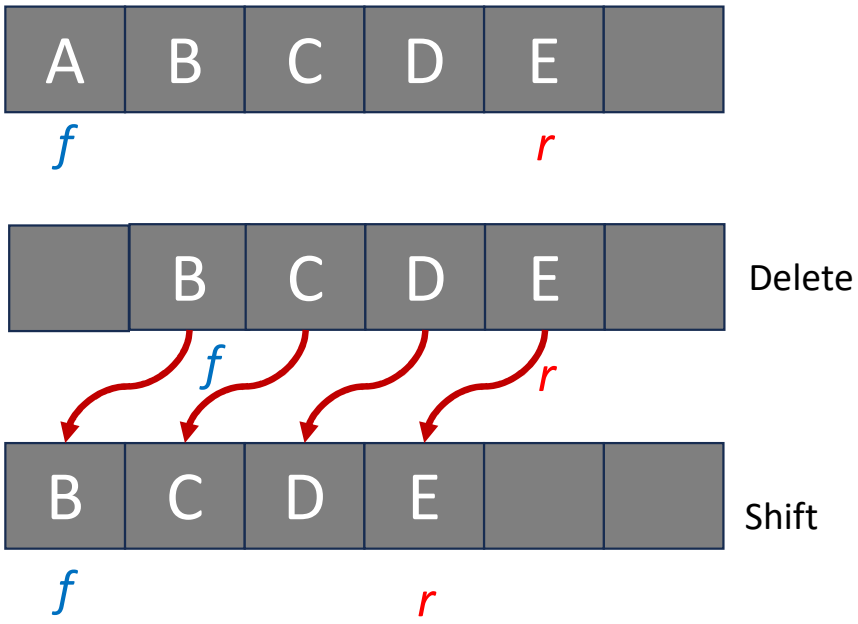
Linked lists

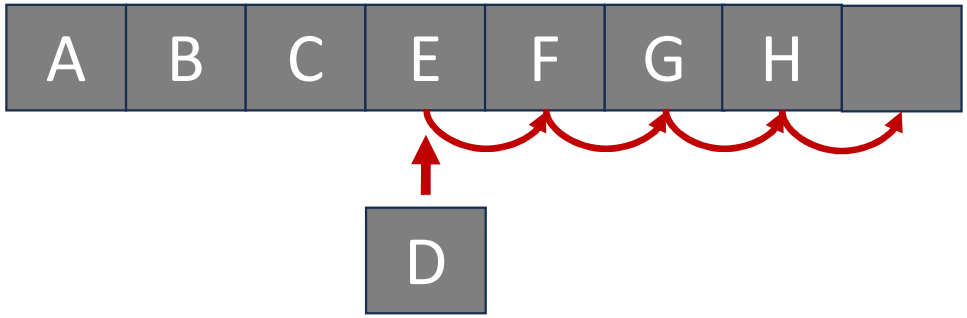
Ch. 4

Deletion and insertion in sequential representation

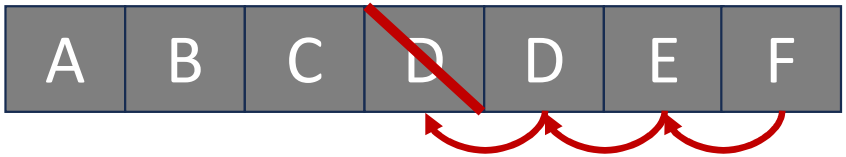
- Deletion in Queue



- Insert an element in Array



- Delete an element into Array



To maintain the sequential representation, we have to move many elements.

Memory layout

To store the list {A, B, C, D, E}

- **Using sequential representation**

Successive items of a list are located a fixed distance apart.

