

Лекция 4

**Динамические структуры данных.
Списки.**

Динамические структуры данных

Если до начала работы с данными невозможно определить, сколько памяти потребуется для их хранения, память выделяется по мере необходимости отдельными блоками, связанными друг с другом при помощи указателей. Такой способ организации данных называется *динамическими структурами данных*, поскольку их размер изменяется во время выполнения программы.

Из динамических структур в программах чаще всего используются *линейные списки, стеки, очереди и бинарные деревья*. Они различаются способами связи элементов и допустимыми операциями над ними.

Динамическая структура может занимать *несмежные* участки оперативной памяти. В процессе работы программы элементы структуры могут по мере необходимости добавляться и удаляться.

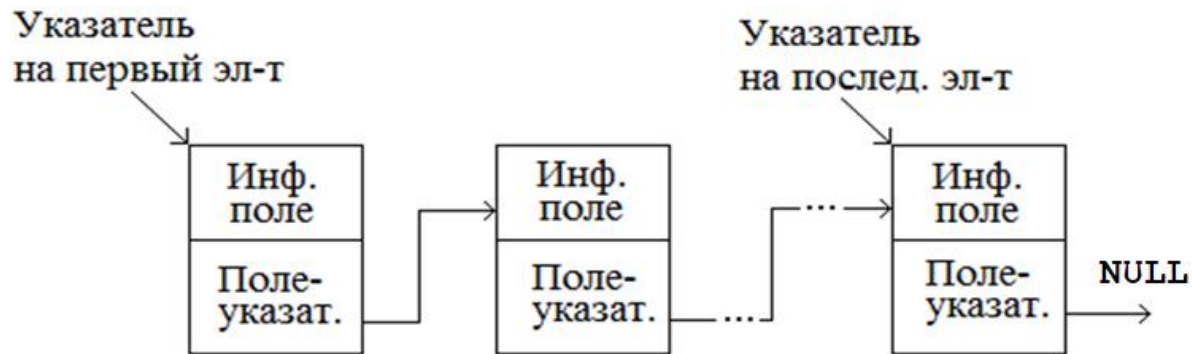
Линейный список

Связным линейным списком называется *структура данных*, представляющая собой последовательность однотипных элементов, в которой порядок следования элементов задается путем отсылок, т. е. каждый элемент списка содержит указатель на следующий элемент (или предыдущий).

Доступ к первому элементу связного списка выполняется с помощью специального указателя – указателя на **вершину (голову)** списка. Последний элемент списка имеет ссылку на **NULL**.

Описание элемента списка

Графическое представление односвязного линейного списка:



Для задания списковых структур необходимо *определить элемент списка в виде структуры*, в состав которой входит информационное поле и поле-указатель на следующий элемент (этого же типа).

```
struct my_list {  
    <тип инф. поля> inf;           // инф. поле  
    struct my_list* next;         // поле-указатель на след. элемент  
};
```

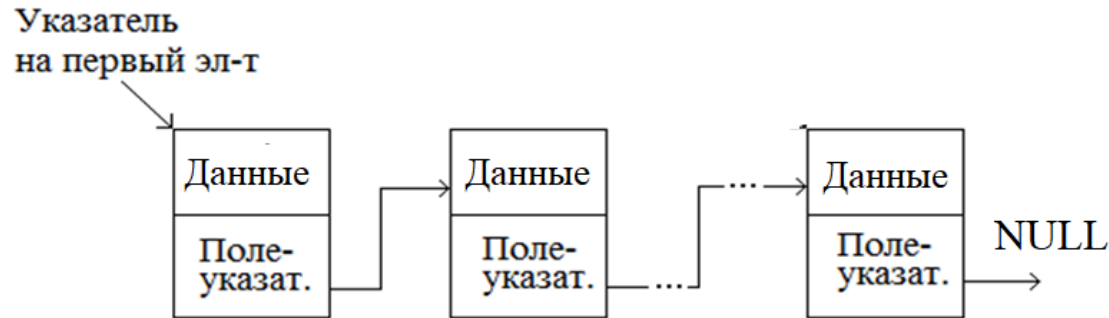
Операции над линейными списками

Над связными линейными списками выполняют такие операции:

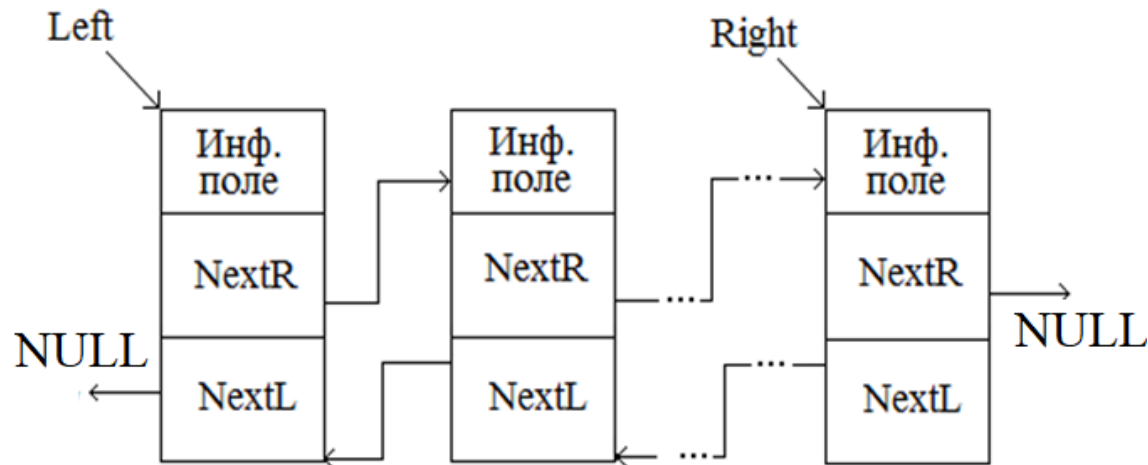
- добавление элемента в начало списка;
- добавление элемента в конец списка;
- добавление элемента между двумя элементами списка;
- удаление заданного элемента списка.

Виды связанных линейных списков

Односвязным линейным списком называют список, в котором предыдущий элемент ссылается на следующий.

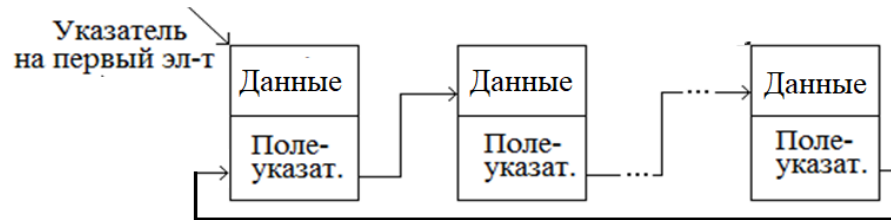


Двусвязный линейный список — это список, в котором предыдущий элемент ссылается на следующий, а следующий — на предыдущий.

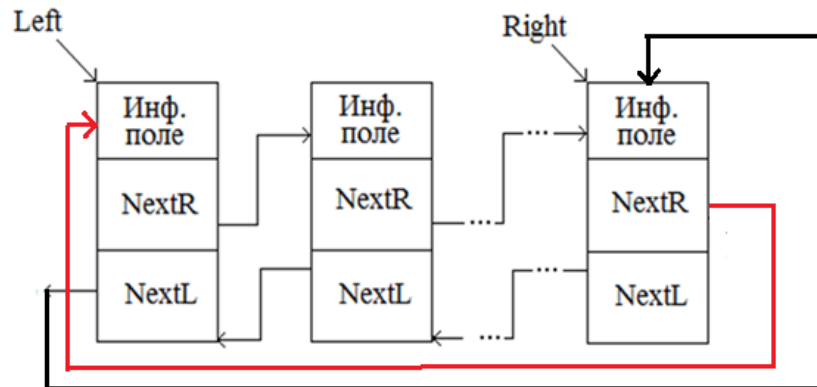


Виды связанных линейных списков

Односвязный циклический список — это односвязный линейный список, в котором последний элемент ссылается на первый.



Двусвязный циклический список — это двусвязный линейный список, в котором последний элемент ссылается на первый, а первый — на последний.



Стек — это односвязный список, в котором компоненты добавляются и удаляются только со стороны вершины списка.

Очередь — это односвязный список, в котором компоненты добавляются в конец списка, а удаляются со стороны вершины списка.

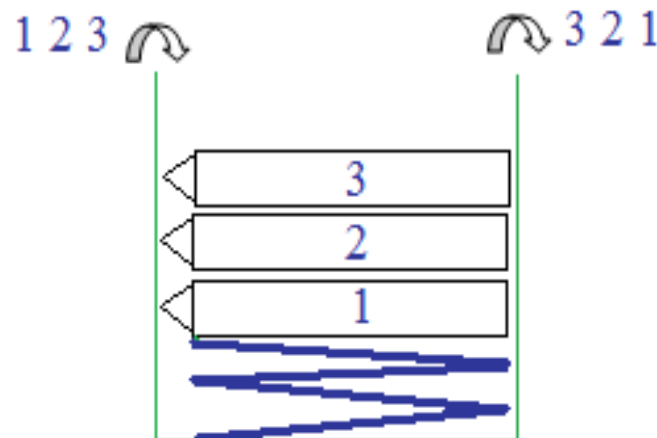
Стек

Стек — это одна из разновидностей линейного связного списка, доступ к элементам которого возможен только через его начало. Первый элемент стека называют **вершиной стека**.

Стек функционирует на основе механизма **LIFO – Last In First Out**.

Добавление элемента в стек и извлечение элемента из стека выполняют со стороны его вершины.

Этот механизм часто называют механизмом **магазинной памяти**.



При работе со стеком **используют указатель на его вершину (Top)** и вспомогательный указатель на элемент стека (**Temp**).

Очередь

Очередь – это одна из разновидностей линейного связного списка, добавление элементов в который выполняется со стороны хвоста, а удаление – со стороны головы.

