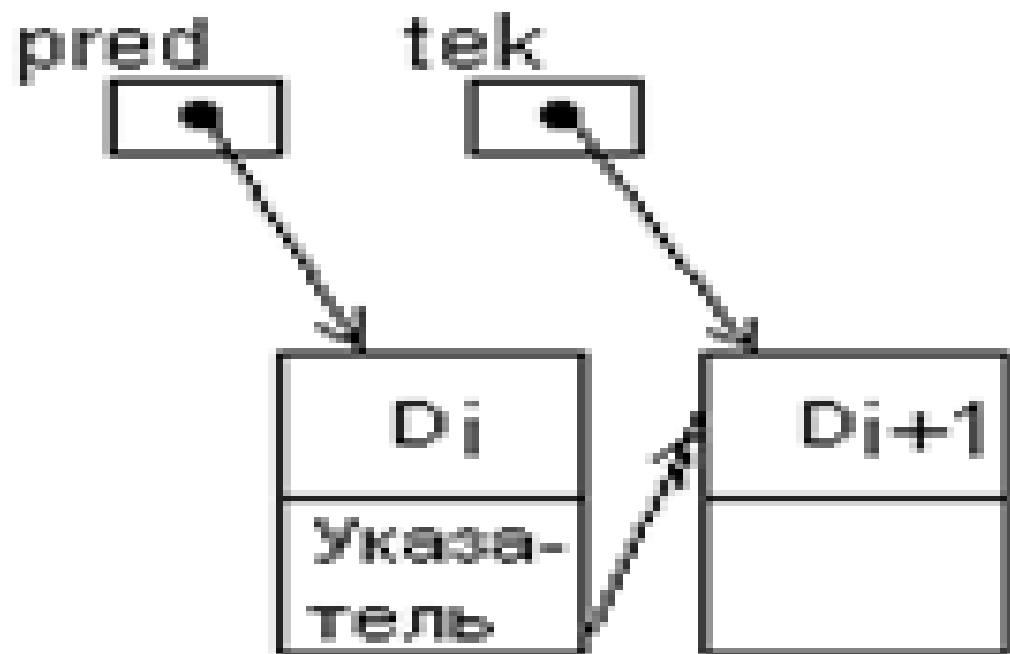


Лекция 6.

Операции над линейными списками

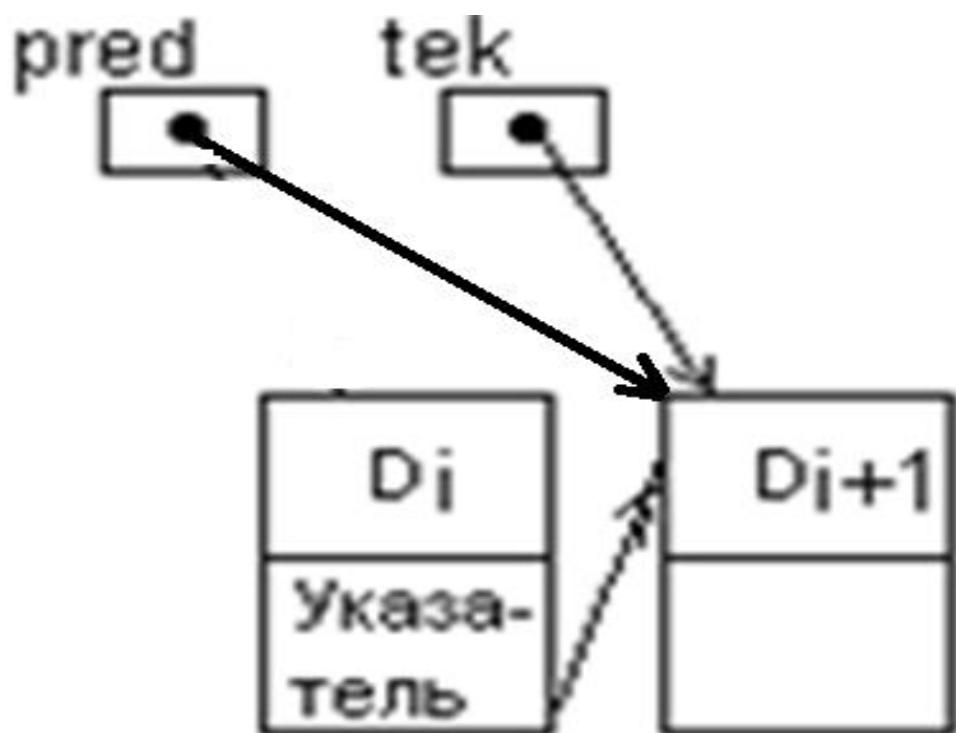
Связывание элементов

```
pred -> link = tek; // в адресную часть pred  
// присваивается значение tek
```



