

Методические указания

Тематическое занятие 18

Двусвязные линейные списки.

Содержание

Двунаправленные связи в списке	1
<i>Основные определения</i>	<i>1</i>
<i>Организация связей</i>	<i>2</i>
Двусвязный список	2
<i>Указатели списка</i>	<i>2</i>
<i>Создание списка</i>	<i>2</i>
<i>Добавление элемента в список</i>	<i>3</i>
<i>Удаление элемента из списка</i>	<i>5</i>

Двунаправленные связи в списке

Основные определения

Динамический линейный список называется **двусвязным списком** (**двунаправленный список, double-linked list, two-way list**), если каждый его элемент с помощью двух указателей связывается с предыдущим и следующим элементами. При этом первый и последний элементы связаны только с одним элементом (следующим или предыдущим, соответственно).

Двусвязный линейный список становится **кольцевым**, если связать последний и первый элементы.

Двусторонняя очередь, которая также называется **дек (deque — double ended queue)**, – частный случай линейного двусвязного списка, элементы которого можно добавлять и удалять как в начало, так и в конец.

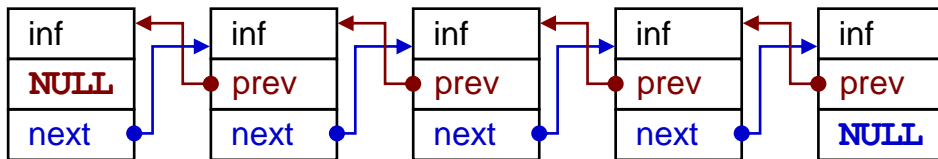
В общем случае элементы линейного двусвязного списка можно добавлять в любое место и удалять из любого места списка.

Организация связей

В простейшем случае элемент линейного двусвязного списка должен состоять из *трёх* полей: **информационного** (inf) и двух **указательных**.

Назовем указательные поля **prev** (от previous – предыдущий) и **next** (следующий). Также часто используются названия **left** и **right**.

Схематичное изображение двусвязного линейного списка:



Соответствующее объявление:

```
typedef struct elem {  
    int inf; /* inf - информационное поле */  
    struct elem *prev; /* prev - указатель на предыдущий эл-т */  
    struct elem *next; /* next - указатель на следующий эл-т */  
} Elem;
```

Двусвязный список

Указатели списка

Для создания двусвязного линейного списка и работы с ним необходимо иметь как минимум два указателя:

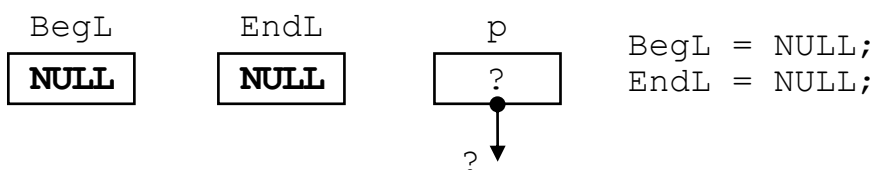
- на **начало** списка (назовем его **BegL**, от *begin of list*),
- на **конец** списка (назовем **EndL** – *end of list*).

Кроме того, потребуются дополнительные указатели: **p** – вспомогательный указатель, **pk** – указатель на некоторый *k*-й элемент списка.