Очередь функционирует на основе механизма ***FIFO – First In First Out***.

Для работы с очередью необходимо иметь **указатель на голову очереди (Left)**, **указатель на хвост очереди (Right)** и вспомогательный указатель на элемент очереди (**Temp**).

Рассмотрим программу, выполняющую организацию списка, добавление элемента в список, удаление элемента из списка.

В примере элементами списка будет структура, содержащая сведения о человеке. Каждая запись содержит следующие поля: номер, имя, возраст.

Пример программы обработки списков

Рассмотрим на примере.

```
struct data
```

```
{  int num;  
   char *name;  
   int age;  
};
```

```
struct list
```

```
{ struct data inf;  
  struct list *next;  
};
```

4	Петя	10
56	Саша	12
68	Маша	11

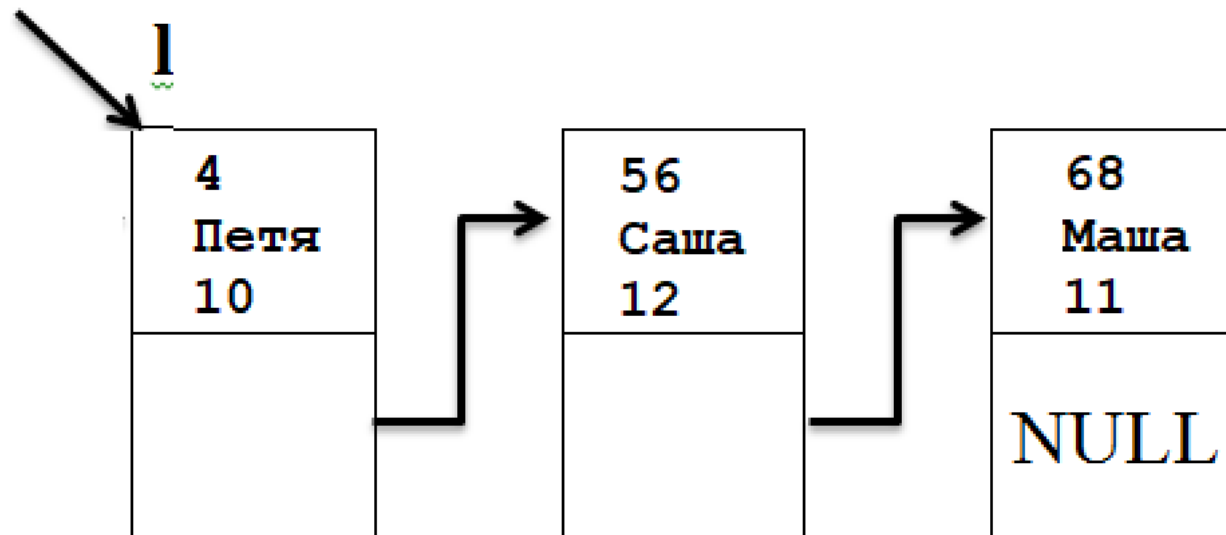
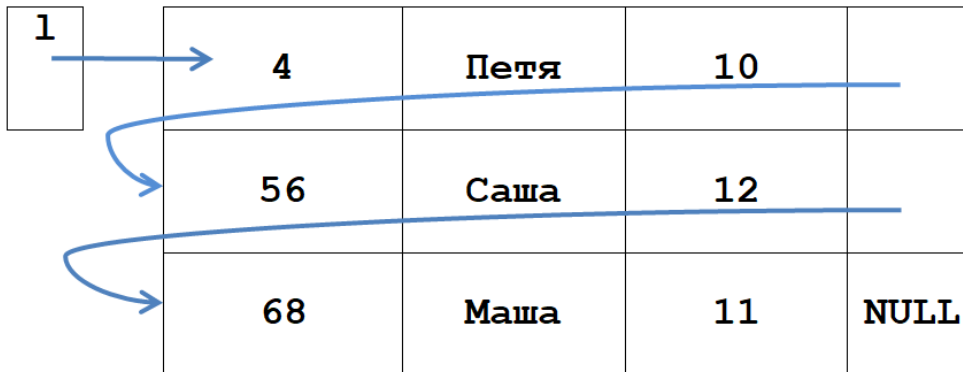


```
struct list *l,*r,*t; //локальные переменные main()
```

```
l=r=NULL; //инициализация в main()
```

Пример программы обработки списков

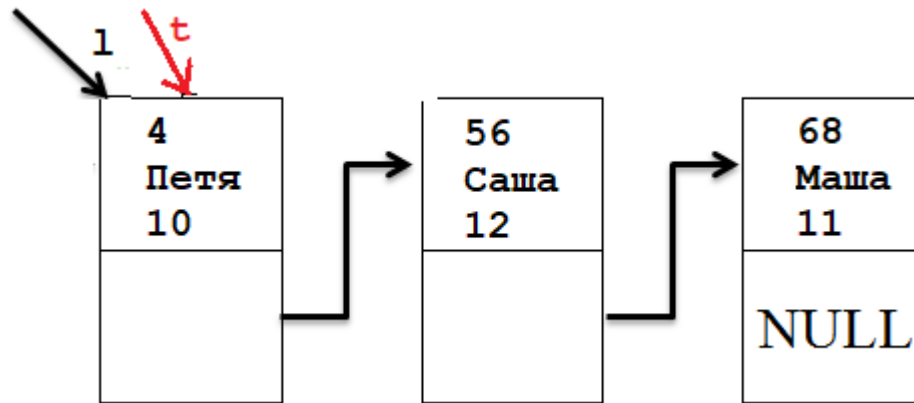
1	4	Петя	10	
	56	Саша	12	
	68	Маша	11	NULL



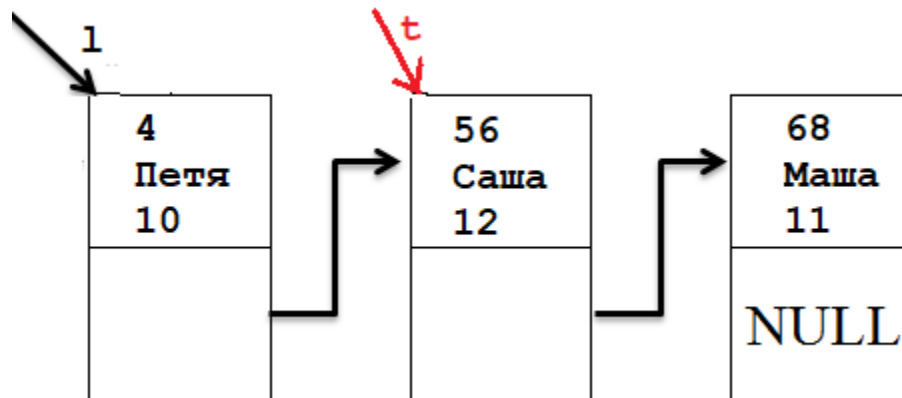
Печать списка

Рассмотрим как вывести на экран такую структуру, алгоритм:

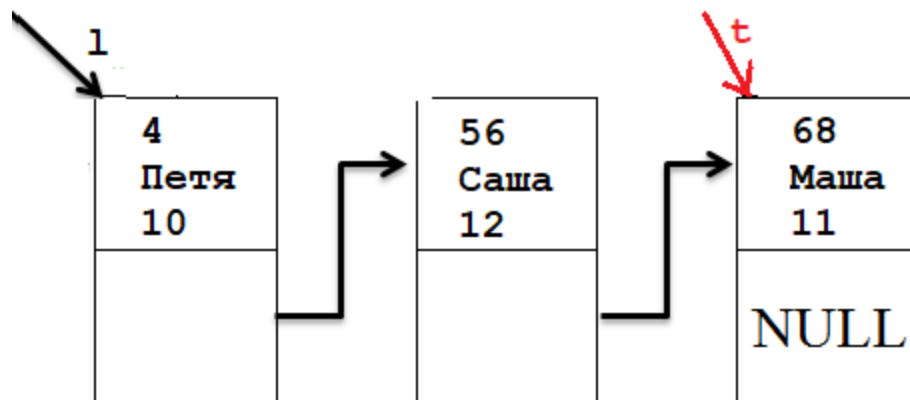
- **t** присвоить адрес первого элемента;
- вывести значение информационного поля **t->inf**;
- **t** переставить (**t** присвоить адрес следующего элемента **t->next**);
- повторить вывод и перестановку до тех пор, пока не **t != NULL**.



Печать списка

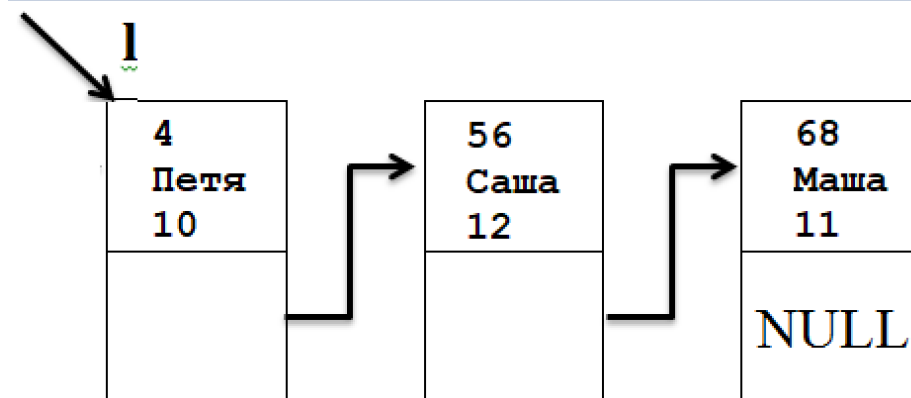


Печать списка



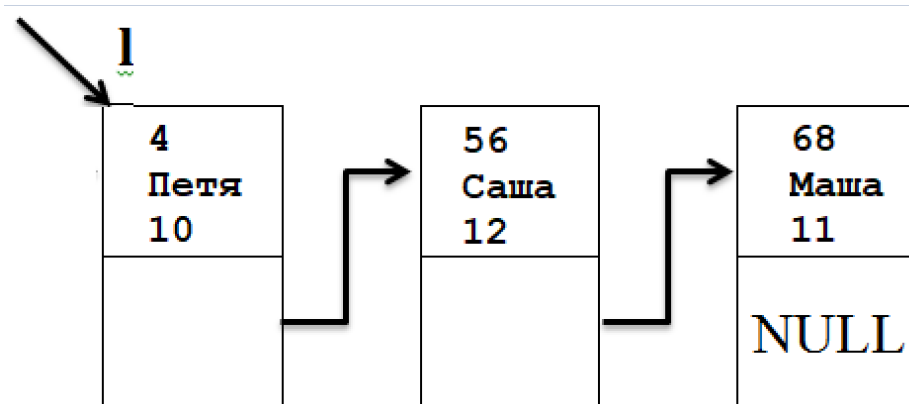
Печать списка (1 способ)

```
void print_list(struct list* l) {  
    struct list* t;  
    if (l==NULL){printf("список пуст");return;}  
    printf("| Номер записи | имя | возраст |\n");  
  
    t=l;                                //запомнить адрес первого  
    while (t!=NULL) {                  //пока не конец списка  
        printf("|%10d|%7s|%10d|\n", t->inf.num,    //печать  
            t->inf.name,t->inf.age) ;  
        t=t->next;                      //переставить указатель  
    }  
    printf("_____\n");  
    return;  
}
```