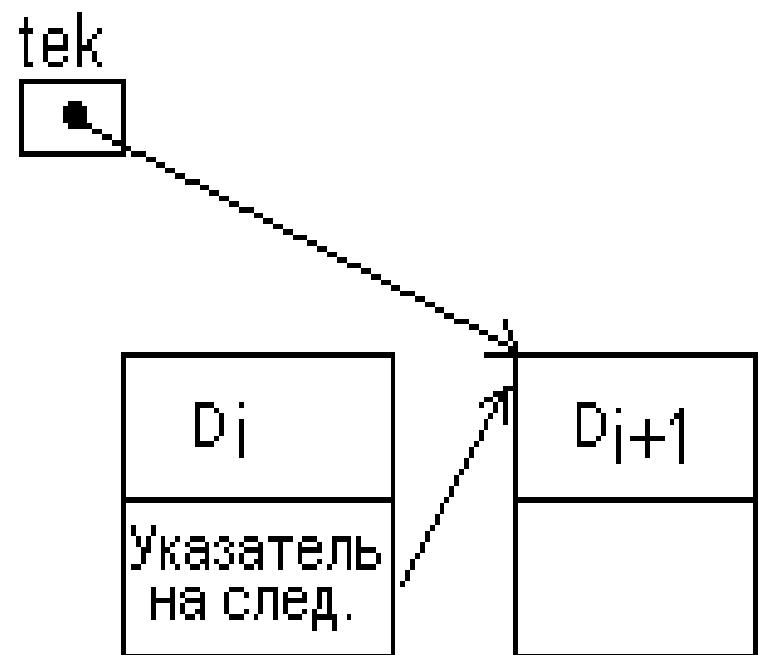
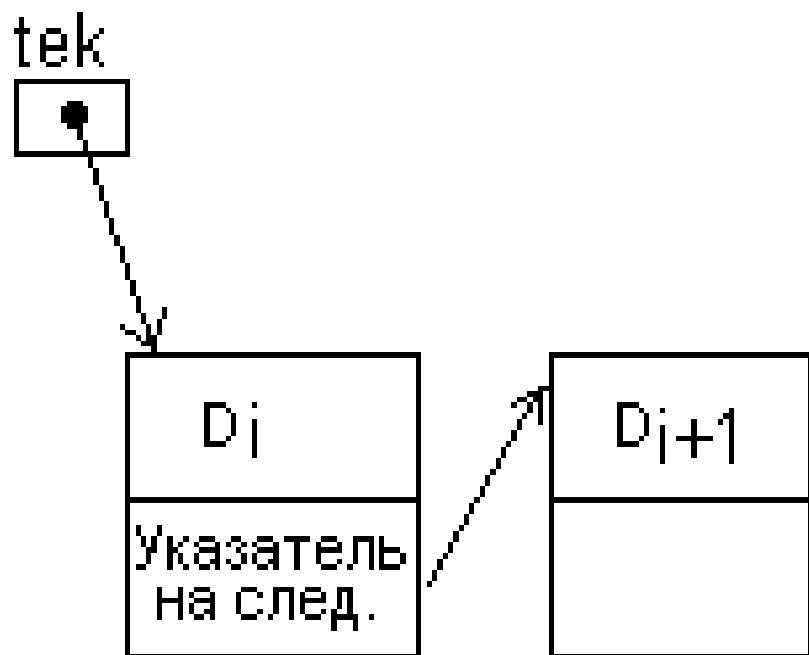
Переприсваивание указателей

```
pred = tek; // pred и tek указывают на  
           // один и тот же элемент
```



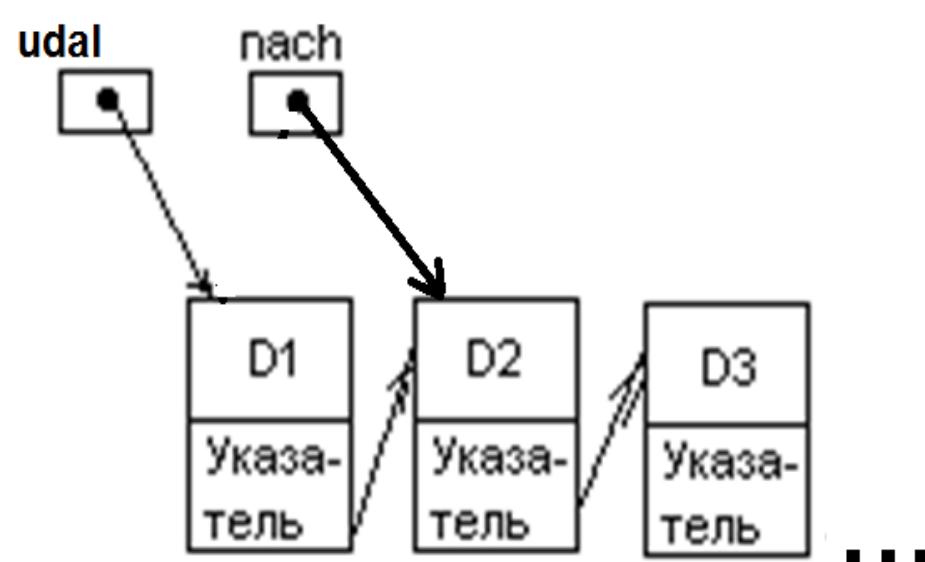
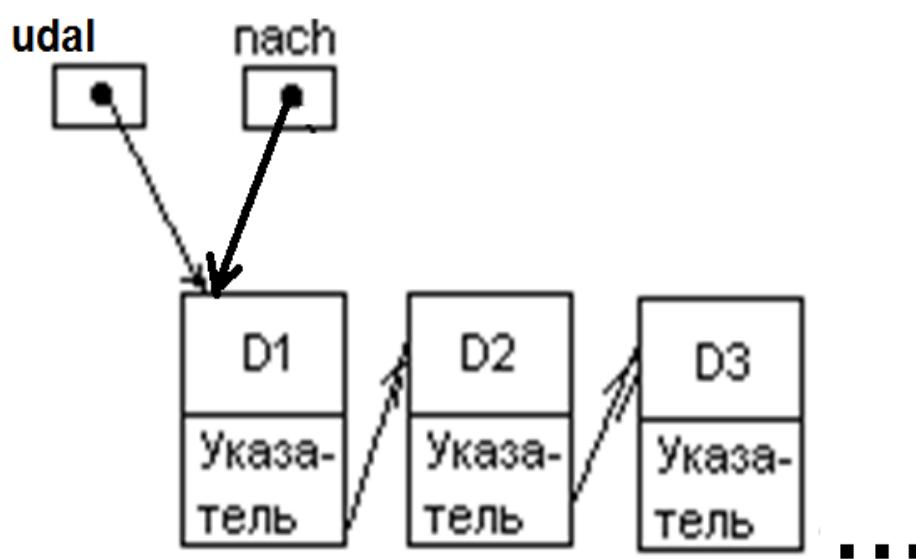
Шаг по связи

```
tek = tek->link; // указателю tek присваивается  
// адрес следующего за tek элемента
```