- **Using linked representation**

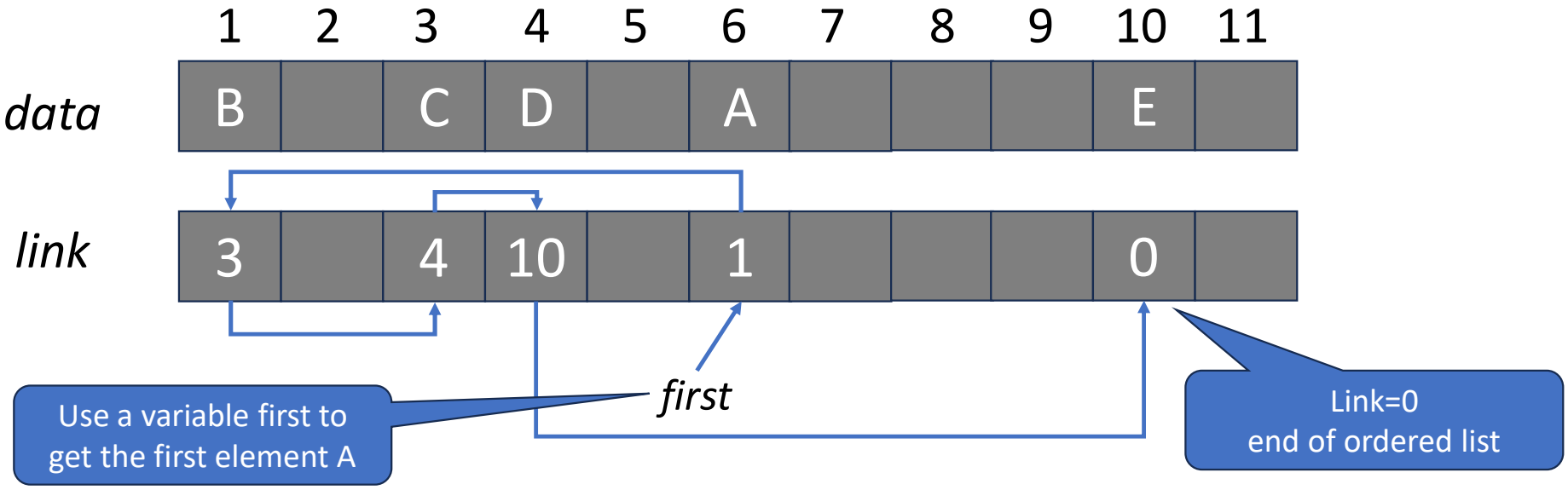
How do we know the order?

Successive items may be placed anywhere in memory.

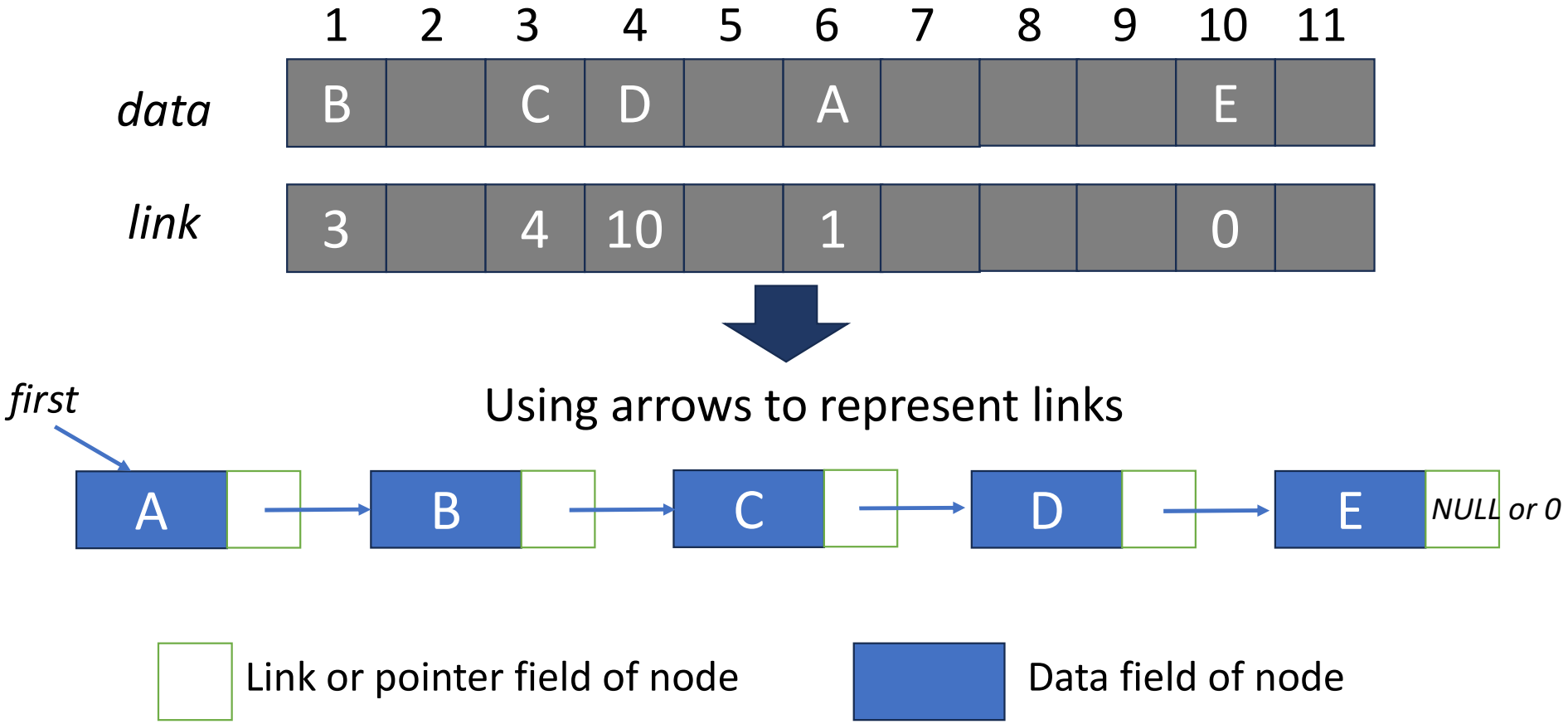
Linked list

- In memory, list elements are stored in an arbitrary order.

