CS-218
DATA STRUCTURE

Dr. Hashim Yasin

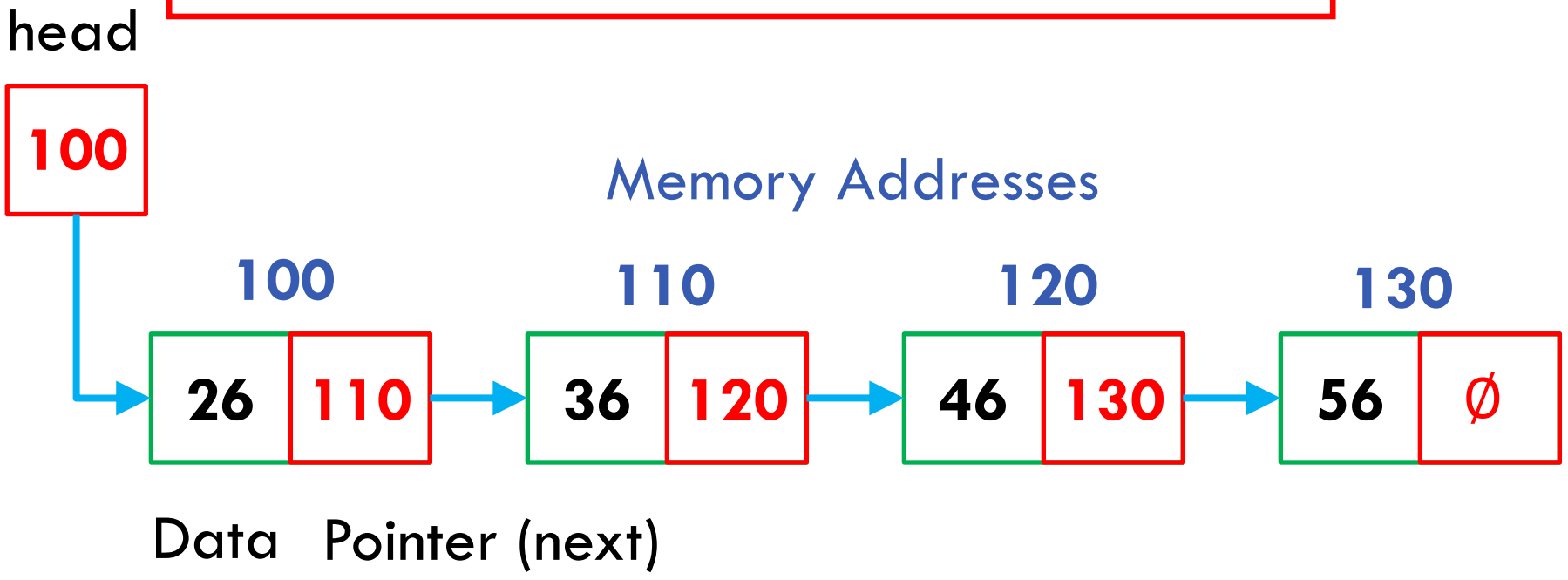
National University of Computer and Emerging Sciences,

Faisalabad, Pakistan.