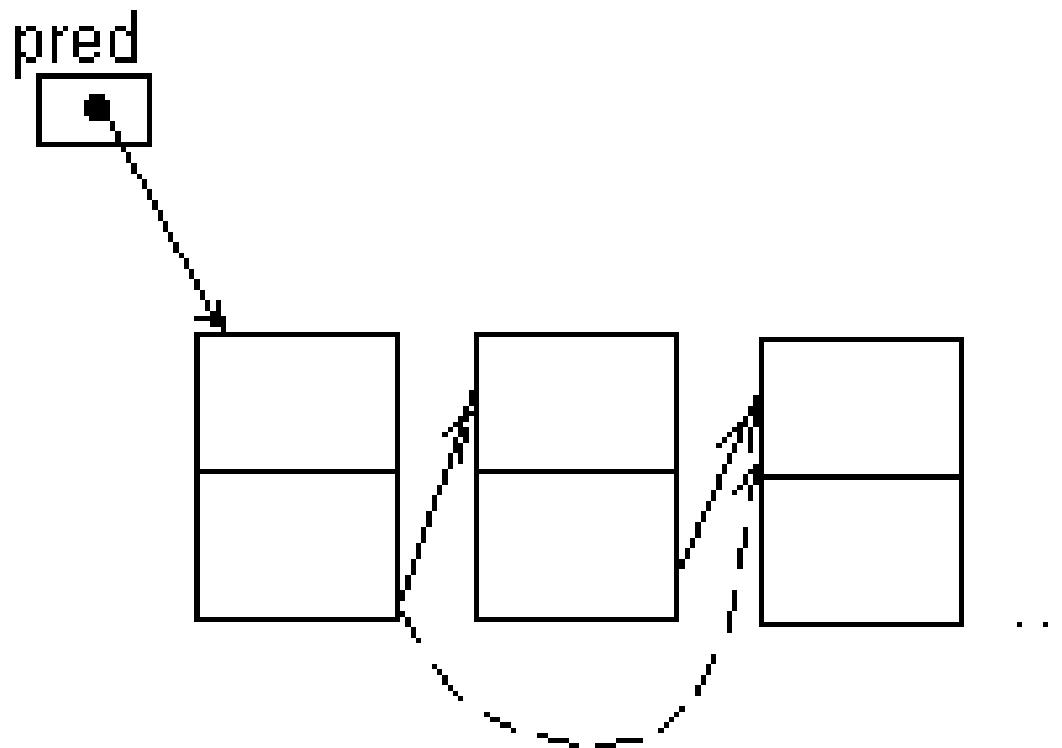
Удаление элемента в начале списка

```
udal = nach; //запомнить первый для удаления  
nach=nach->link; //запомнить новое начало  
free(udal); //освободить память
```