- **Link** is used to point the next element.

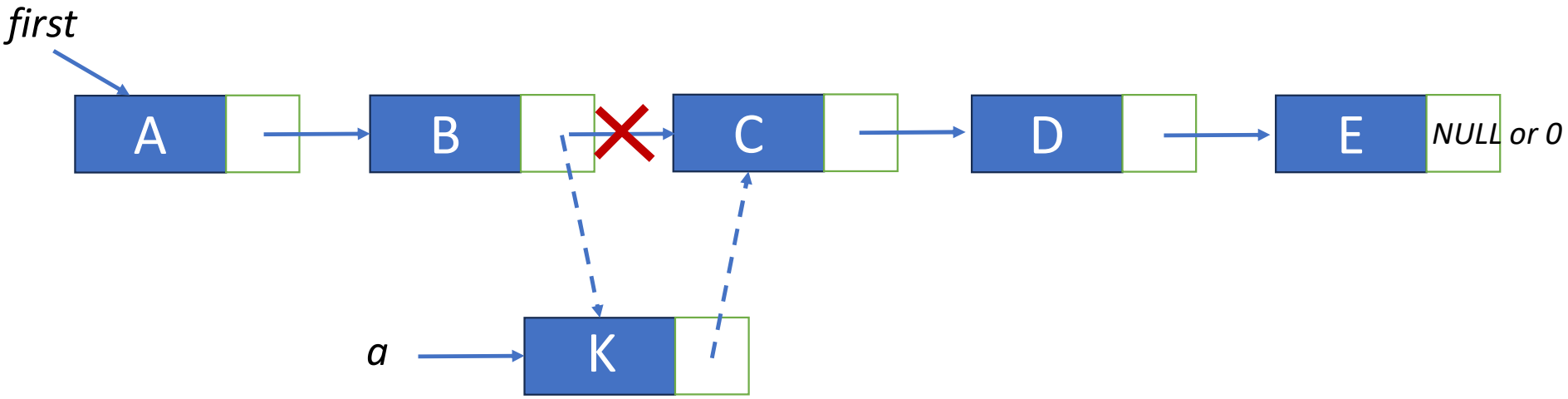


Normal way to draw a linked list



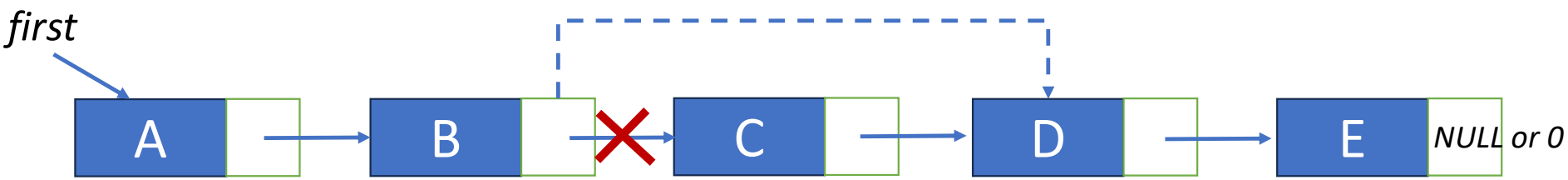
Inserting K between B and C

- Get an unused node *a*.
- Set the data field of *a* to K.
- Set the link of *a* to point to C.
- Set the link of B to point to *a*.



Deleting C

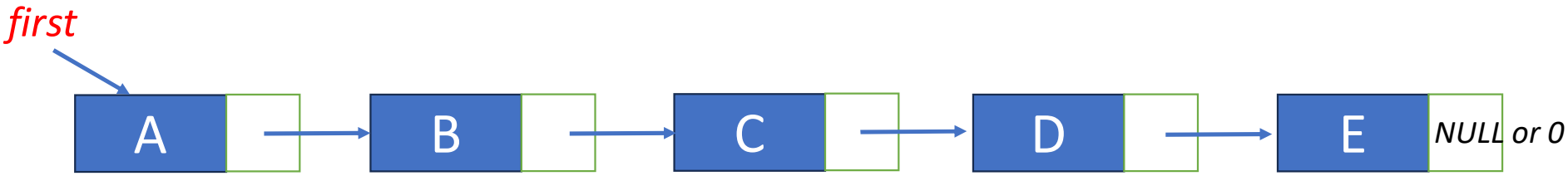
- Find the element precedes C.
- Set the link of the element to the position of D.



