НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»
Кафедра информатики и процессов управления (№17)

Дисциплина «Информатика» (основной уровень), 2-й курс, 3-й семестр.

Методические указания

Тематическое занятие 20

Двусвязные линейные списки.

Содержание

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.

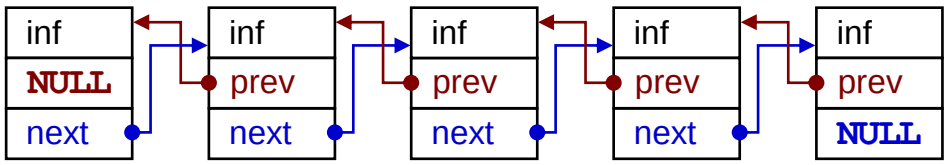
В общем случае элементы линейного двусвязного списка можно добавлять в любое место и удалять из любого места списка.

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.

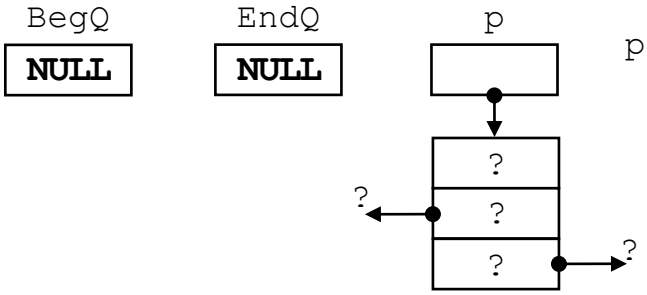
```
Elem *BegL;
Elem *EndL;
Elem *p, *pk;
```

Создание списка

1. Исходное состояние.


```
BegL = NULL;
EndL = NULL;
```

2. Выделение памяти под первый элемент списка и .

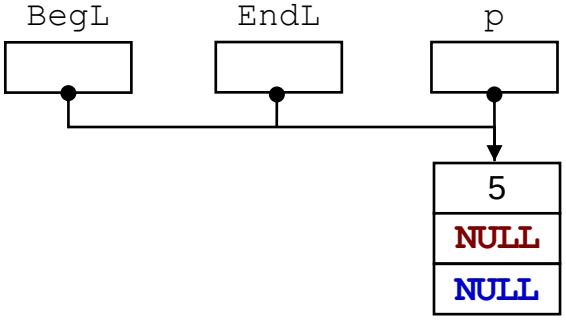



```
p = (Elem *) malloc(sizeof(Elem));
```

3. Занесение данных в первый элемент списка.


```
p->inf = 5;
p->prev = NULL;
p->next = NULL;
```