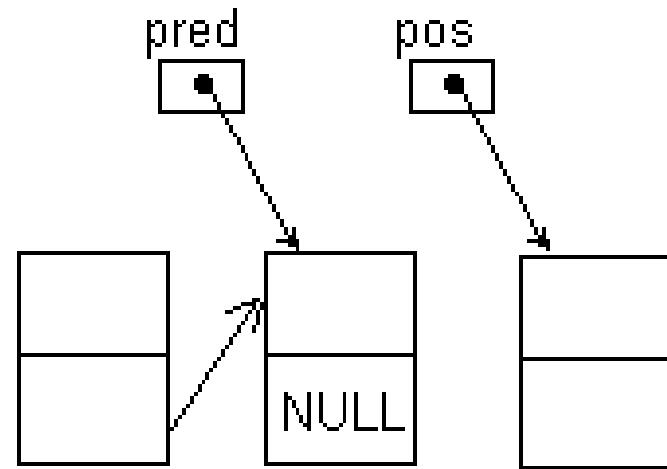
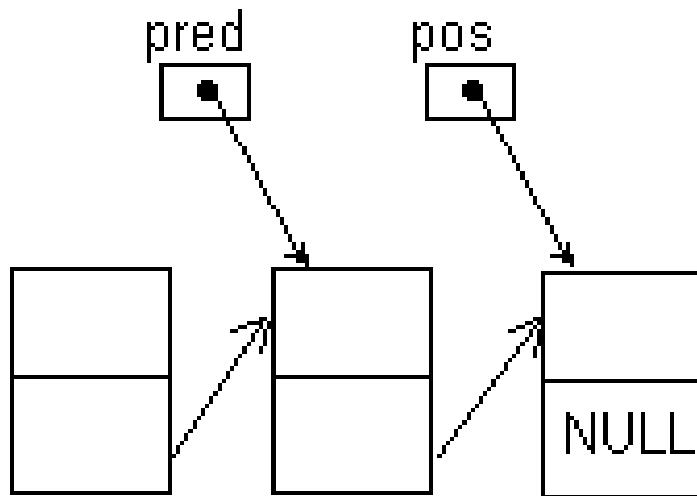


Удаление элемента из середины списка

`pred->link = pred->link->link;`

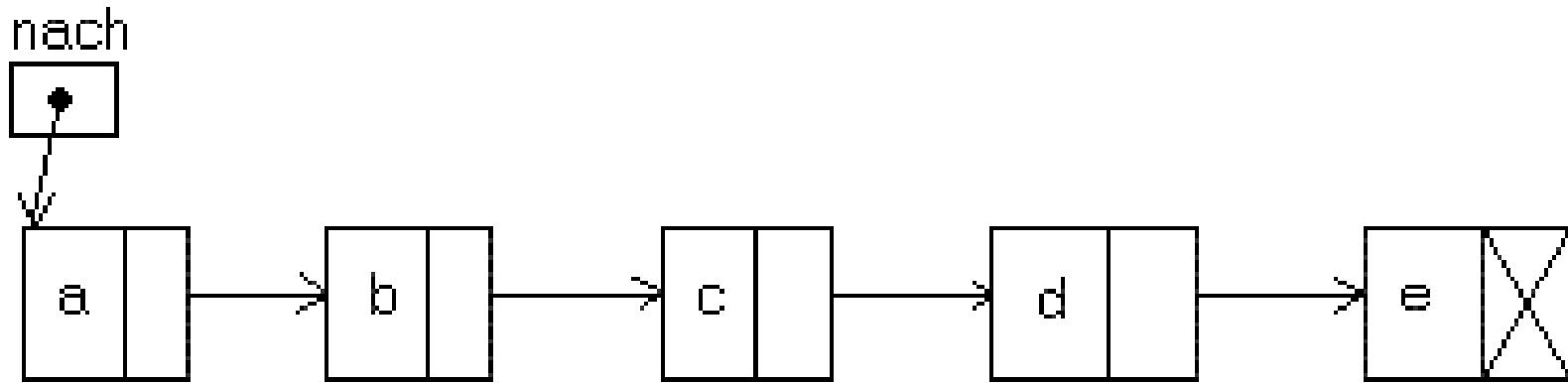


Удаление элемента в конце списка



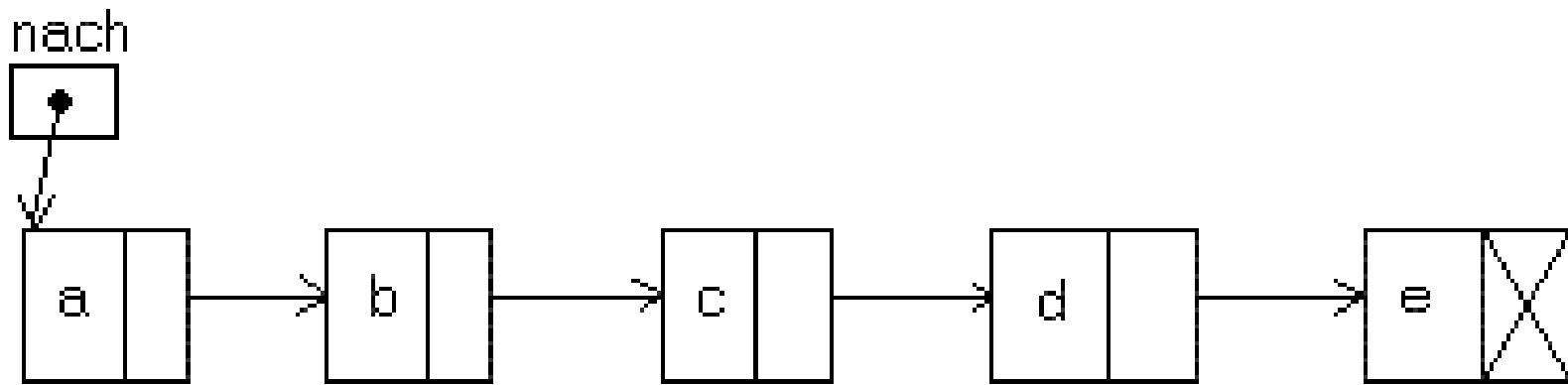
1. **pred->link=NULL;**
2. **free(pos);**

