

print\_list(list);

cout << " Добавление нескольких нод" << endl;

cout << " Сколько нод вы хотите добавить? ";

int colVo;

cin >> colVo;

for (int i = 0; i < colVo; i++)

{

cout << " Введите число: ";

addTo(list);

}

cout << " Текущий список: ";

print\_list(list);

cout << endl;

//удаление элемента

cout << " Введите индекс ноды, которую надо удалить: ";

int index;

cin >> index;

del\_pos(list, index);

print\_list(list);

//удаление элемента и последующей части списка

cout << " Введите индекс ноды: ";

cin >> index;

del2(list, index);

print\_list(list);

// поиск элемента по его значению

int value;

cout << " Введите значение элемента, индекс которого надо найти: ";

cin >> value;

index = find(list, value);

cout << " Индекс элемента со значением: " << value << " равен: " << index << endl;

return 0;

}

int find(Node\* p\_first, int value)

{

int size = get\_size(p\_first);

int index = 0;

while (true)

{

if ((p\_first->value == value) )

{

break;

}

index++;

p\_first = p\_first->p\_next;

}

return index;

}

void del\_end(Node\* p\_cur)

{

while (true)

{

if (p\_cur->p\_next == 0)

{

break;

}

p\_cur = p\_cur->p\_next;

}

(p\_cur->p\_prev)->p\_next = 0;

delete p\_cur;

}

void del2(Node\* p\_first, int new\_size)

{

if (new\_size <= 0)

{

cout << "Wrong new size!" << endl;

return;

}

int cur\_size = get\_size(p\_first);

int diff = new\_size - cur\_size;

if (diff < 0)

{

for (int i = 0; i < -diff; i++)

{

del\_end(p\_first);

}

}

}

int get\_size(Node\* p\_first)

{

int size = 1;

while (true)

{

if (p\_first->p\_next == 0)

{

break;

}

p\_first = p\_first->p\_next;

size++;

}

return size;

}

void del\_pos(Node\* p\_cur, int index)

{

int size = get\_size(p\_cur);

if ((index >= size) || (index < 0))

{

cout << "Out of range!" << endl;

return;

}

if (index == 0)

{

Node\* deletable = p\_cur->p\_next;

p\_cur->value = (p\_cur->p\_next)->value;

if ((p\_cur->p\_next)->p\_next != 0)

{

p\_cur->p\_next = (p\_cur->p\_next)->p\_next;

(p\_cur->p\_next)->p\_prev = p\_cur;

}

delete deletable;

return;

}

if (index == (size - 1))

{

while (true)

{

if (p\_cur->p\_next == 0)

{

break;

}

p\_cur = p\_cur->p\_next;

}

p\_cur = p\_cur->p\_prev;

delete p\_cur->p\_next;

p\_cur->p\_next = 0;

return;

}

int i = 1;

p\_cur = p\_cur->p\_next;

while (true)

{

if (i == index)

{

(p\_cur->p\_next)->p\_prev = p\_cur->p\_prev;

(p\_cur->p\_prev)->p\_next = p\_cur->p\_next;

delete p\_cur;

return;

}

p\_cur = p\_cur->p\_next;

i++;

}

}

void addTo(Node\* p\_cur)

{

int value;

cin >> value;

while (true)

{

if (p\_cur->p\_next == 0)

{

break;

}

p\_cur = p\_cur->p\_next;

}

Node\* part = new Node;

part->value = value;

part->p\_next = 0;

part->p\_prev = p\_cur;

p\_cur->p\_next = part;

}

Node\* init(int value) // создание первой ноды возвращает указатель на первую ячейку этой структуры

{

Node\* part = new Node; // new - выделяет память и возвращает указатель на парт -> получить доступ к полю

part->value = value; // указатели на след и пред равны 0

part->p\_next = 0;

part->p\_prev = 0;

return part;

}

void print\_list(Node\* p\_first)

{

cout << '{';

while (true)

{

cout << p\_first->value;

if (p\_first->p\_next == 0)

{

break;

}

else

{

cout << ", ";

}

p\_first = p\_first->p\_next;

}

cout << "}\n";

}