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

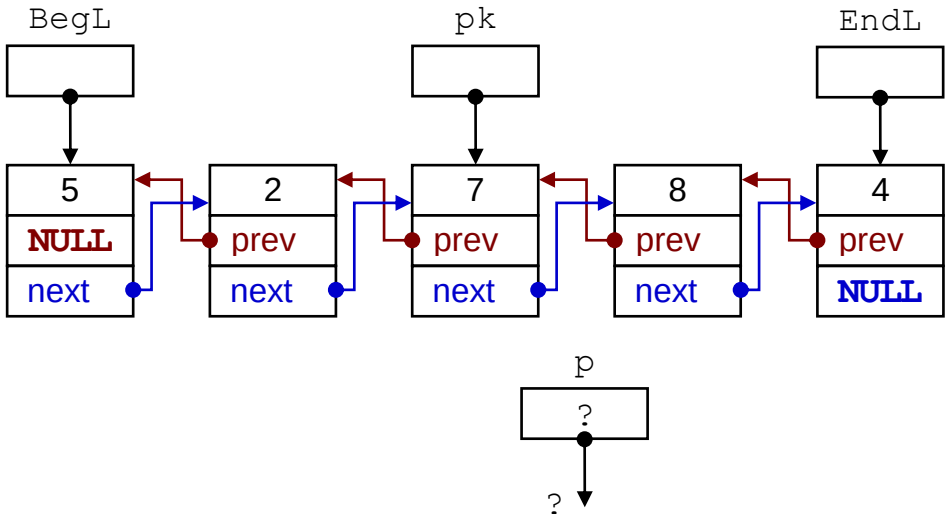



```
BegL = p;
EndL = p;
```

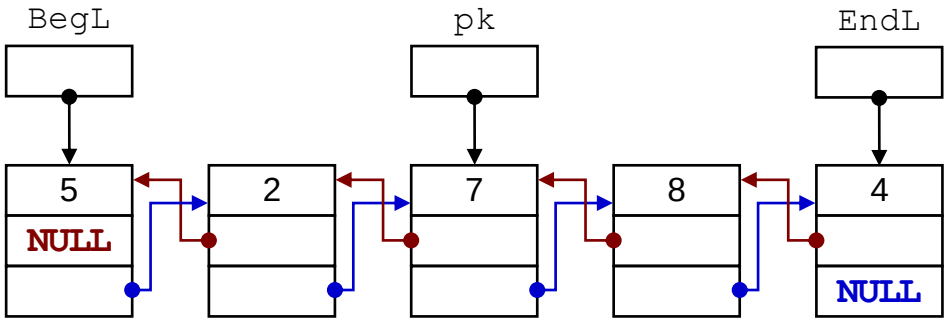
Добавление элемента в список

Цель – добавление нового элемента после k-го.

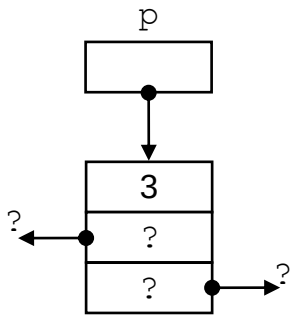
1. Исходное состояние.



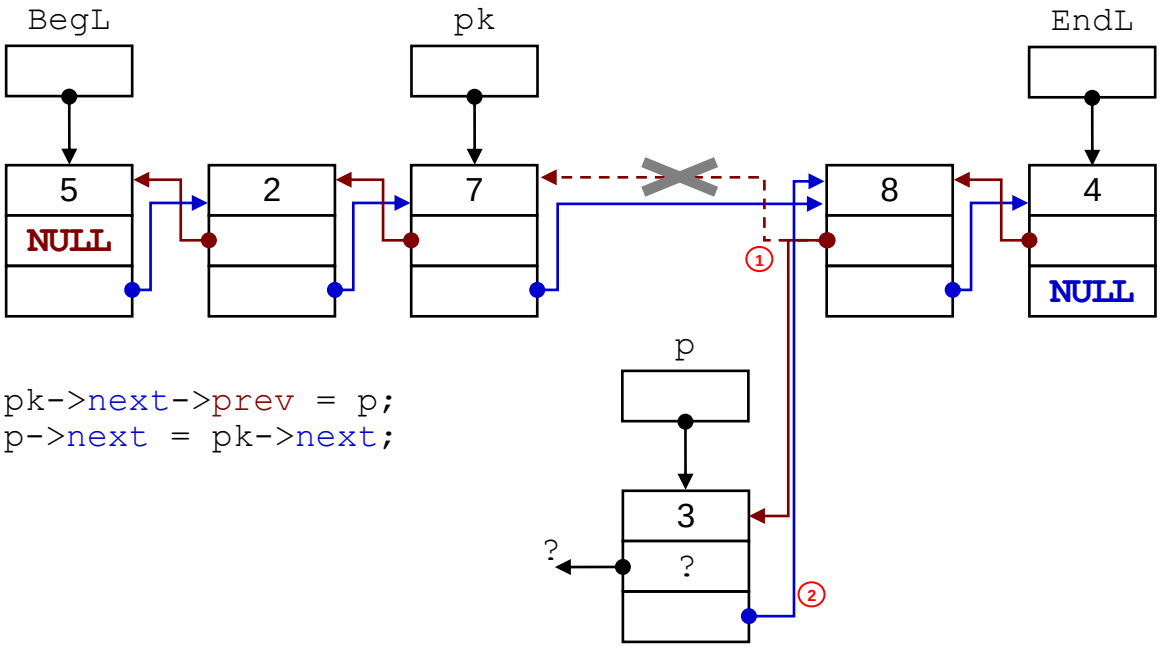
2. Выделение памяти под новый элемент списка и заполнение его информационного поля.




```
p=(Elem*)malloc(sizeof(Elem));
p->inf = 3;
```