Пример 1. Длинный доступ по связям



1. `printf("%s", nach->d);`
2. `printf("%s", nach->link->d);`
3. `printf("%s", nach->link->link->link->d);`

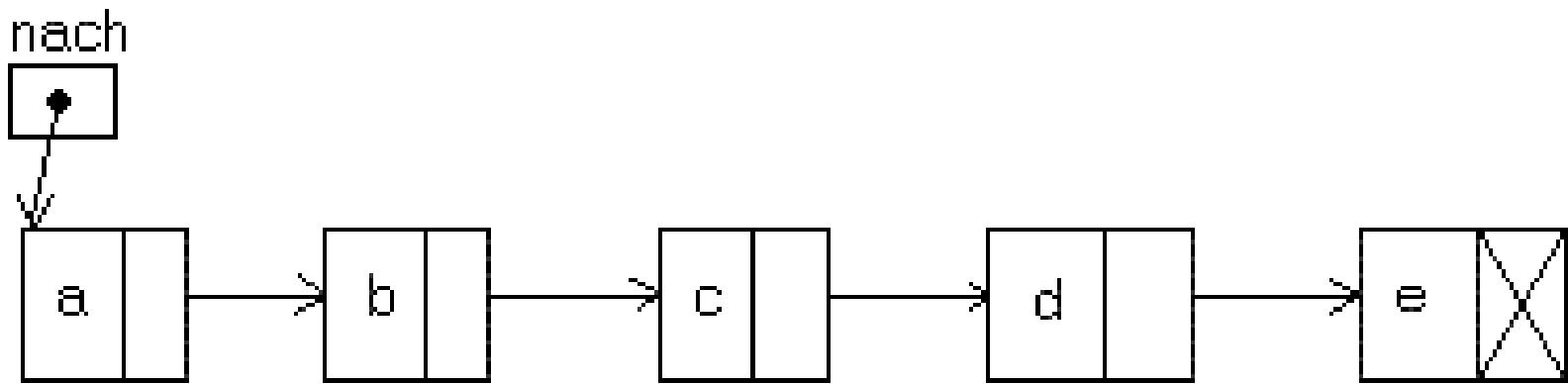
Пример 2. Перемещение по списку



Фрагмент программы:

```
tek=nach;  
while(tek->d != 'c')  
    tek=tek->link;  
printf("%s", tek->d);
```

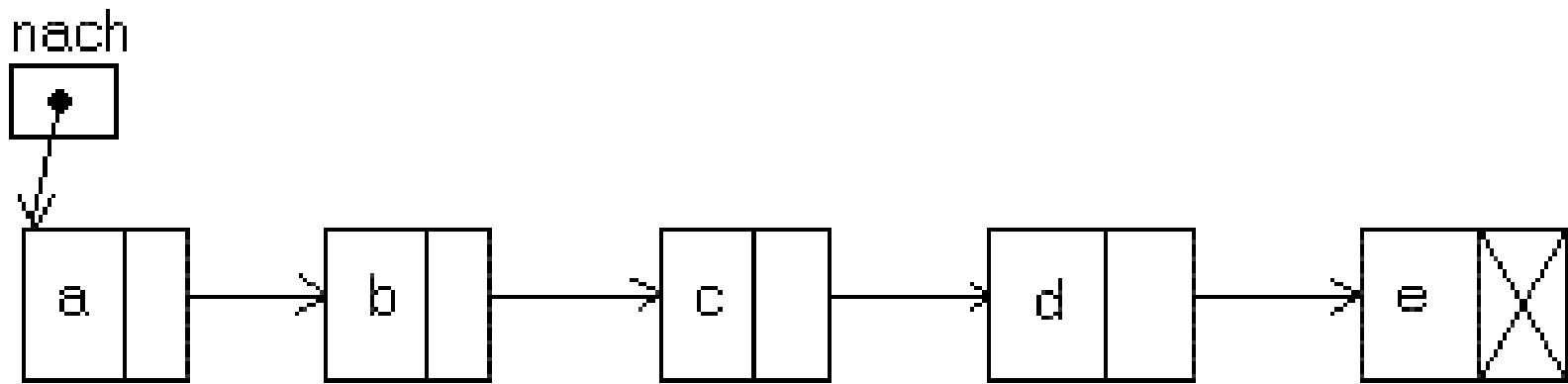
Пример 3. Перемещение по списку



Фрагмент программы:

```
tek=nach;  
for(i=1; i<=4; i++)  
    tek=tek->link;  
printf("%s", tek->d);
```

Пример 4. Перемещение по списку



Фрагмент программы:

```
tek=nach;  
while(tek->link != NULL)  
    tek=tek->link;  
printf("%s", tek->d);
```

Пример 5. Перемещение по списку

Фрагмент 1 программы (ошибка сегментирования):

```
tek=nach;  
while(tek != NULL)  
    tek = tek->link;  
printf("%s", tek->d);
```

Фрагмент 2 программы:

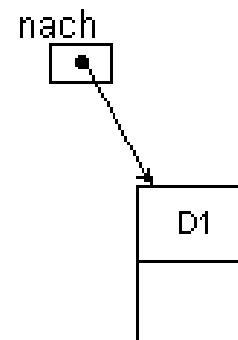
```
tek=nach;  
while(tek->link->link != NULL)  
    tek = tek->link;  
printf("%s", tek->d);
```