Representing node in C

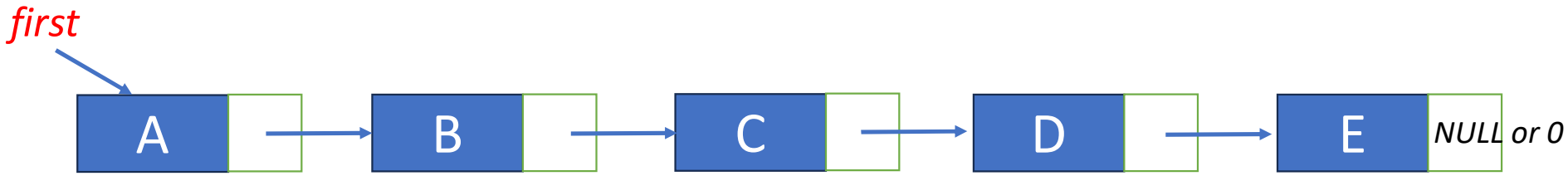
```
typedef struct listNode *listPointer;
typedef struct {
        char data;
        listPointer link;
        } listNode;
```

get(0)



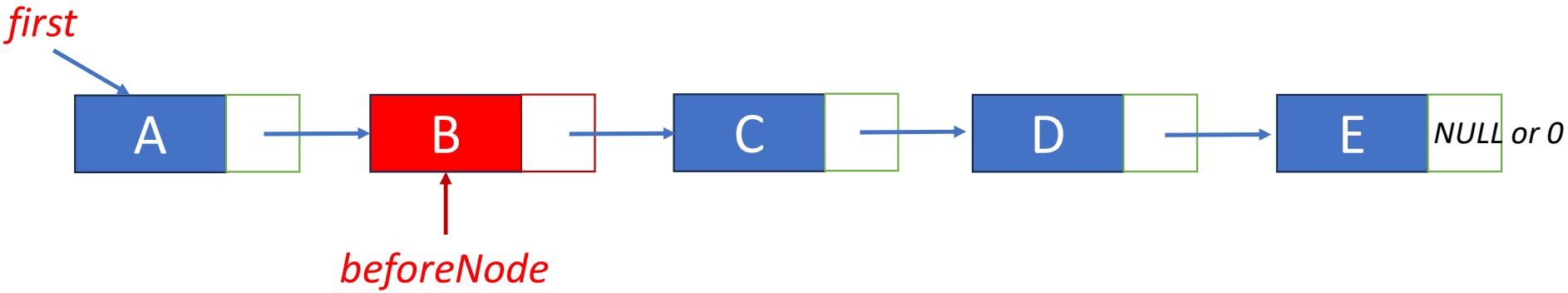

```
desiredNode = first; // gets you to first node
return desiredNode->data;
```

get(1)




```
desiredNode = first->link; // gets you to second node
return desiredNode->data;
```

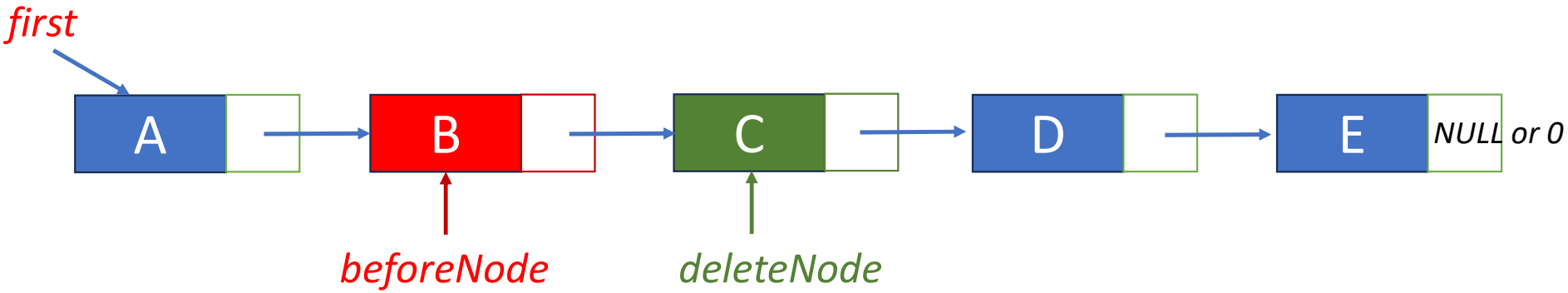
Delete “C”



- Step 1: find the node before the node to be removed

```
beforeNode = first->link;
```