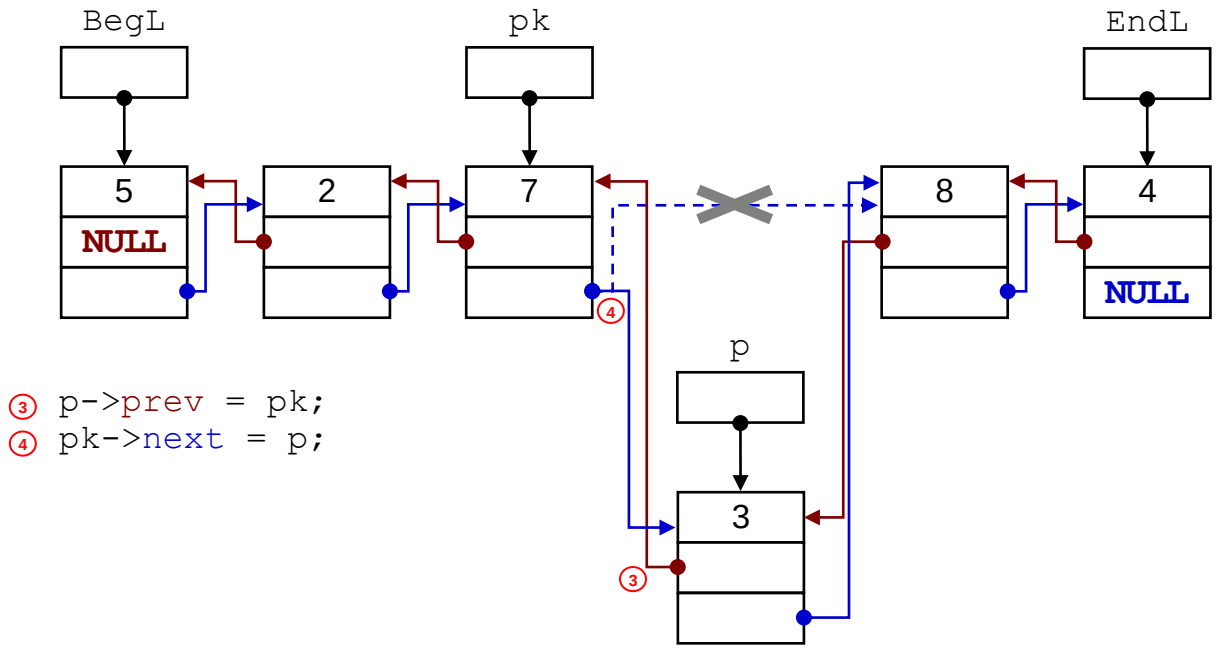


3. Связывание нового элемента с $(k+1)$-м элементом.

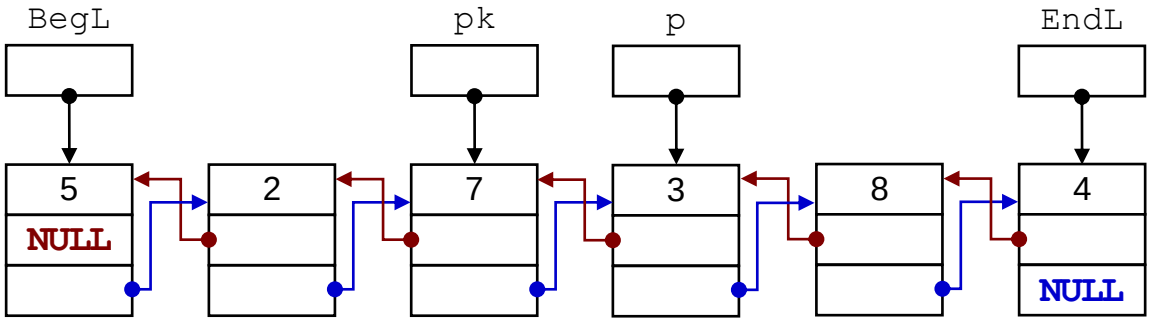



```
① pk->next->prev = p;
② p->next = pk->next;
```

4. Связывание нового элемента с k-м элементом.



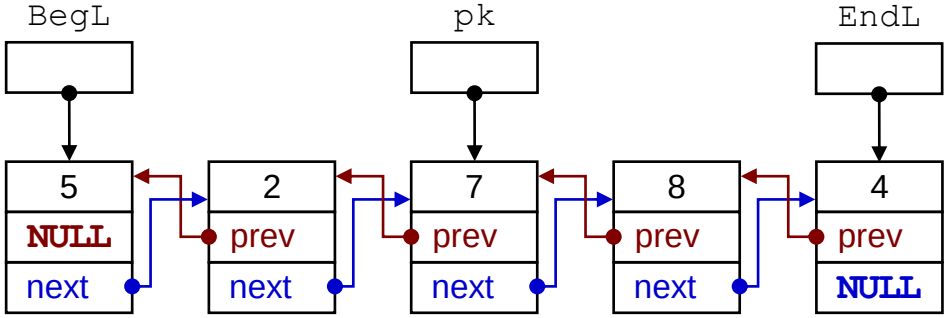
5. Конечное состояние.