DOUBLY LINKED LIST

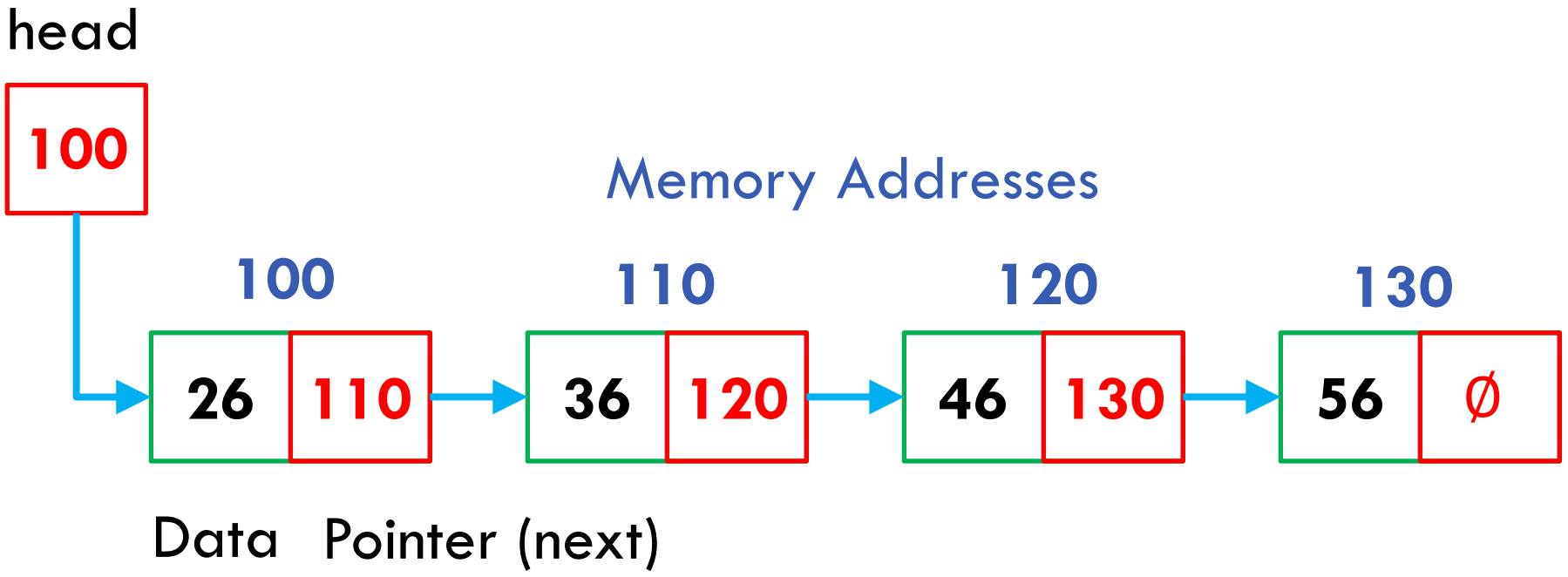
Linked List


```
class Node {
public:
        double data;            // data
        Node*  next;            // pointer to next
};
```



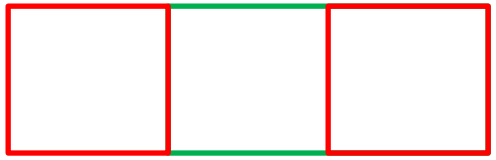
Linked List


```
struct Node {
        int    data;          // data
        struct Node*  next;   // pointer to next
};
```



Doubly Linked List

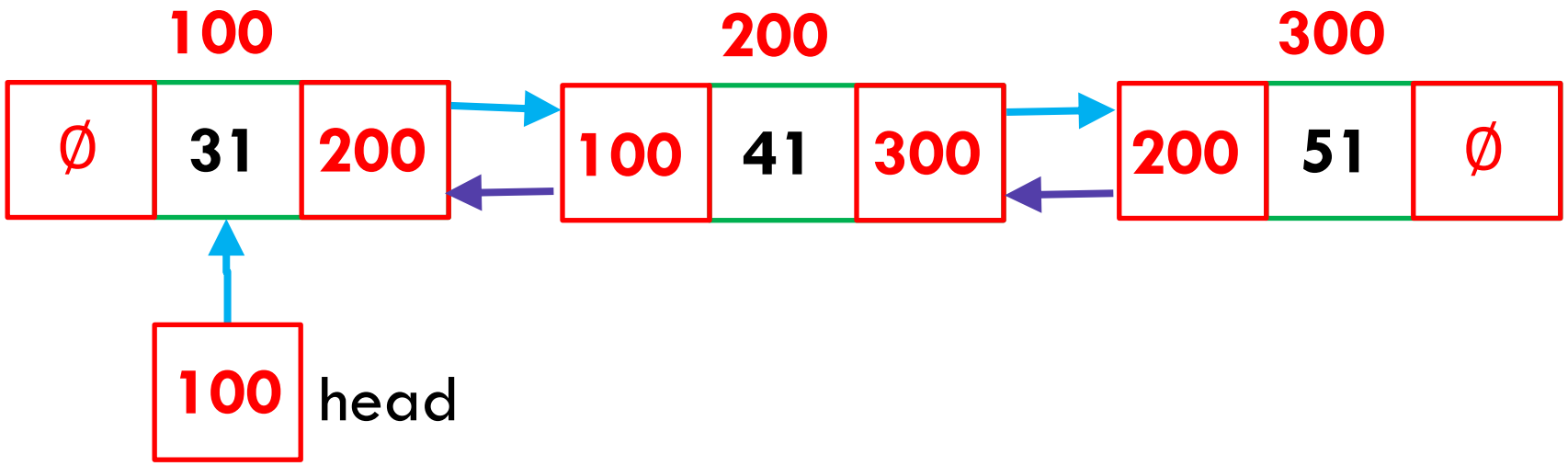

```
struct Node {
        int    data;          // data
        struct Node*  next;   // pointer to next
        struct Node*  prev;   // pointer to prev
};
```