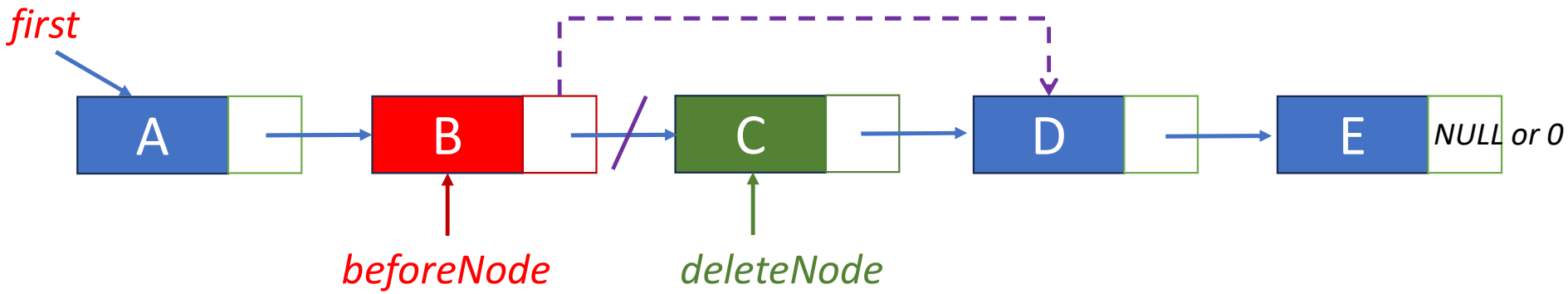
Delete “C”



- Step 2: save pointer to node that will be deleted

```
deleteNode = beforeNode->link;
```

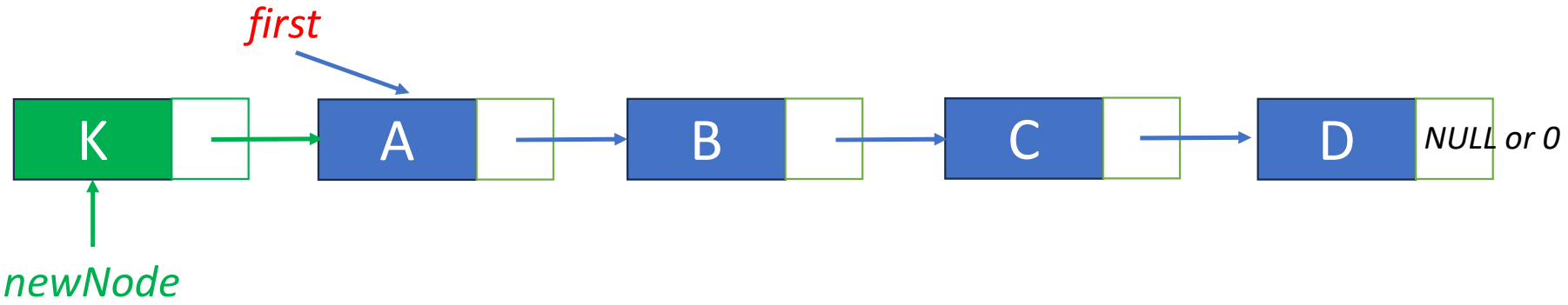
Delete “C”



- Step 3: change pointer in beforeNode

```
beforeNode->link = beforeNode->link->link;
free(deleteNode);
```

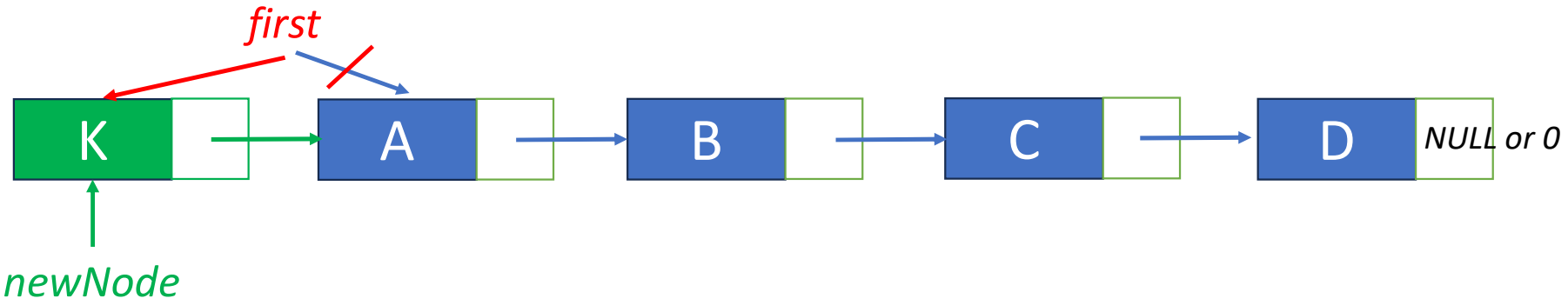
Insert “K” before “A”



- Step 1: get an unused node, set its data and link fields

```
MALLOC( newNode, sizeof(*newNode));
newNode->data = 'K';
newNode->link = first;
```