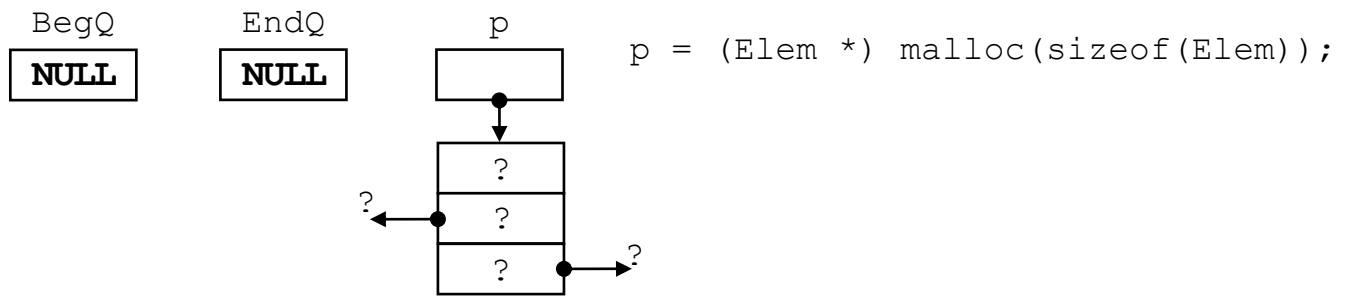
```
Elem *BegL;  
Elem *EndL;  
Elem *p, *pk;
```

Создание списка

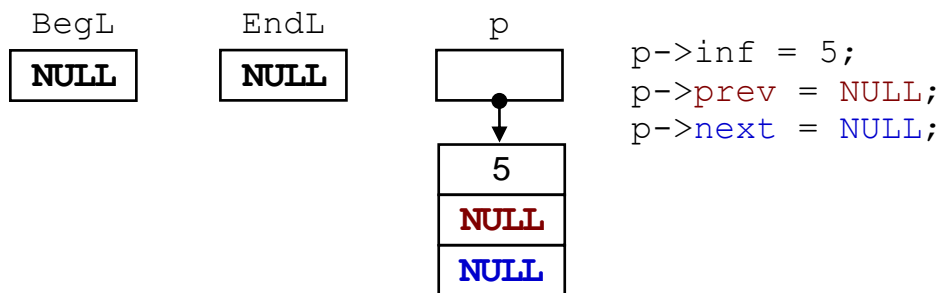
1. Исходное состояние.



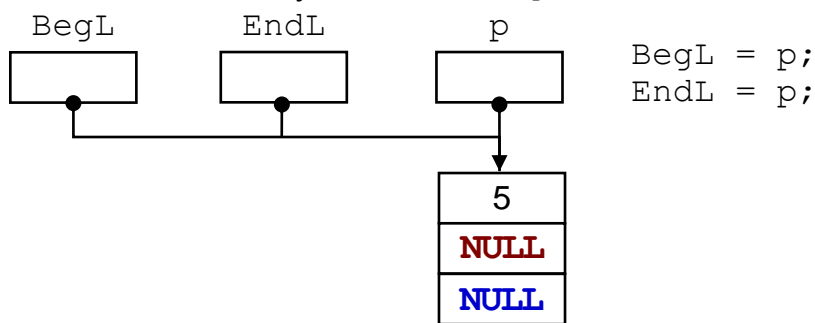
2. Выделение памяти под первый элемент списка и .



3. Занесение данных в первый элемент списка.



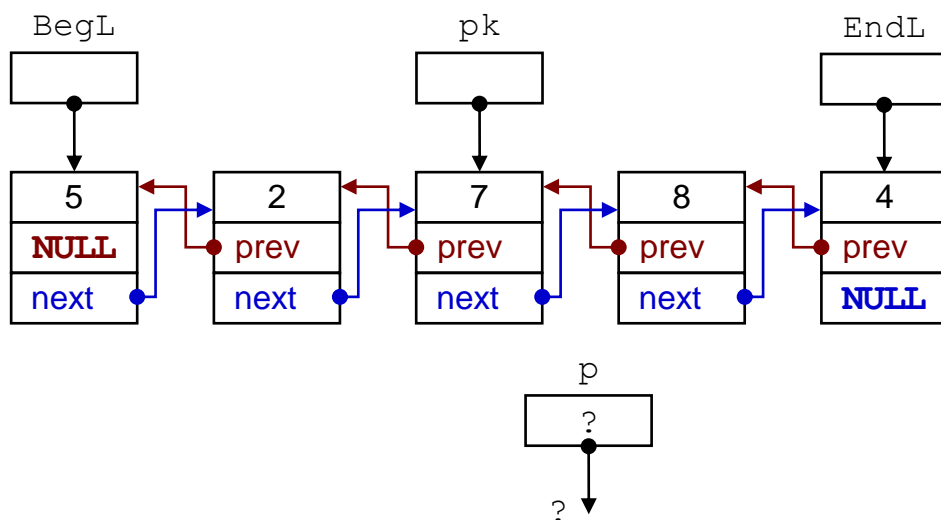
4. Установка указателей BegL и EndL на созданный первый элемент.