Pointer (prev) Data Pointer (next)

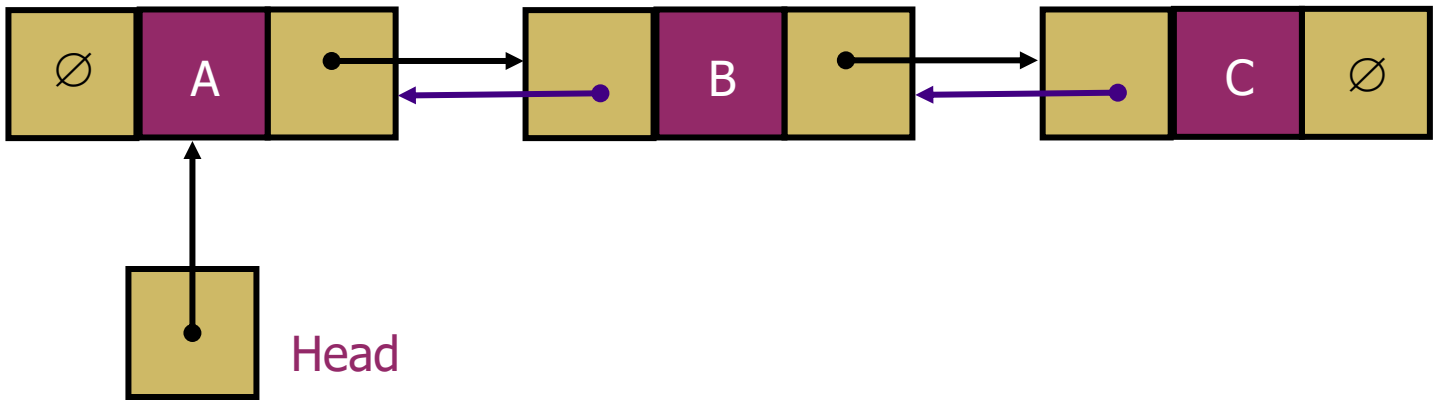
Doubly Linked List


```
struct Node {
        int    data;          // data
        struct Node*  next;   // pointer to next
        struct Node*  prev;   // pointer to prev
};
```



