**7.2. Линейные списки**

Самый простой способ связать множество элементов — сделать так, чтобы каждый элемент содержал ссылку на следующий. Такой список называется *однонаправленным (односвязным).* Если добавить в каждый элемент вторую ссылку — на предыдущий элемент, получится *двунаправленный список (двусвязный),* если последний элемент связать указателем с первым, получится *кольцевой список.* Каждый элемент списка содержит *ключ,* идентифицирующий этот элемент. Ключ обычно бывает либо целым числом, либо строкой и является частью поля данных. В качестве ключа в процессе работы со списком могут выступать разные части поля данных. Например, если создается линейный список из записей, содержащих фамилию, год рождения, стаж работы и пол, любая часть записи может выступать в качестве ключа: при упорядочивании списка по алфавиту ключом будет фамилия, а при поиске, к примеру, ветеранов труда ключом будет стаж. Ключи разных элементов списка могут совпадать.

Над списками можно выполнять следующие *операции:*

• начальное формирование списка (создание первого элемента);

• добавление элемента в конец списка;

• чтение элемента с заданным ключом;

• вставка элемента в заданное место списка (до или после элемента с заданным ключом);

• удаление элемента с заданным ключом;

• упорядочивание списка по ключу.

Рассмотрим двунаправленный линейный список. Для формирования списка и работы с ним требуется иметь по крайней мере один указатель — на начало списка. Удобно завести еще один указатель — на конец списка. Для простоты допустим, что список состоит из целых чисел, то есть описание элемента списка выглядит следующим образом:

struct Node{

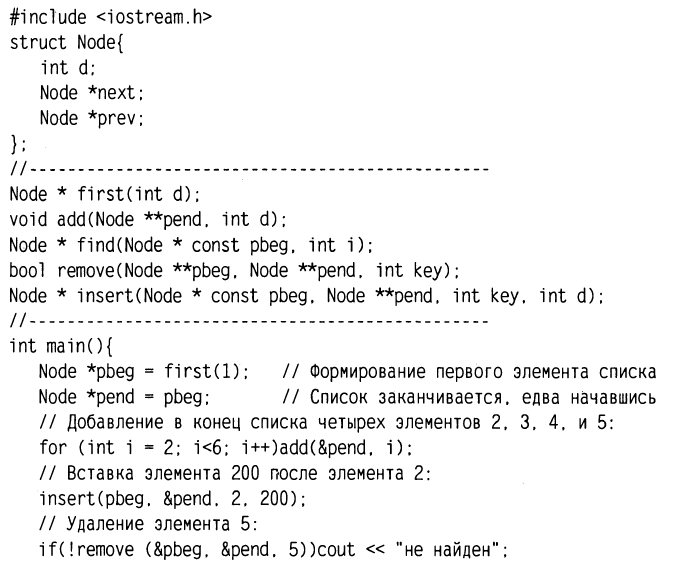
int d;

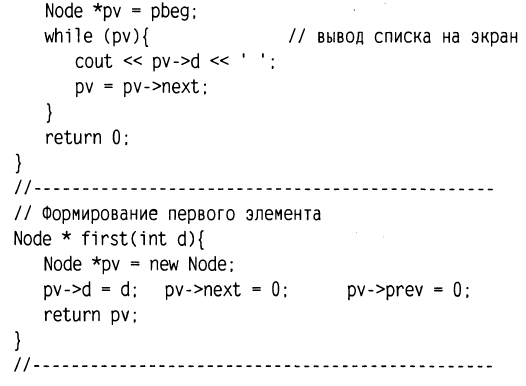
Node \*next;

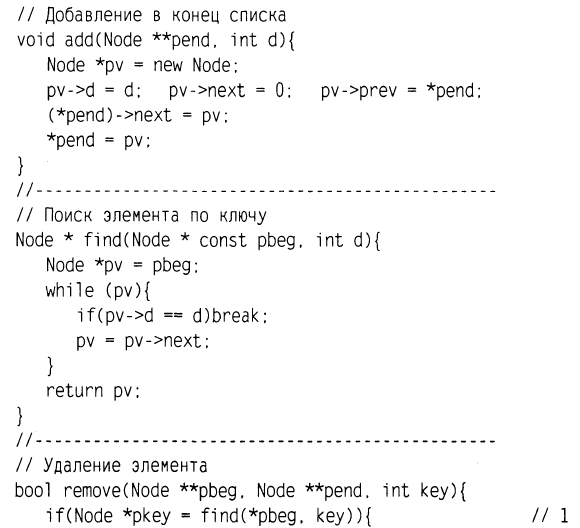
Node \*prev;

};

Ниже приведена программа, которая формирует список из 5 чисел, добавляет число в список, удаляет число из списка и выводит список на экран. Указатель на начало списка обозначен pbeg, на конец списка — pend, вспомогательные указатели — pv и ркеу.