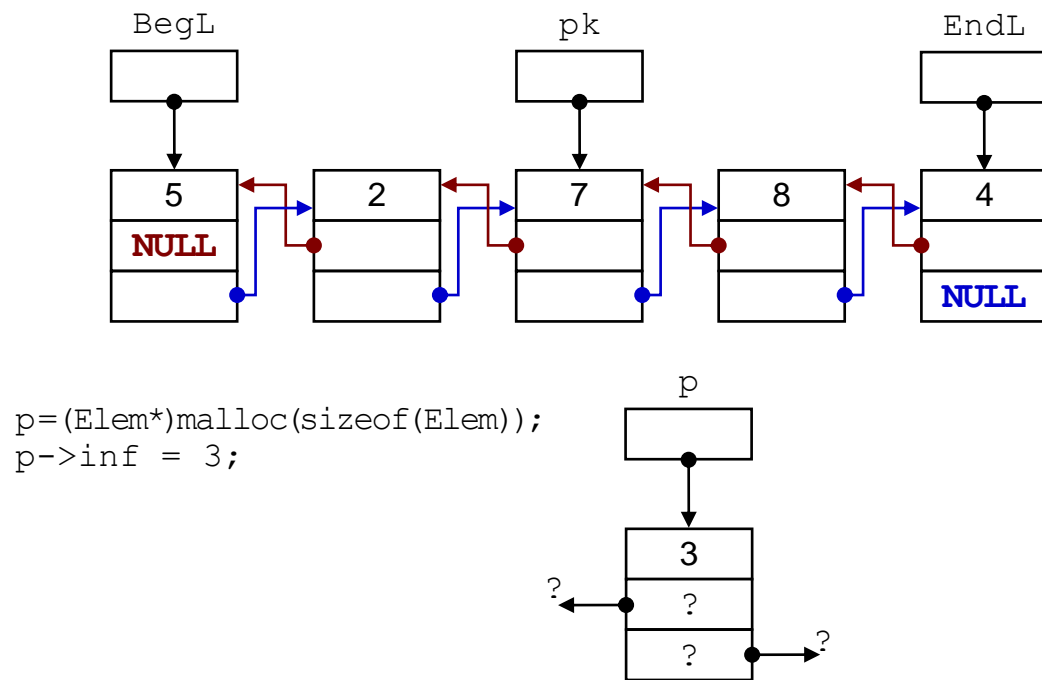
Добавление элемента в список

Цель – добавление нового элемента после k -го.