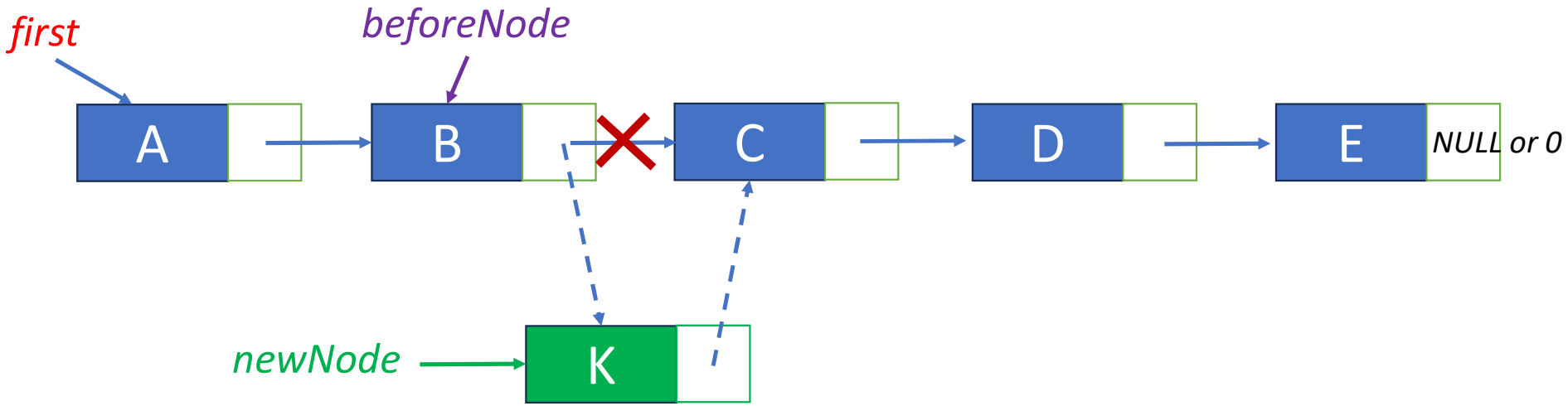
Insert “K” before “A”



- Step 2: update first

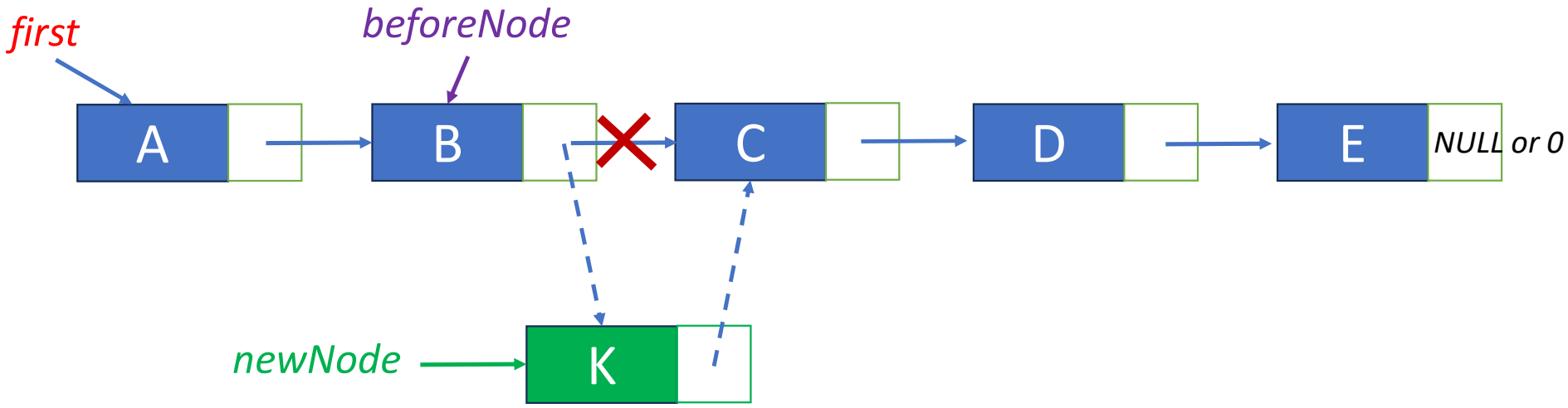
```
first = newNode
```

Insert “K” after “B”



- Find node “B”
- Create a new node and set its data and link fields
- Link beforeNode and newNode

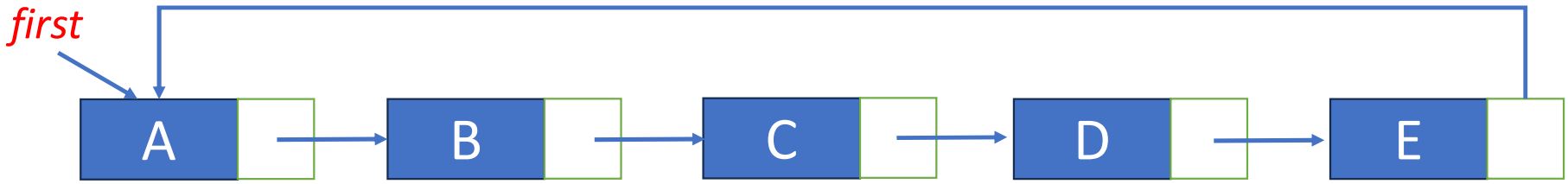
Insert “K” after “B”




```
beforeNode = first->link;
MALLOC( newNode, sizeof(*newNode));
newNode->data = 'K';
newNode->link = beforeNode->link;
beforeNode->link = newNode;
```