Алгоритм построения списка в обратном порядке

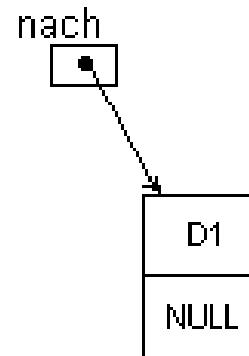
1. Указатью **tek** присвоить
пустое значение адреса:
tek=NULL;



2. Создать в динамической
памяти элемент **nach**.
Ввести **nach->d;**

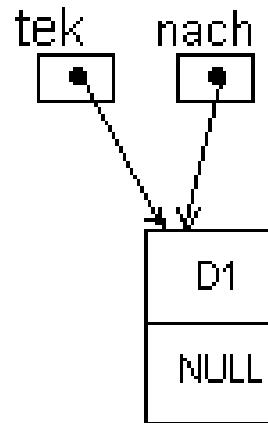


3. Связать
элементы **nach** и **tek**:
nach->link=tek;

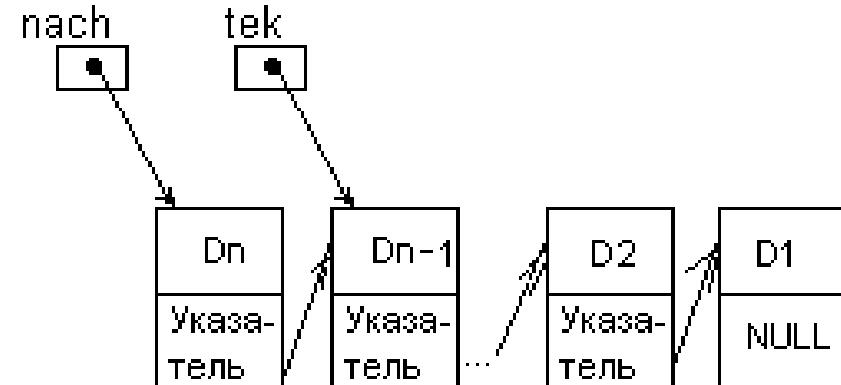


Продолжение. Алгоритм построения списка в обратном порядке

4. Установить
указатель **tek** на
начальный элемент:
tek=nach;



5. Если не конец ввода то
перейти на 2.



Начало программы построения списка в обратном порядке. Описания.

Функция построения списка в обратном порядке (следующий слайд)

struct element

{

int d;

struct element *link;

};

struct element *nach;

```
1. void vvod_2( ) { //добавление нового в начало списка
2.     struct element *tek;
3.     int i;
4.     tek = NULL;
5.     scanf("%d", &i);
6.     while(i != 0) {
7.         nach=(struct element *)malloc(sizeof(struct element));
8.         nach->d = i;
9.         nach->link = tek;
10.        tek = nach;
11.        scanf("%d", &i);
12.    }
13. }
```

Функция просмотра списка

```
1. void prosmotr() {  
2.     struct element *tek;  
3.     tek = nach;          // Встали на начало списка  
4.     while(tek != NULL) {  // Пока не конец списка  
5.         printf("%d", tek->d);  
6.         tek = tek->link;    // переход на следующий  
7.     }  
8. }
```

Функция main()

```
1. int main() {  
2.     vvod_2();  
3.     prosmotr();  
4.     return 0;  
5. }
```

Рекурсивная функция просмотра списка

```
1. void prosmotr_2( struct element *tek )
2. {
3.     if( tek != NULL )
4.     {
5.         printf("%d", tek->d);
6.         prosmotr2(tek->link);
7.     }
8. }
```

Обращение:

prosmort_2(nach);

Стек в динамической памяти

```
struct stack
{
    int d;
    struct stack *link;
};

void push( struct stack**, int n);
int pop(struct stack**);
```

```
1. int main() {  
2.     struct stack *S;  
3.     int i;  
4.     scanf("%d", &i);  
5.     while(i!=0) {  
6.         push(&S, i);  
7.         scanf("%d", &i);  
8.     }  
9.     while(S!= NULL) {  
10.         i=pop(&S);  
11.         printf("%d ", i);  
12.     }  
13.     return 0;  
14. }
```