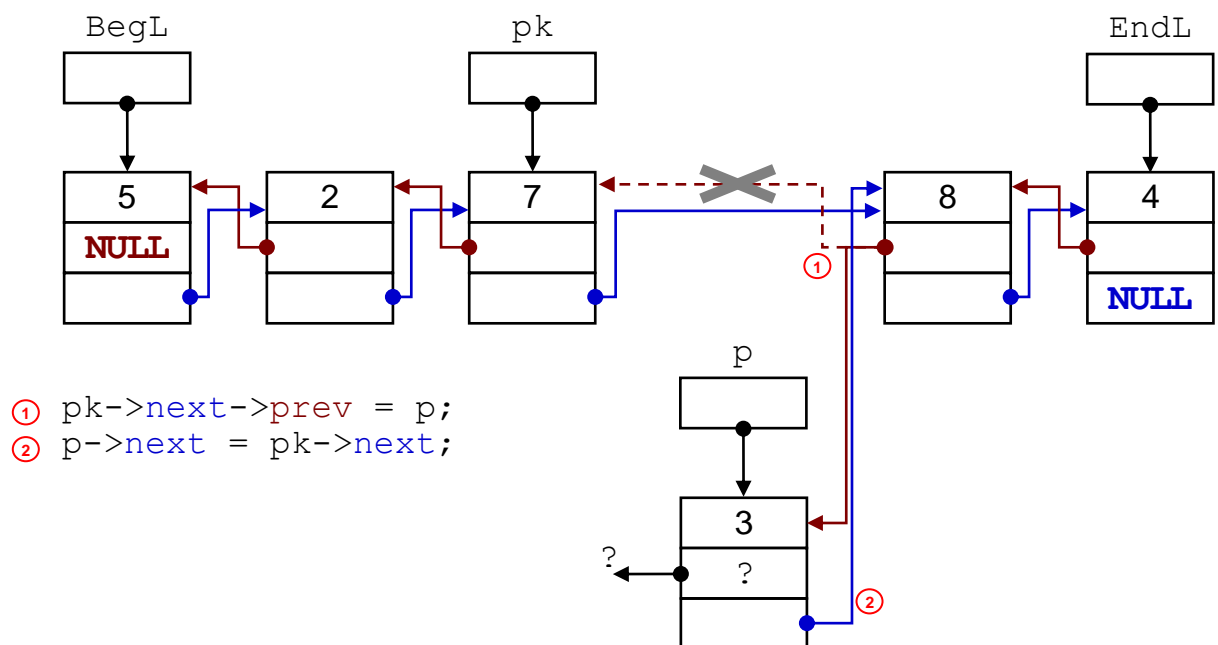
1. Исходное состояние.



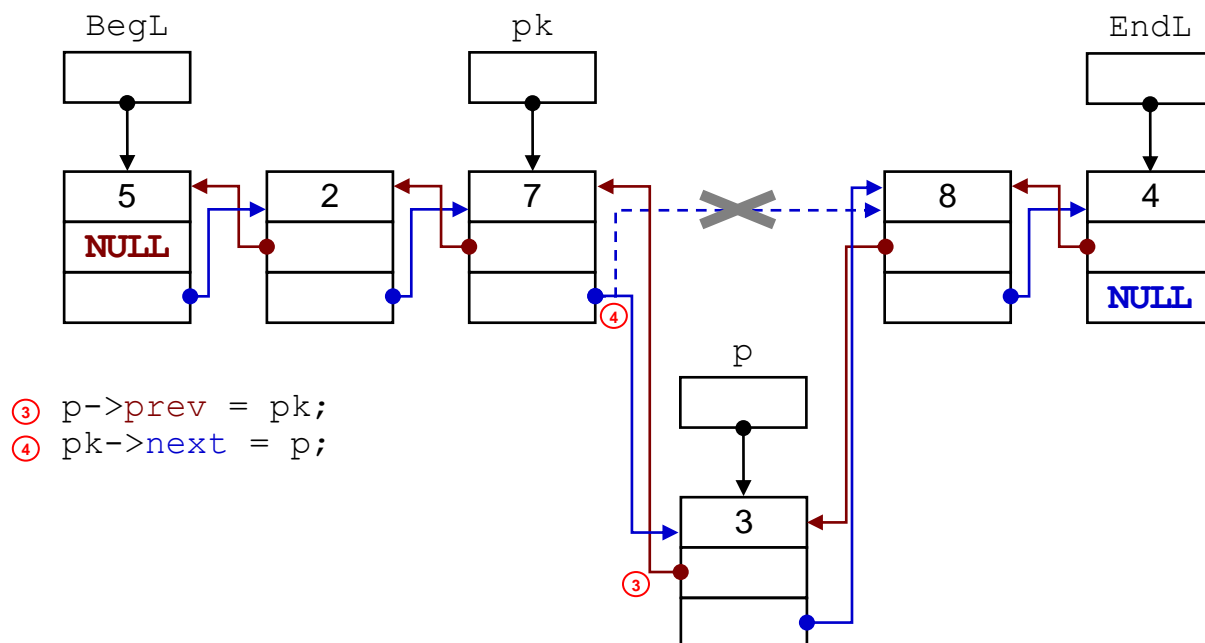
2. Выделение памяти под новый элемент списка и заполнение его информационного поля.