Печать списка (2 способ)

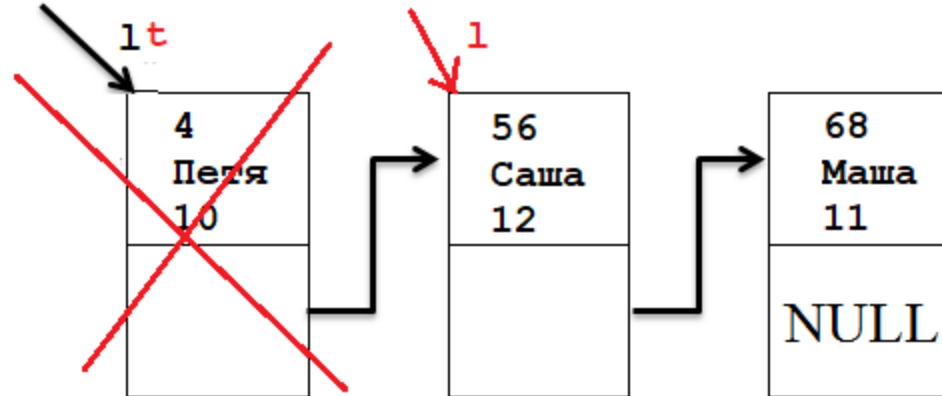
```
void print_list(struct list* l) {  
    struct list *t;  
    if (l==NULL){printf("список пуст");return;}  
    printf("| Номер записи | имя | возраст |\n");  
  
    for(t=l; t!=NULL; t=t->next)  
        printf("|%d|%s|%d|\n", t->inf.num, t->inf.name,  
                t->inf.age);  
  
    printf("_____\n");  
    return;  
}
```



Удаление первого элемента

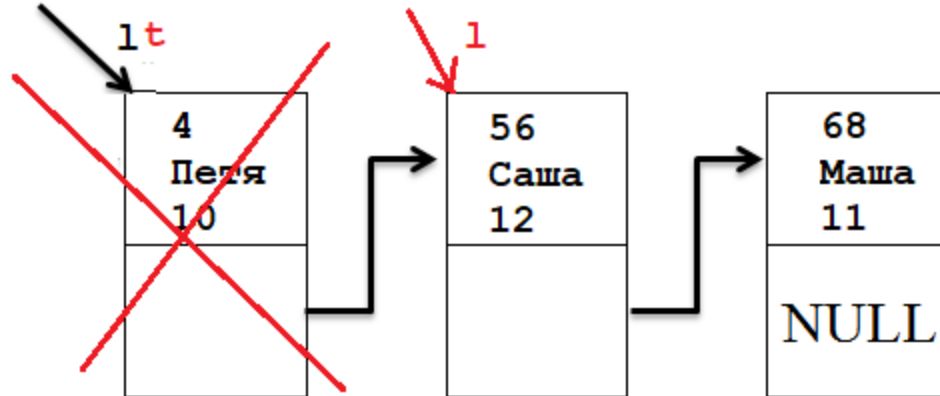
Алгоритм:

- **t** присвоить адрес первого элемента;
- **1** переставить (в **1** запомнить адрес второго элемента, т.к. после удаления второй элемент станет первым);
- удалить первый элемент (освободить память, на которую указывает **t**).



Удаление первого элемента

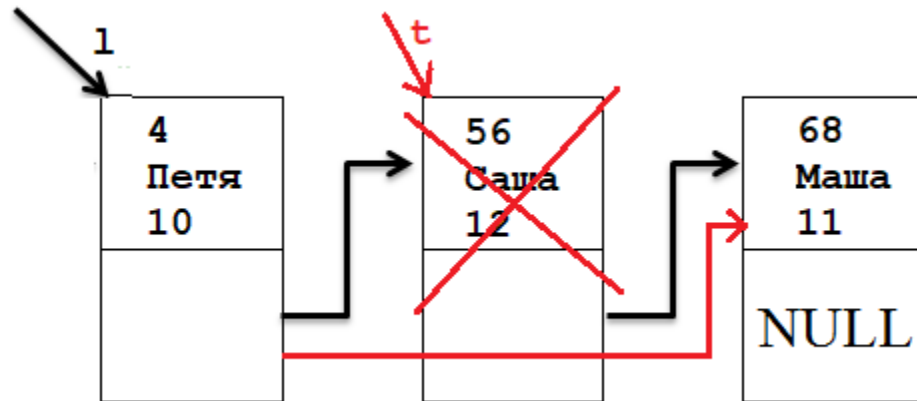
```
struct list* del_first(struct list* l) {  
    struct list* t;  
  
    t=l;  
    l=t->next;           // l=l->next;  
    free(t);  
    return l;  
}
```



Удаление второго элемента

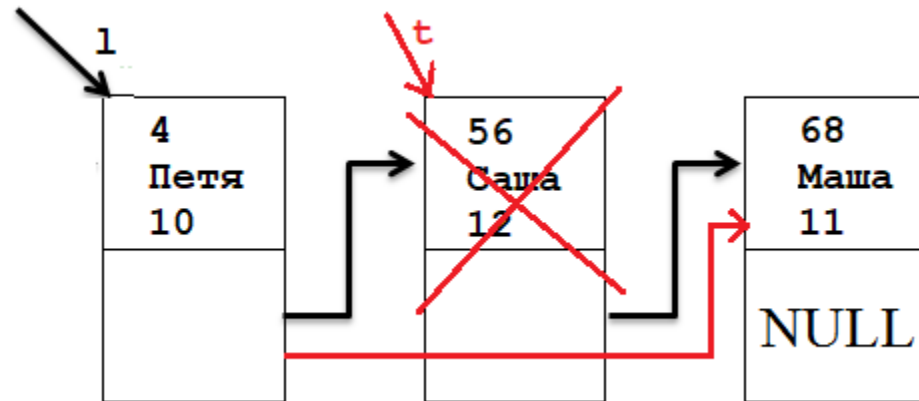
Алгоритм:

- **t** присвоить адрес второго элемента;
- расставить указатели (следующим за первым станет третий элемент);
- удалить второй элемент (освободить память, на которую указывает **t**).



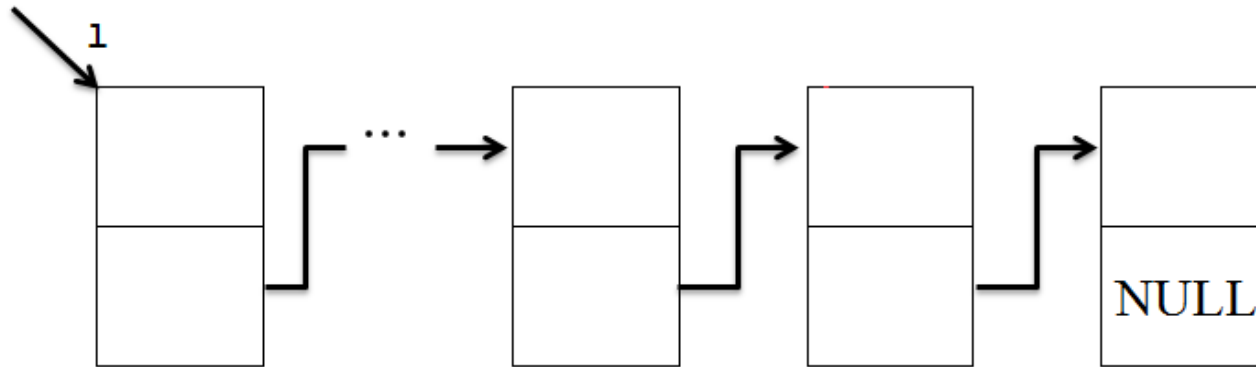
Удаление второго элемента

```
struct list* del_second(struct list* l) {  
    struct list* t;  
  
    t=l->next;  
    l->next=t->next;    //l->next=l->next->next;  
    free(t);  
    return l;  
}
```