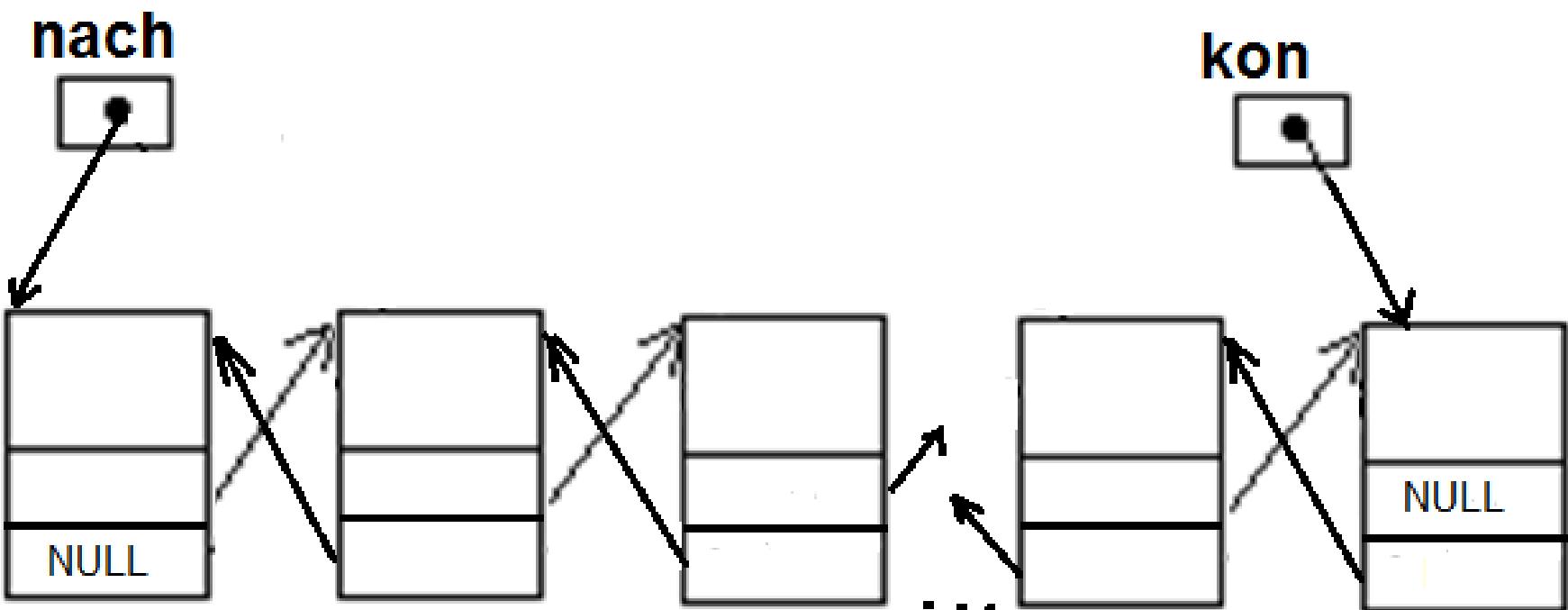
ФУНКЦИЯ ЗАПОЛНЕНИЯ СТЕКА

1. **void push(struct stack **S, int i) {**
2. **struct stack *tek;**
3. **tek = (struct stack *)(malloc(sizeof(struct stack)));**
4. **tek->d = i;**
5. **tek->link = *S;**
6. ***S = tek;**
7. }

Функция извлечения элемента из стека

```
1. int pop(struct stack **S) {  
2.     struct stack *tek;  
3.     int i;  
4.     tek= *S;  
5.     i=tek->d;  
6.     *S=(*S) ->link;;  
7.     return i;  
8. }
```