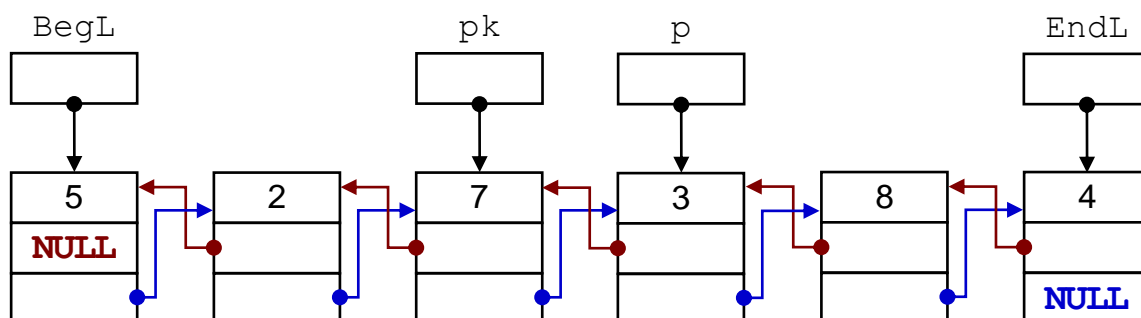
3. Связывание нового элемента с $(k+1)$ -м элементом.



4. Связывание нового элемента с k -м элементом.



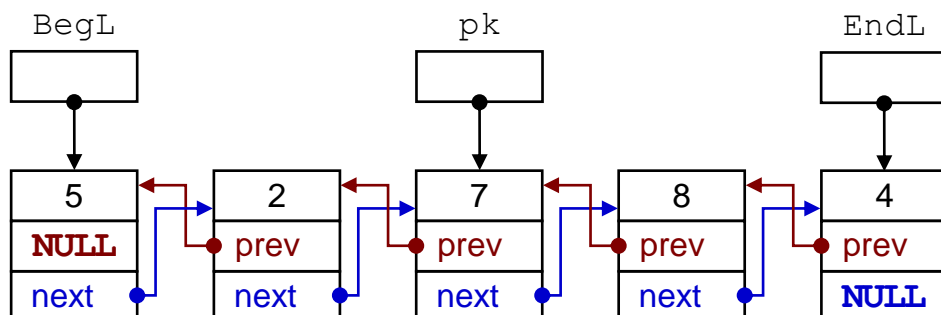
5. Конечное состояние.



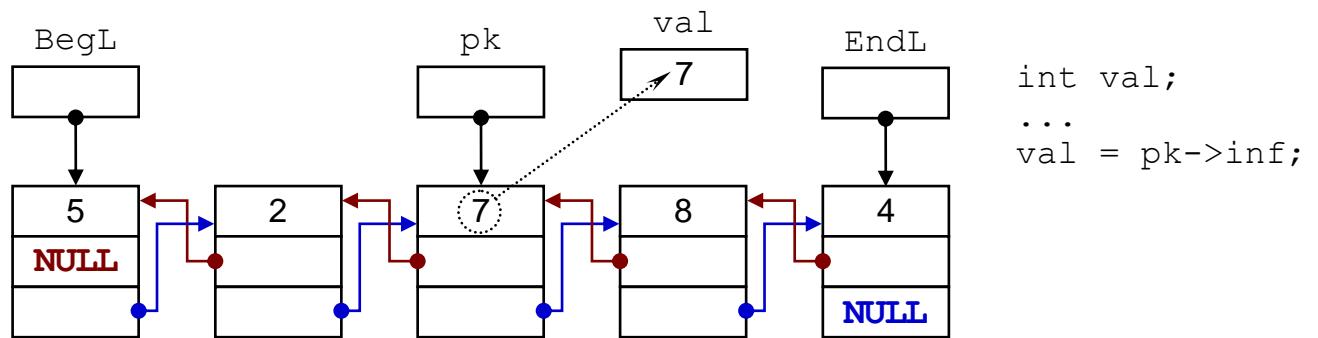
Удаление элемента из списка

Цель – удаление k -го элемента.