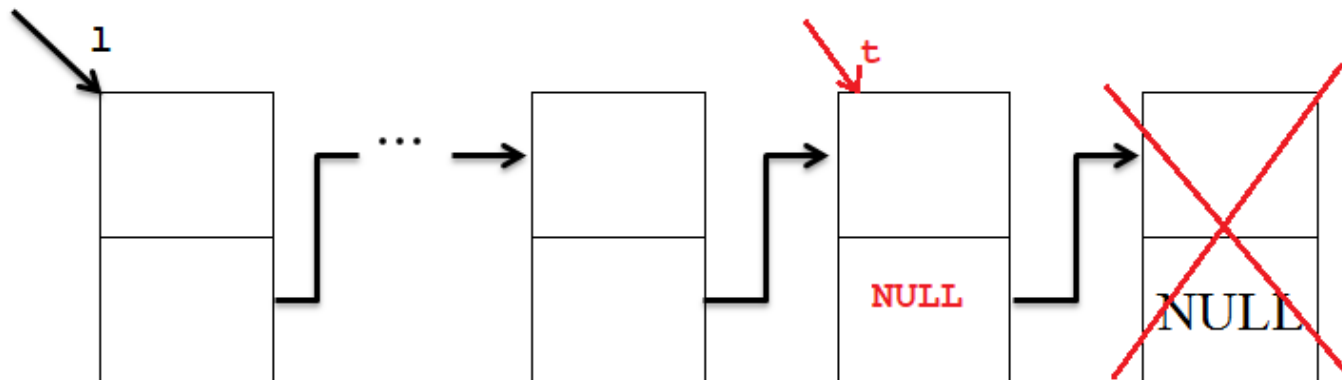


Удаление последнего элемента

Исходный список:



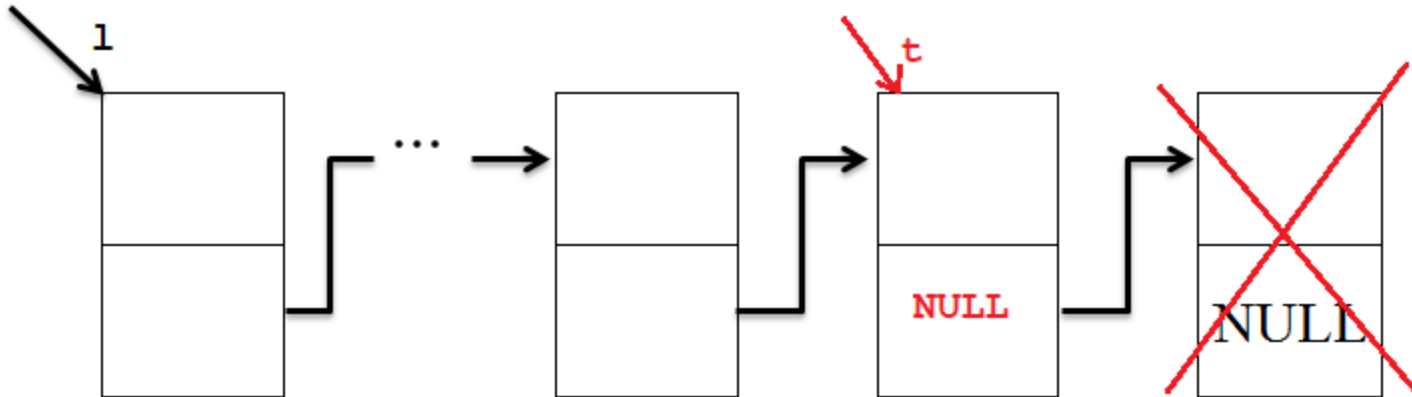
Результат:



Чтобы удалить последний элемент из списка, необходимо предварительно найти предпоследний элемент - путем просмотра списка получить указатель на него.

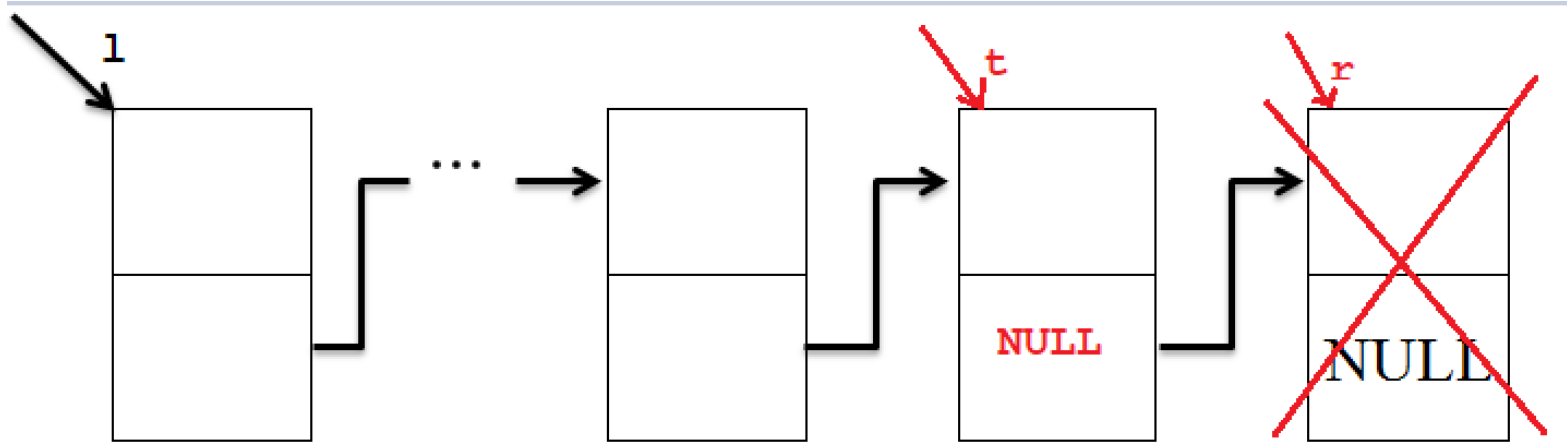
Удаление последнего элемента (1 способ)

```
struct list* del_last(struct list *l) {  
    struct list* t;  
    t=l;  
    while (t->next->next!=NULL)  
        t=t->next;  
    // for(t=l; t->next->next!=NULL; t=t->next);  
    free(t->next);  
    t->next=NULL;  
    return l;  
}
```



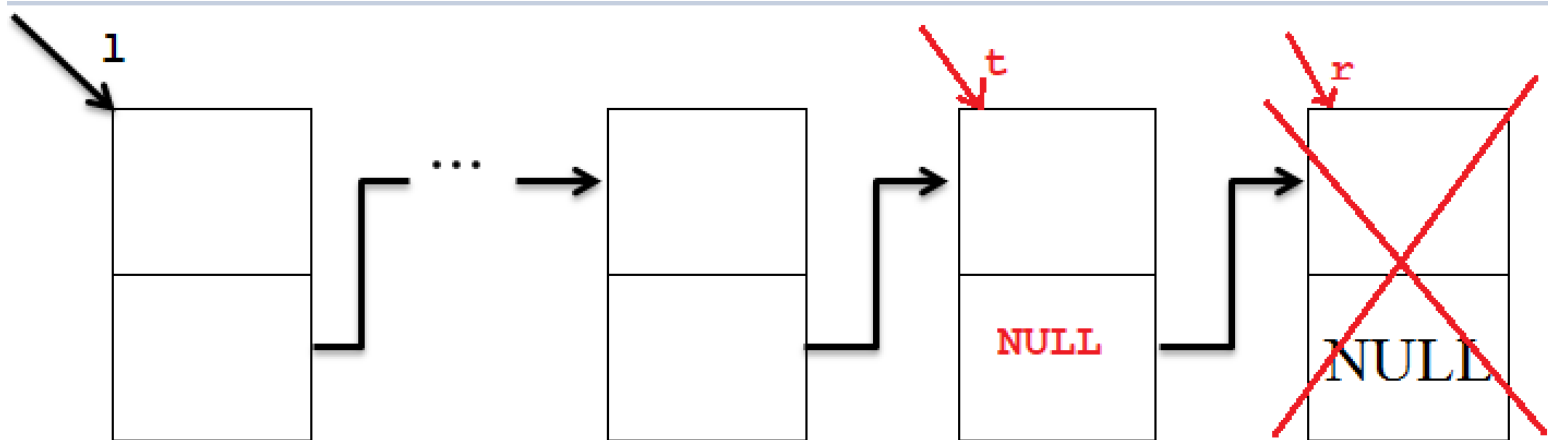
Удаление последнего элемента (2 способ)

Используем дополнительную переменную r – указатель на удаляемый элемент.



Удаление последнего элемента (2 способ)

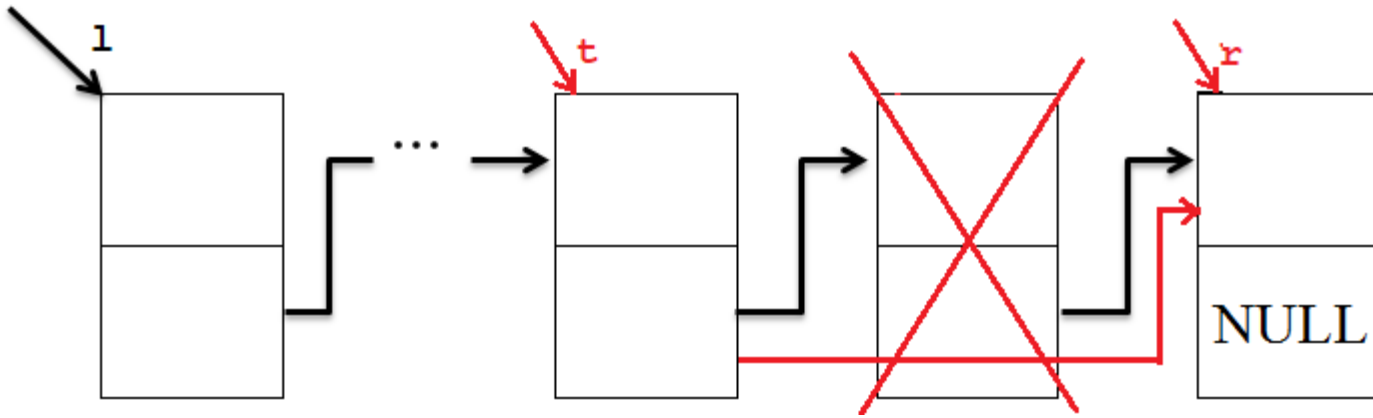
```
struct list* del_last(struct list *l) {  
    struct list* t,*r;  
  
    for(t=l; t->next->next!=NULL; t=t->next);  
    r=t->next;  
    free(r);  
    t->next=NULL;  
  
    return l;  
}
```



Удаление предпоследнего элемента (1)

Последовательность действий:

- ищем третий элемент от конца (t),
- находим указатель на последний элемент (r),
- удаляем предпоследний элемент (освобождаем память),
- расставляем указатели.