****

****

****

if (pkey == \*pbeg){ // 2

\*pbeg = (\*pbeg)->next;

(\*pbeg)->prev =0;}

else if (pkey == \*pend){ // 3

\*pend = (\*pend)->prev;

(\*pend)->next =0;}

else{ // 4

(pkey->prev)->next = pkey->next;

(pkey->next)->prev = pkey->prev;}

delete pkey;

return true; // 5

}

return false; // 6

}

//------------------------------------------------

// Вставка элемента

Node \* insert (Node \*const pbeg, Node \*\*pend, int key, int d){

if(Node \*pkey = find(pbeg, key)){

Node \*pv = new Node;

pv->d = d;

// 1 - установление связи нового узла с последующим:

pv->next = pkey->next;

// 2 - установление связи нового узла с предыдущим:

pv->prev = pkey;

// 3 - установление связи предыдущего узла с новым:

pkey->next = pv;

// 4 - установление связи последующего узла с новым:

1f( ркеу != \*рend) (pv->next)->prev = pv;

// Обновление указателя на конец списка, если узел вставляется в конец:

else \*pend = pv;

return pv;

}

return 0;

}

Результат работы программы:

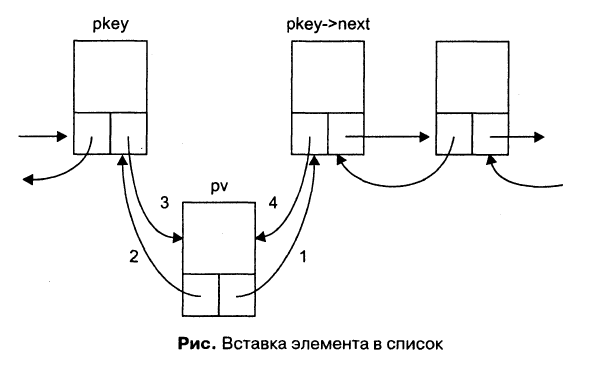
1 2 200 3 4

Все параметры, не изменяемые внутри функций, должны передаваться с модификатором const. Указатели, которые могут измениться (например, при удалении из списка последнего элемента указатель на конец списка требуется скорректировать), передаются по адресу.

Рассмотрим подробнее *функцию удаления элемента из списка* remove. Ее параметрами являются указатели на начало и конец списка и ключ элемента, подлежащего удалению. В строке 1 выделяется память под локальный указатель ркеу, которому присваивается результат выполнения функции нахождения элемента по ключу find. Эта функция возвращает указатель на элемент в случае успешного поиска и 0, если элемента с таким ключом в списке нет. Если ркеу получает ненулевое значение, условие в операторе if становится истинным (элемент существует), и управление передается оператору 2, если нет — выполняется возврат из функции со значением false (оператор 6).

Удаление из списка происходит по-разному в зависимости от того, находится элемент в начале списка, в середине или в конце. В операторе 2 проверяется, находится ли удаляемый элемент в начале списка — в этом случае следует скорректировать указатель pbeg на начало списка так, чтобы он указывал на следующий элемент в списке, адрес которого находится в поле next первого элемента. Новый начальный элемент списка должен иметь в своем поле указателя на предыдущий элемент значение 0. Если удаляемый элемент находится в конце списка (оператор 3), требуется сместить указатель pend конца списка на предыдущий элемент, адрес которого можно получить из поля prev последнего элемента. Кроме того, нужно обнулить для нового последнего элемента указатель на следующий элемент. Если удаление происходит из середины списка, то единственное, что надо сделать, — обеспечить двустороннюю связь предыдущего и последующего элементов. После корректиpoвки указателей память из-под элемента освобождается, и функция возвращает значение true.

Работа *функции вставки элемента* в список проиллюстрирована на рисунке. Номера около стрелок соответствуют номерам операторов в комментариях.



Сортировка связанного списка заключается в изменении связей между элементами. Алгоритм состоит в том, что исходный список просматривается, и каждый элемент вставляется в новый список на место, определяемое значением его ключа.

Ниже приведена *функция формирования упорядоченного списка* (предполагается, что первый элемент существует):

void add\_sort(Node \*\*pbeg, Node \*\*pend, 1nt d){

Node \*pv = new Node; // добавляемый элемент

pv->d = d;

Node \* pt = \*pbeg;

while (pt){ // просмотр списка

if (d < pt->d){ // занести перед текущим элементом (pt)

pv->next = pt;

if (pt == \*pbeg){ // в начало списка

pv->prev = 0;

\*pbeg = pv;}

else{ // в середину списка

(pt->prev)->next = pv;

pv->prev = pt->prev;}

pt->prev = pv;

return;

}

pt = pt->next;

}

pv->next = 0; // в конец списка

pv->prev = \*pend;

(\*pend)->next = pv;

\*pend = pv;

}