More advanced linked lists: Circular List

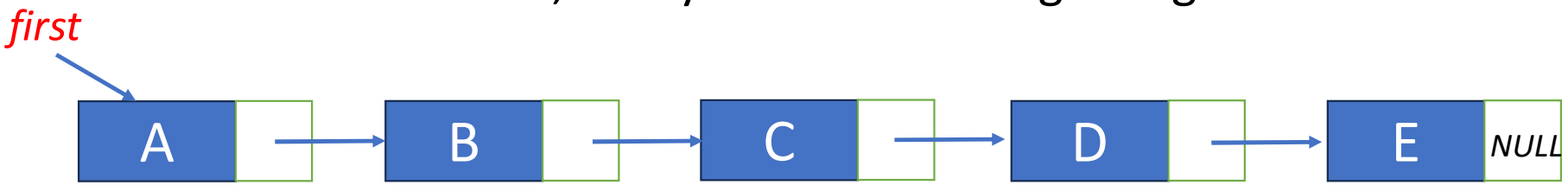
- The last node points to the first node.



Doubly Linked List

- **Singly linked list**

To find an element, always start at the beginning of the list.



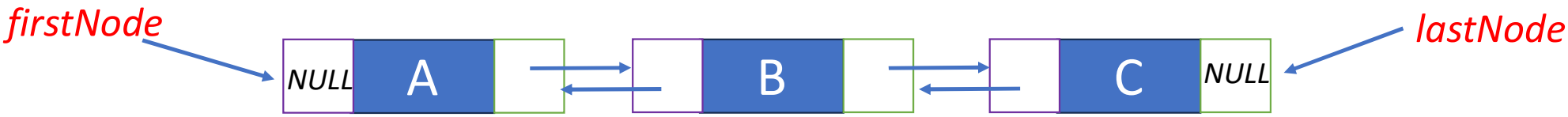
Delete "D": start at the beginning to find preceding node.

- **Doubly linked list**

Right link: forward direction

Left link: backward direction

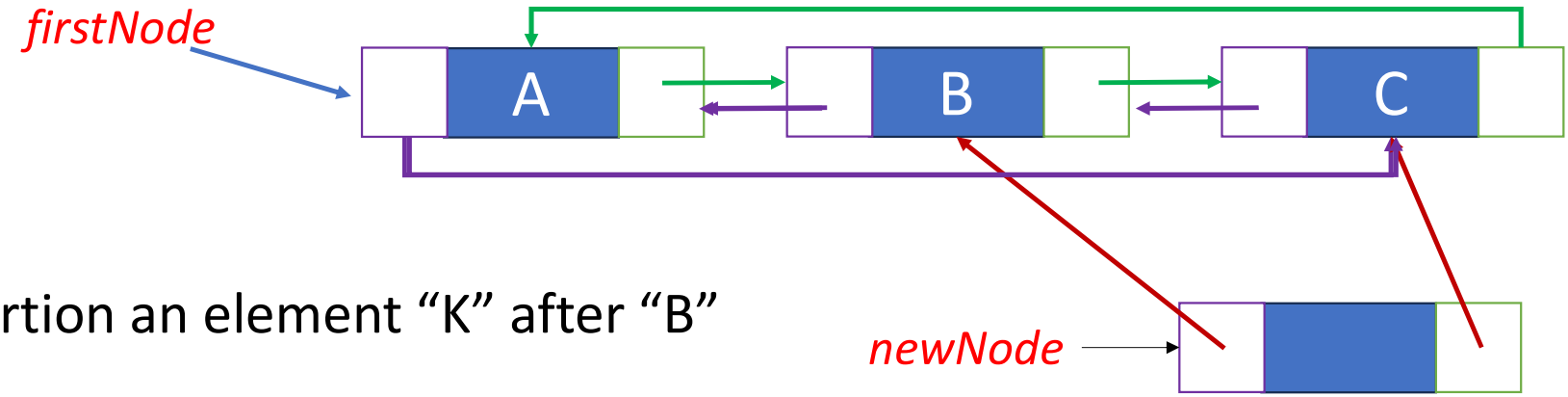
Left link Data Right link



Node in doubly linked list

```
typedef struct node *nodePointer;
typedef struct node{
        nodePointer llink;
        element data;
        nodePointer rlink;
        };
```