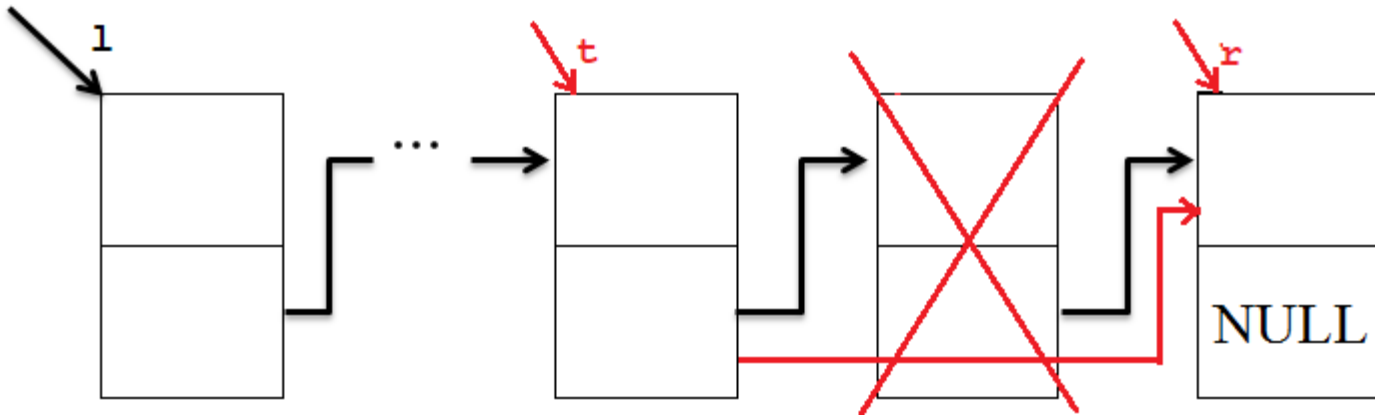
Удаление предпоследнего элемента (1)

```
struct list* del_before_last(struct list * l) {  
    list* t,* r;
```

```
    for(t=l; t->next->next->next!=NULL; t=t->next);  
    r=t->next->next;  
    free(t->next);  
    t->next=r;
```

```
    return l;
```

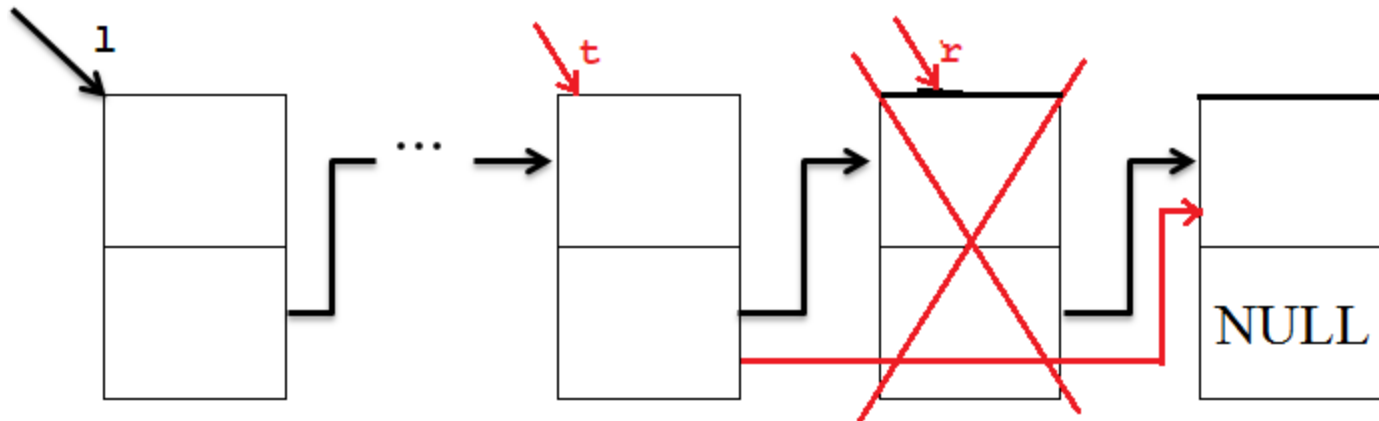
```
}
```



Удаление предпоследнего элемента (2)

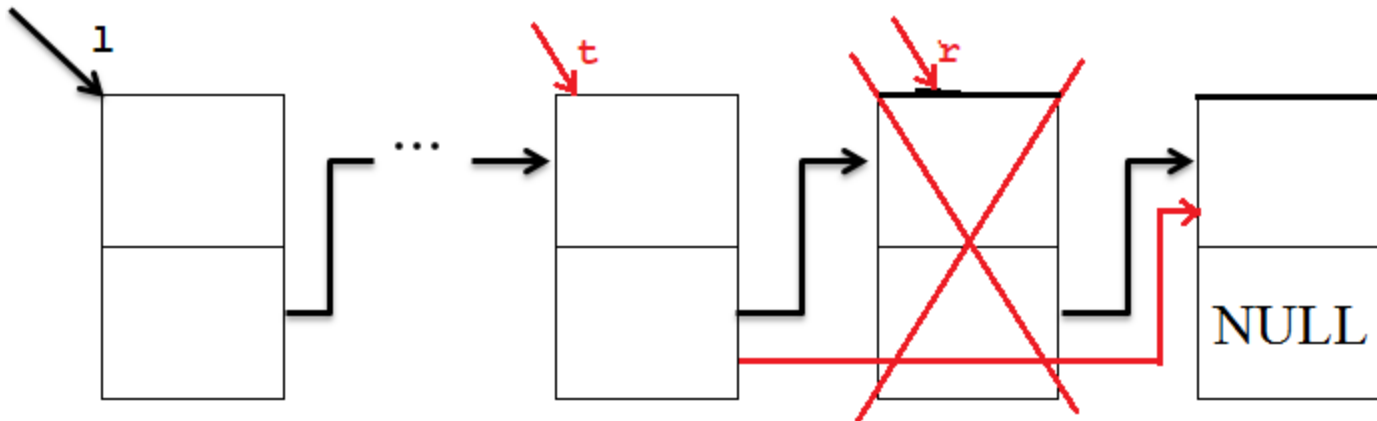
Последовательность действий:

- ищем третий элемент от конца (t),
- находим указатель на предпоследний элемент (r),
- расставляем указатели,
- удаляем предпоследний элемент (освобождаем память).



Удаление предпоследнего элемента (2)

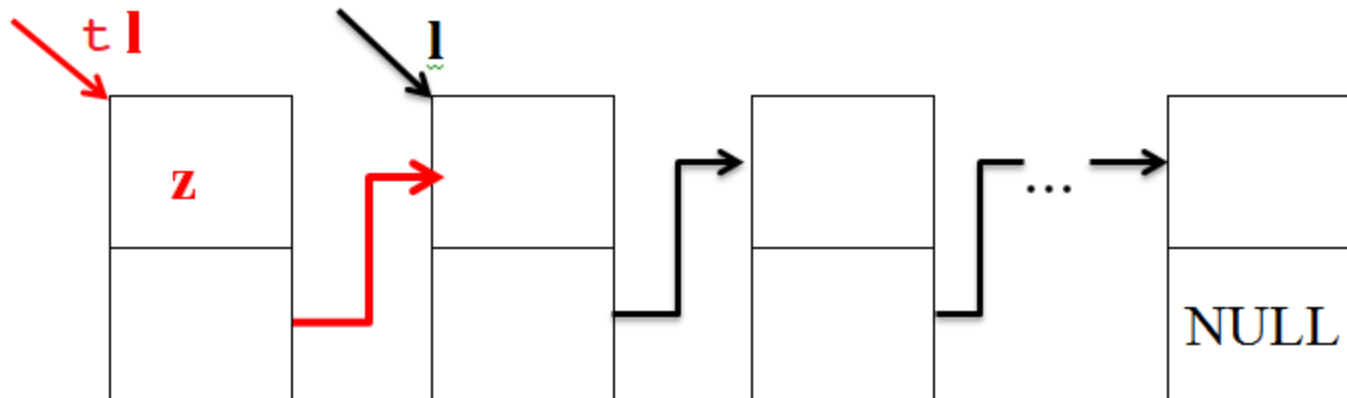
```
struct list* del_before_last(struct list * l) {  
    list* t,* r;  
  
    for(t=l; t->next->next->next!=NULL; t=t->next);  
    r=t->next;  
    t->next=r->next;    free(r);  
  
    return l;  
}
```



Добавление в начало списка (перед первым элементом)

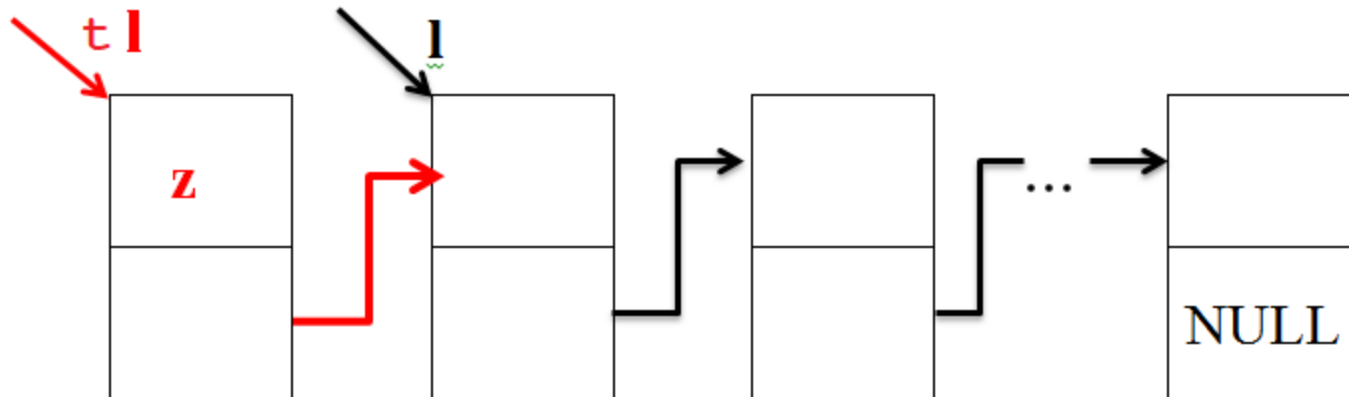
Алгоритм:

- выделить память под новый элемент (**t**);
- заполнить информационное поле (**z**);
- расставить указатели (связать).