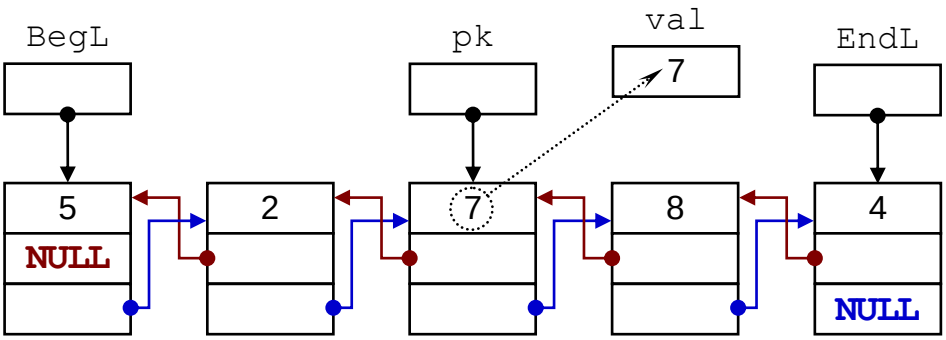
Удаление элемента из списка

Цель – удаление k-го элемента.

1. Исходное состояние.

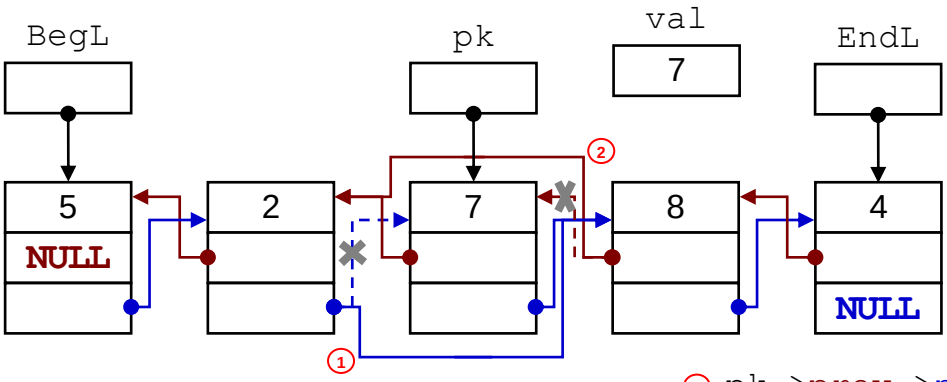


2. Извлечение информации из удаляемого k-го элемента в переменную `val`.



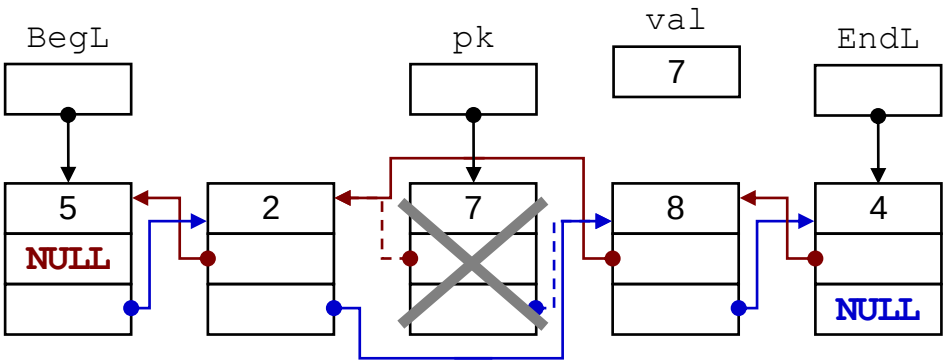

```
int val;
...
val = pk->inf;
```

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




```
① pk->prev->next = pk->next;
② pk->next->prev = pk->prev;
```

4. Освобождение памяти удаляемого k-го элемента.




```
free(pk);
```

5. Конечное состояние.