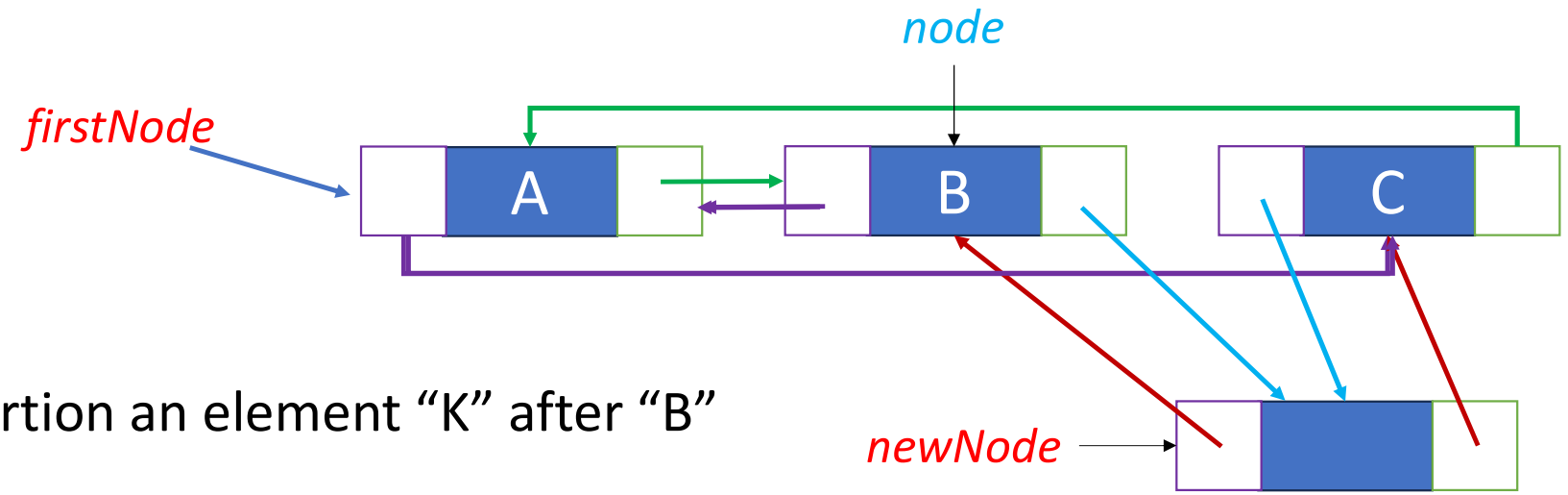
Doubly circular linked list: Insertion (1)



- Insertion an element “K” after “B”

```
newNode->llink = node;
newNode->rlink = node->rlink;
```

Doubly circular linked list: Insertion (2)

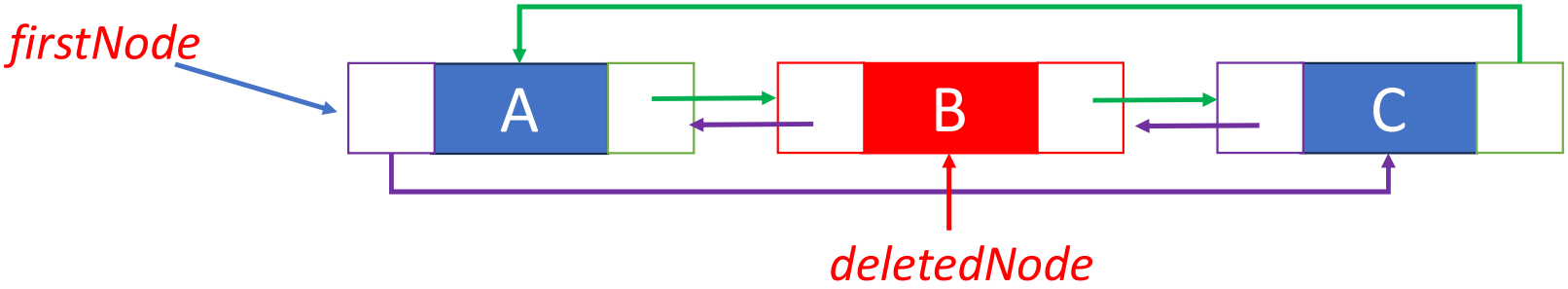


- Insertion an element “K” after “B”

```
newNode->llink = node;
newNode->rlink = node->rlink;

node->rlink->llink = newNode;
node->rlink = newNode;
```

Doubly circular linked list: Deletion



- Deleting the node containing "B" (*deletedNode*)
  Q5: Please describe the procedure to remove *deletedNode* from the doubly circular linked list. The node includes three fields: llink, data, and rlink.

```
deletedNode->llink->rlink = deletedNode->rlink;
deletedNode->rlink->llink = deletedNode->llink;
free(deletedNode);
```

Please reply your answers of Q5 via the following link:

Group members: 1~3 people