Добавление в начало списка (перед первым элементом)

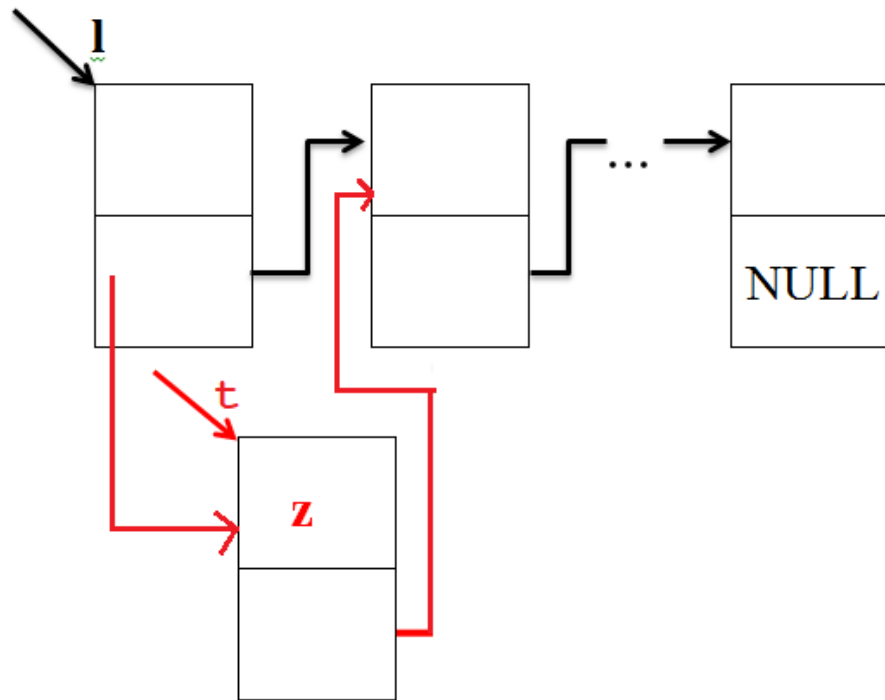
```
struct list * add_first(struct list * l, struct data z) {  
    struct list* t;  
  
    t=(struct list *)malloc(sizeof(struct list));  
    t->inf=z;  
    t->next=l;  
    l=t;  
  
    return l;  
}
```



Добавление после первого элемента

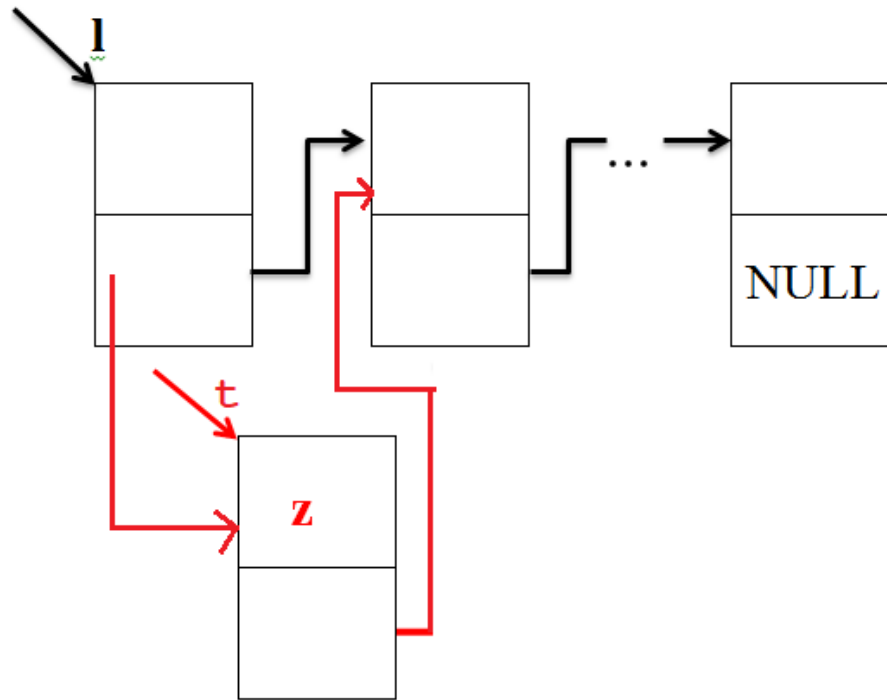
Алгоритм:

- выделить память под новый элемент (t);
- заполнить информационное поле (z);
- расставить указатели (связать).



Добавление после первого элемента

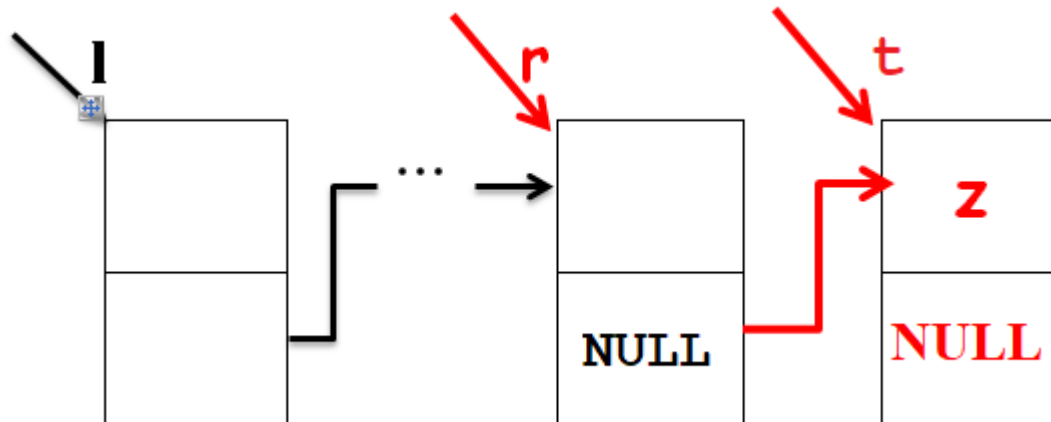
```
struct list* add_second(struct list * l, struct data z) {  
    struct list* t;  
  
    t=(struct list *)malloc(sizeof(struct list));  
    t->inf=z;  
    t->next=l->next;  
    l->next=t;  
  
    return t;  
}
```



Добавление после последнего элемента

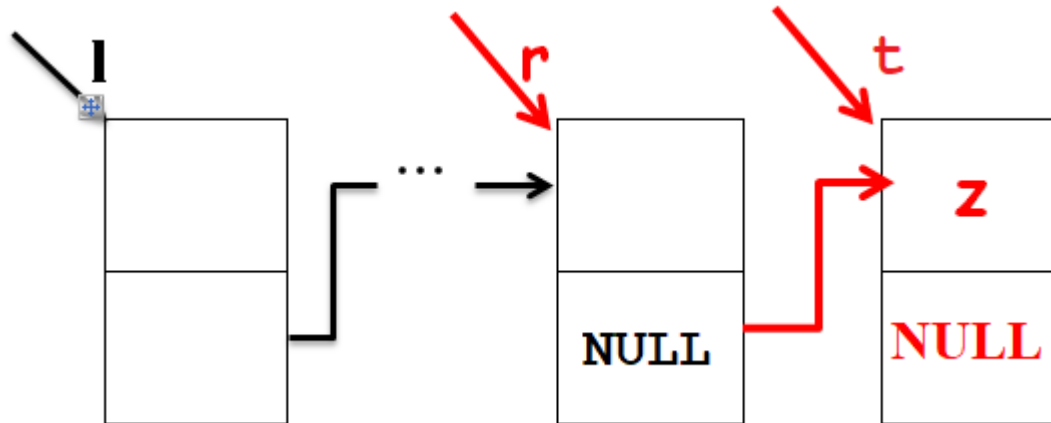
Алгоритм:

- выделить память под новый элемент;
- заполнить информационное поле;
- найти указатель на последний элемент;
- расставить указатели (связать с новым элементом).