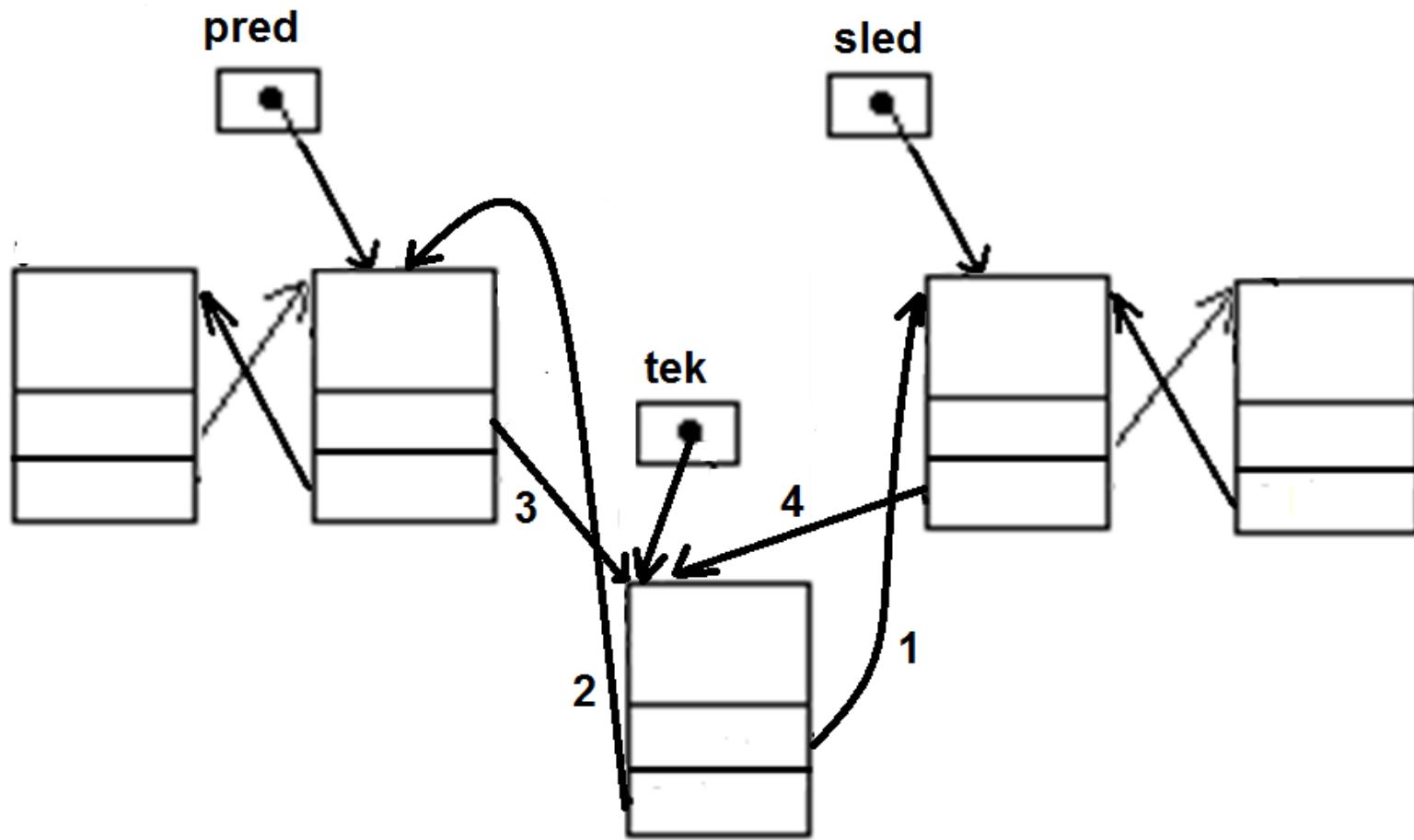
Списки с двумя связями



Списки с двумя связями

1. **struct element**
2. {
3. **int d;**
4. **struct element *rlink, *llink;**
5. };
6. **struct element *nach, *kon;**

Добавление элемента в середину двусвязного списка



Связи

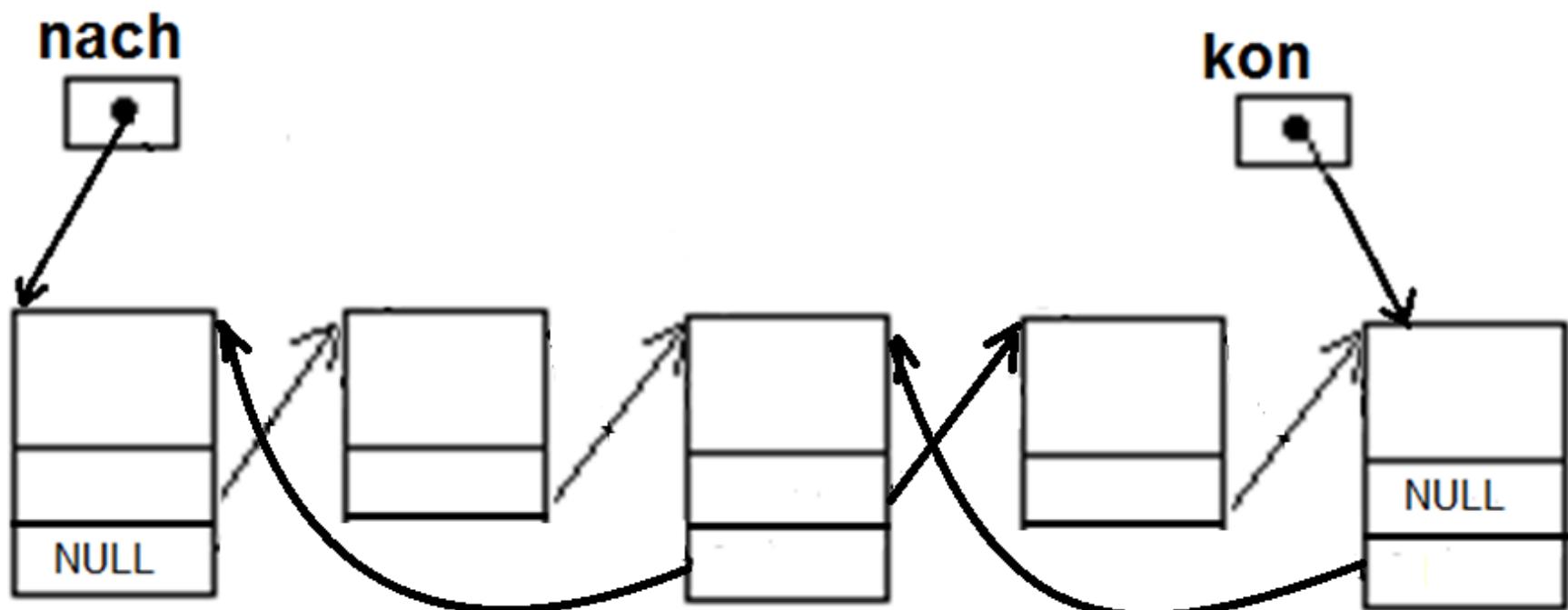
1. **tek -> right = sled;**
2. **tek -> left = pred;**
3. **pred ->right = tek;**
4. **sled -> left= tek;**

или (1) и (4):

tek -> right = pred -> right;

pred -> right -> left = tek;

Списки с полутора связями



Списки с полутора связями. Описания

```
1. struct element2
2. {
3.     char c;
4.     struct element1 *rlink;
5.     struct element2 *llink;
6. };
7. struct element1
8. {
9.     int d;
10.    struct element2 *link;
11.};
```