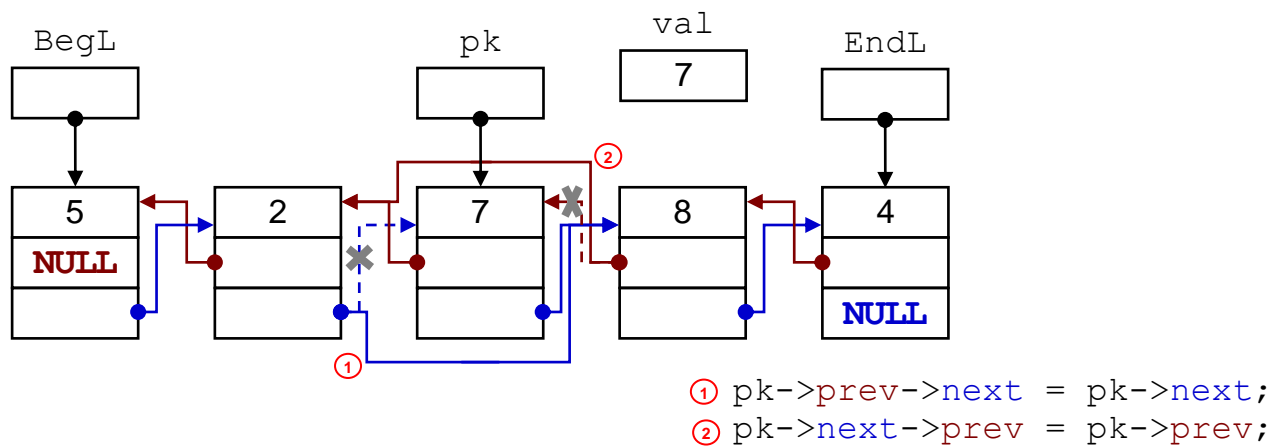
1. Исходное состояние.



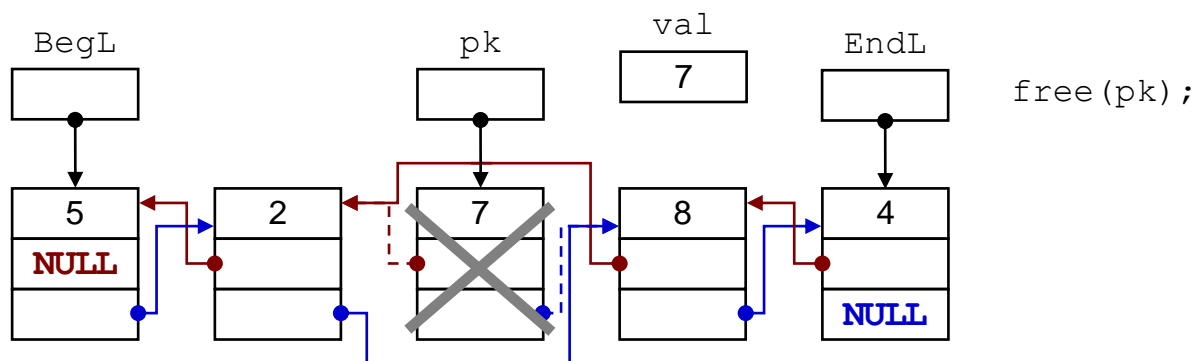
2. Извлечение информации из удаляемого k -го элемента в переменную `val`.



3. Связывание $(k-1)$ -го элемента с $(k+1)$ -м элементом.



4. Освобождение памяти удаляемого k -го элемента.



5. Конечное состояние.