Добавление после последнего элемента

```
struct list * add_last(struct list * l, struct data z) {  
    list *t, *r;  
    t=(struct list *)malloc(sizeof(struct list));  
    t->inf=z;  
    t->next=NULL;  
  
    r=l;                                // поиск  
    for(; r->next!=NULL; r=r->next);  
    r->next=t;                          // связывание  
  
    return t;  
}
```



Чтение данных информационного поля

```
struct data read_data() {  
    struct data z;  
    printf(" введите номер, имя и возраст \n");  
    scanf("%d %s %d", &z.num, &z.name, &z.age);  
    return z;  
}
```

Вызов функции добавления первого элемента в main():

```
case 1:{  
    z=read_data();  
    l=add_first(l,z);  
    break;  
}
```

Организация списка

```
struct list* create_list(struct list* l) {
    int fl=0;
    struct data z;
    do {
        if (l==NULL) {                // если список пустой
            z=read_data();             // считать данные
            l=add_first(l,z);          // добавить в начало
        } else {
            z=read_data();             // иначе считать данные
            add_last(l,z);              // добавить в конец
        }
        printf("еще 1-да 0-нет");
        scanf("%d",&fl);
    } while(fl);

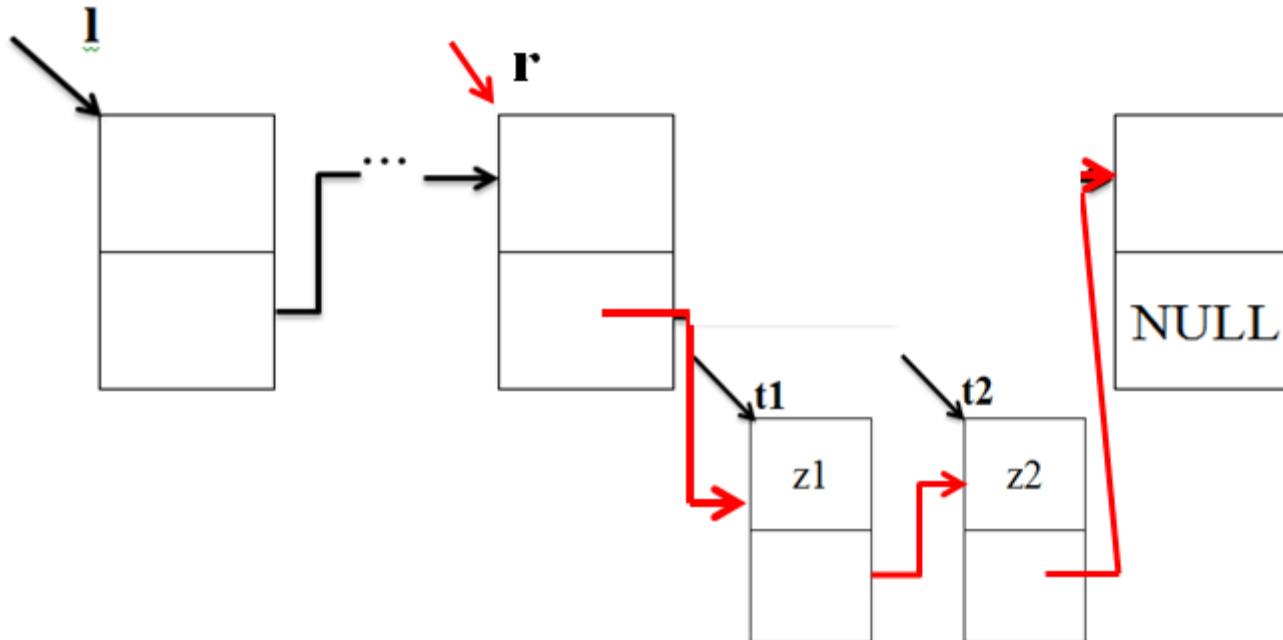
    return l;
}
```

Поиск значения среднего возраста

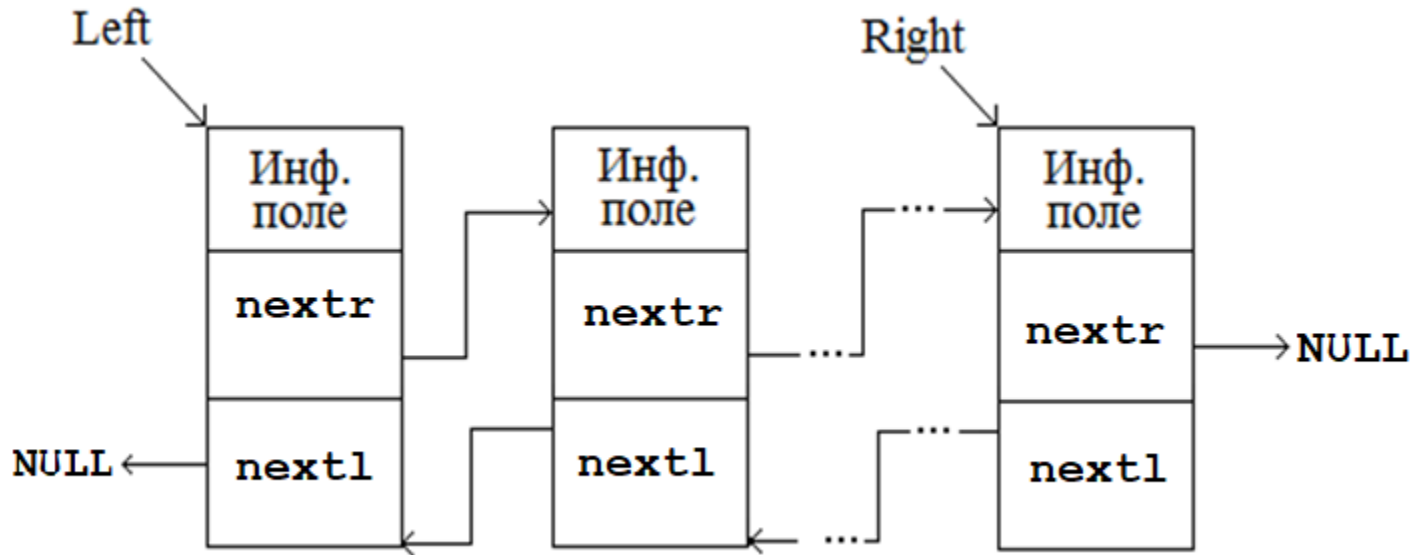
```
float average(struct list * l) {  
    float s=0;           //сумма  
    int k=0;             // количество  
  
    list *t;  
    for(t=l; t!=NULL; t=t->next) {  
        k++;  
        s+=t->inf.age;  
    }  
  
    if (k==0) return 0;  // если список пуст  
  
    return s/k;  
}
```

Добавление двух новых элементов перед последним

Схематичное решение задачи:



Двунаправленные списки



Описание:

```
struct bidir_list {  
    <ТИП ИНФ. ПОЛЯ> inf;  
    struct bidir_list * nextl, * nextr;  
};
```

В main() локальные переменные:

```
struct bidir_list * Left, * Right;
```

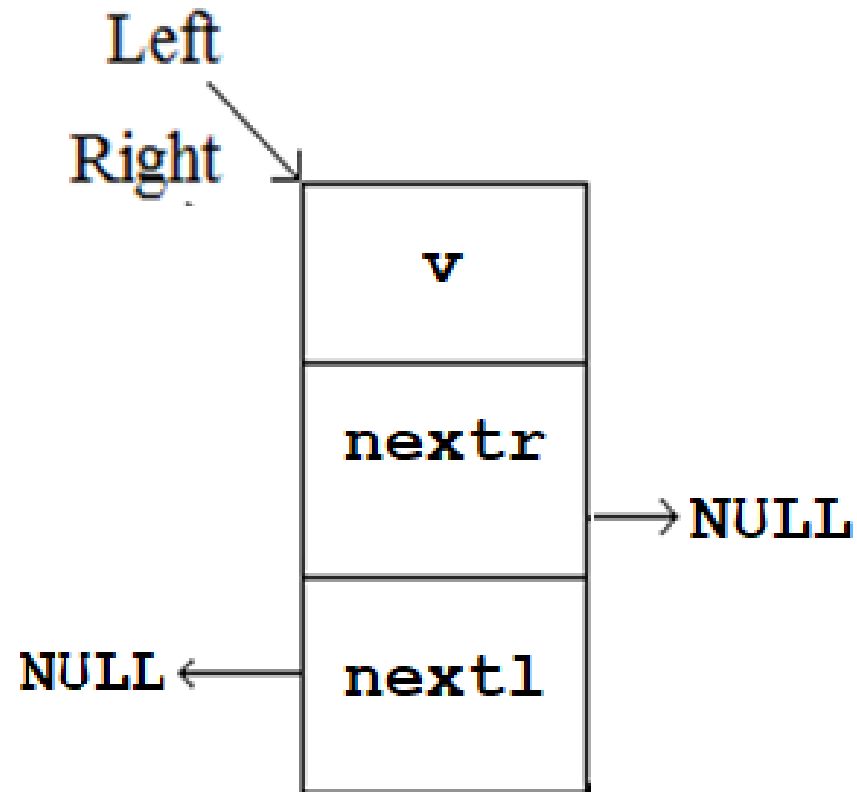
Двунаправленные списки

Операции над двунаправленными списками:

- создание списка, т.е. внесение в список первого элемента;
- добавление элемента в начало списка (со стороны **Left**);
- добавление элемента в конец списка (со стороны **Right**);
- вставка элемента в середину списка;
- удаление первого или последнего элемента списка;
- удаление элемента из середины списка.

Двунаправленные списки

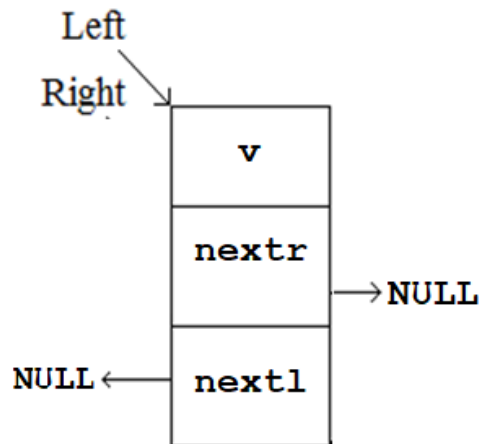
Создание 2-направленного списка:



Двунаправленные списки

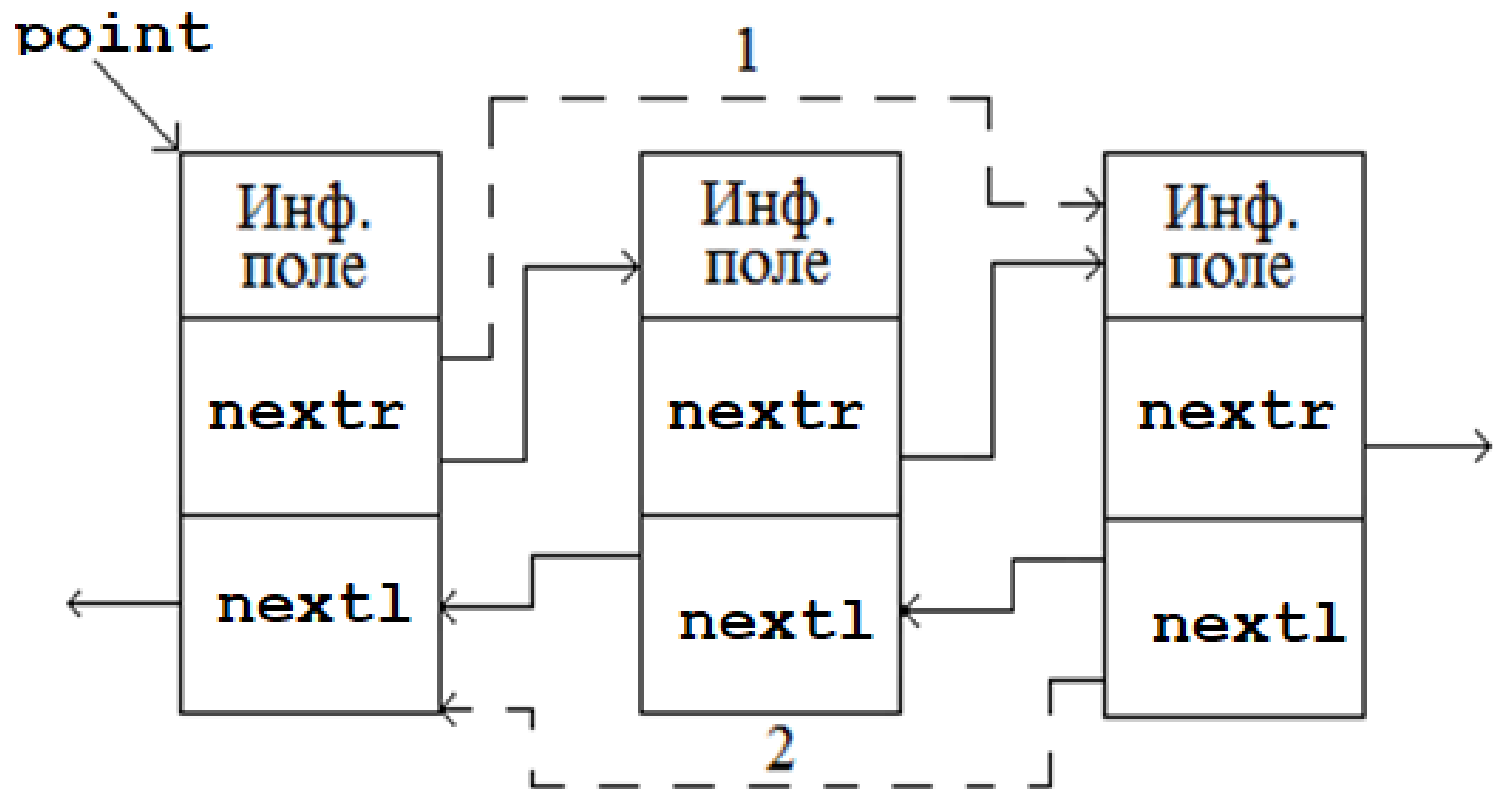
Создание 2-направленного списка:

```
struct bidir_list* create2list(bidir_list * Left,  
                               bidir_list * Right){  
  
    int v;  
    scanf("%d", &v);          // считывание данных для инф. поля  
                               // выделение памяти  
    struct bidir_list* t=(struct bidir_list*) malloc  
                               (sizeof(struct bidir_list));  
  
    t->inf=v;                  // присвоение значения инф. полю  
    t->nextl=NULL;             // присвоение значения указателю на сосед. эл-т слева  
    t->nextr=NULL;             // присвоение значения указателю на сосед. эл-т справа  
    Left=t;                   // присвоение значения указателю конца списка  
    Right=t;                   // присвоение значения указателю начала списка  
    return Left;  
}
```



Двунаправленные списки

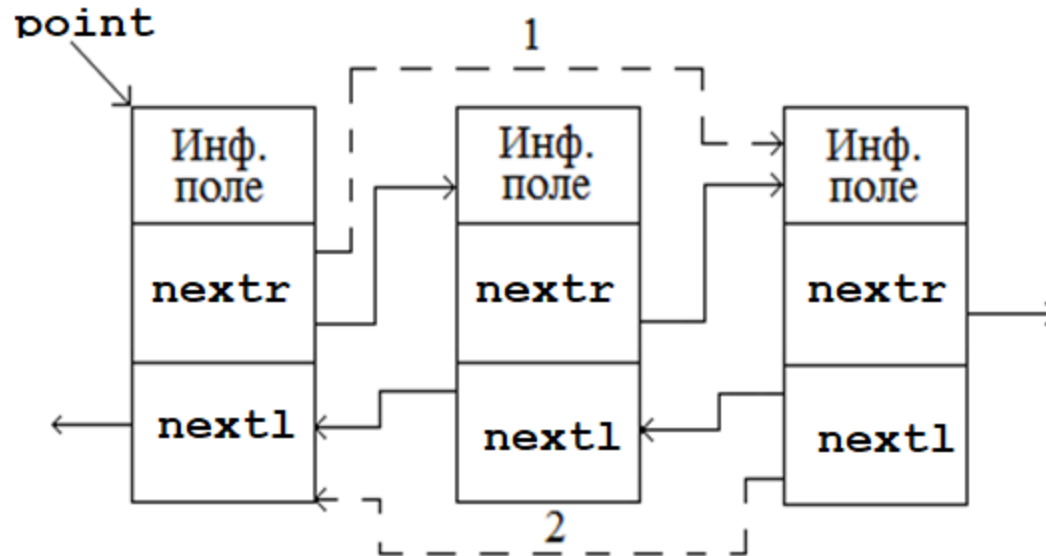
Удаление элемента после `point`:



Двунаправленные списки

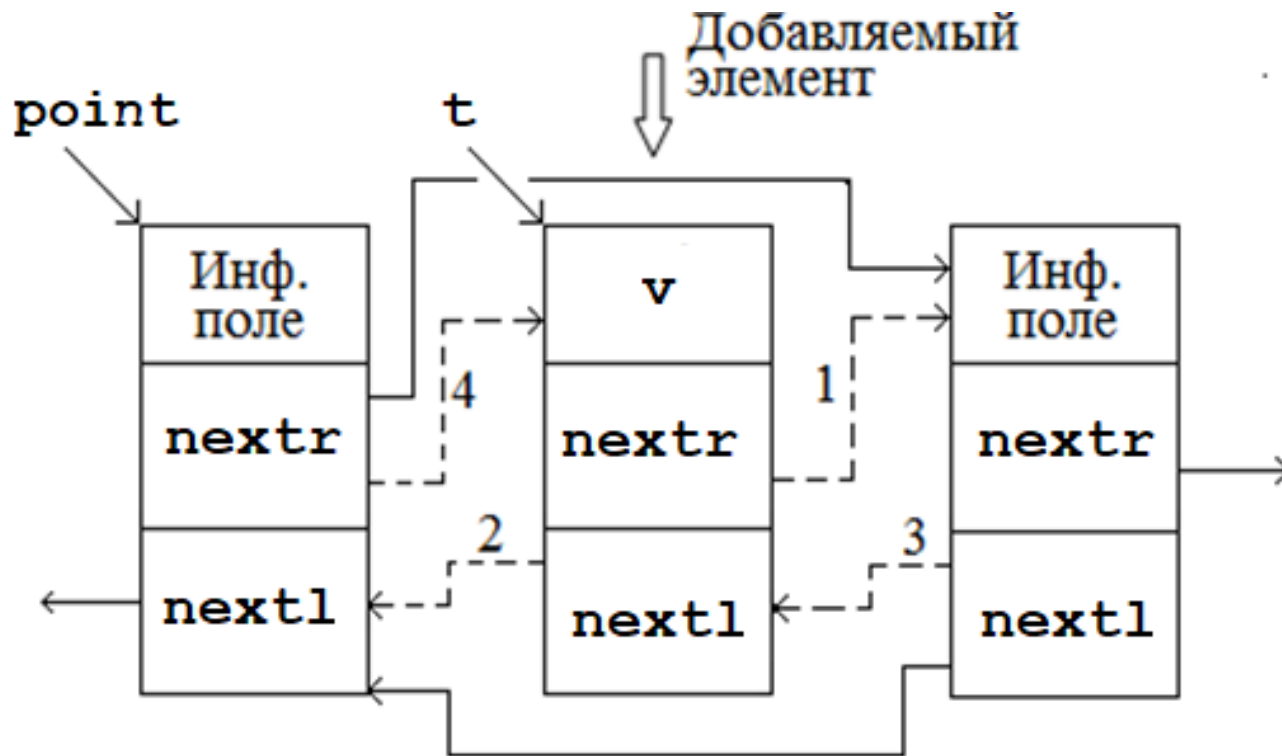
Удаление элемента после point:

```
struct bidir_list* del_el_after_point(bidir_list * point){  
    struct bidir_list* t;  
  
    t=point->nextr;           //запомнить адрес удаляемого эл-та  
    point->nextr=point->nextr->nextr; //создать связь 1  
    //point->nextr= temp->nextr;  
    point->nextr->nextl=point; //создать связь 2  
    free(t);                 //освободить память от элемента  
}
```



Двунаправленные списки

Добавление нового элемента после `point`:



Двунаправленные списки

Добавление нового элемента после point

```
struct bidir_list* insert_after_point (bidir_list* point){
    struct bidir_list* t;
    int v;
    scanf ("%d", &v); //ввести значение инф. поля
    t=(struct bidir_list*) malloc /* создать новый эл-т */
        (sizeof(struct bidir_list));
    t->inf=v; // присвоить значение инф. полю
    t->nextr=point->nextr; // создать связь 1
    t->nextl=point; // создать связь 2
    t->nextr->nextl=t; // создать связь 3
    point->nextr=t; // создать связь 4
}
```