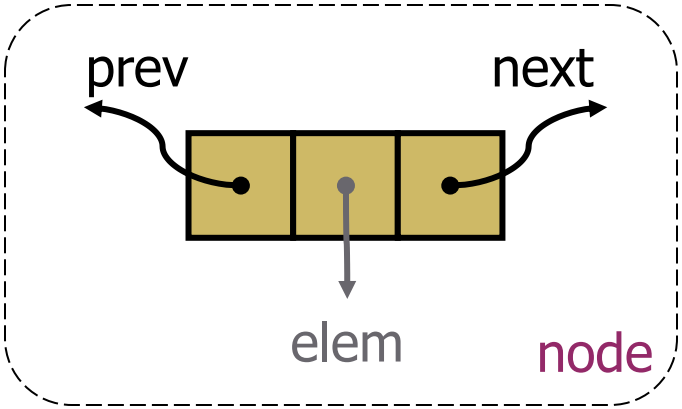
Doubly Linked List

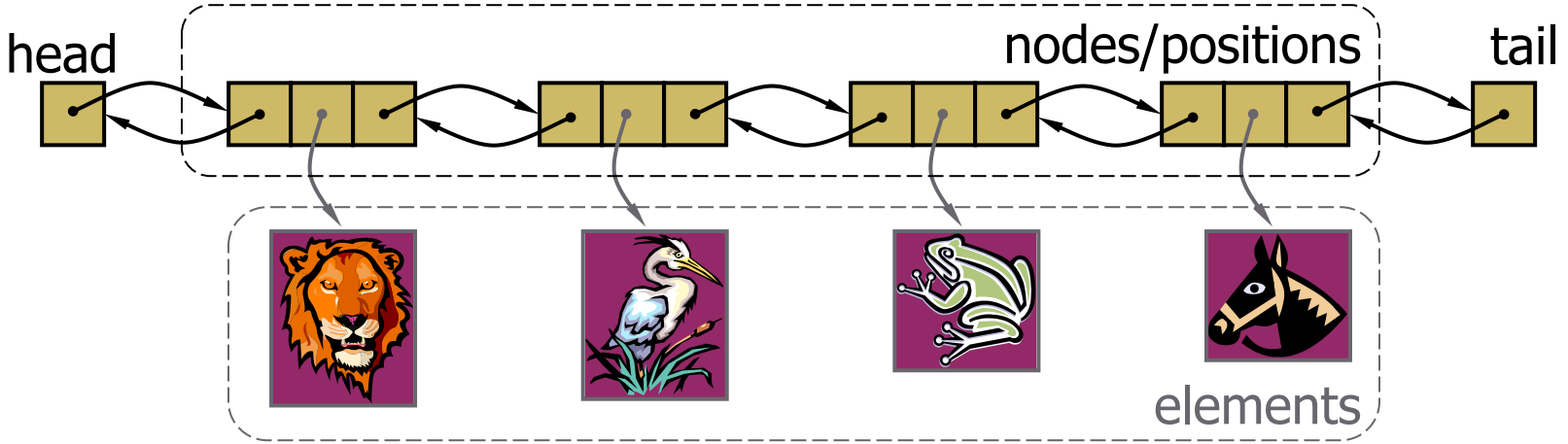
- *Each node points to not only successor but also the predecessor*
- There are two `NULLs` :
  - at the first and last nodes in the list
- Advantage: given a node, it is easy to visit its predecessor. Convenient to traverse lists backwards



Doubly Linked List

- A doubly linked list provides a natural implementation of the List ADT
- Nodes implement Position and store:
  - **element**
  - **link to the previous node**
  - **link to the next node**
- Special tail and head nodes





Doubly Linked List … Sentinels

- To simplify programming, *two special nodes have been added at both ends of the doubly-linked list*.
- Head and tail are dummy nodes, also called sentinels, do not store any data elements.
- **Head:** head sentinel has a ***null-prev*** reference (link).
- **Tail (Last):** tail sentinel has a ***null-next*** reference (link).

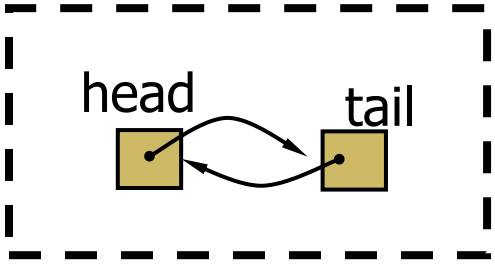
Doubly Linked List

Empty Doubly-Linked List:

Using sentinels, we have no null-links; instead, we have:

head.next = tail

tail.prev = head

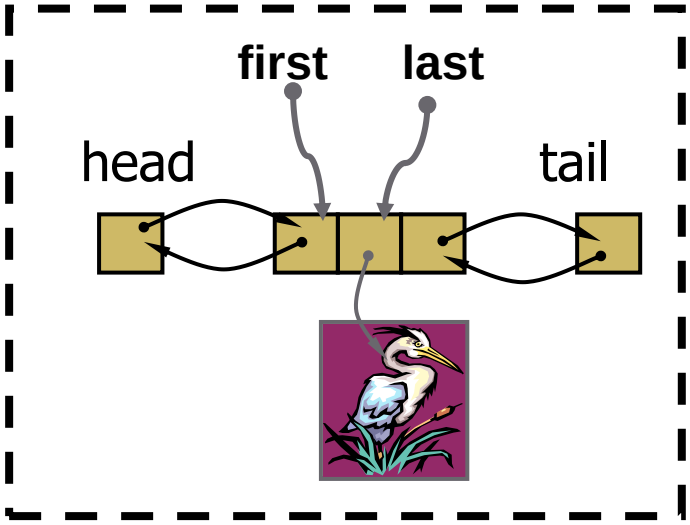


Single Node List: Size = 1

This single node is the first node, and also is the last node:

first node is head.next

last node is tail.prev



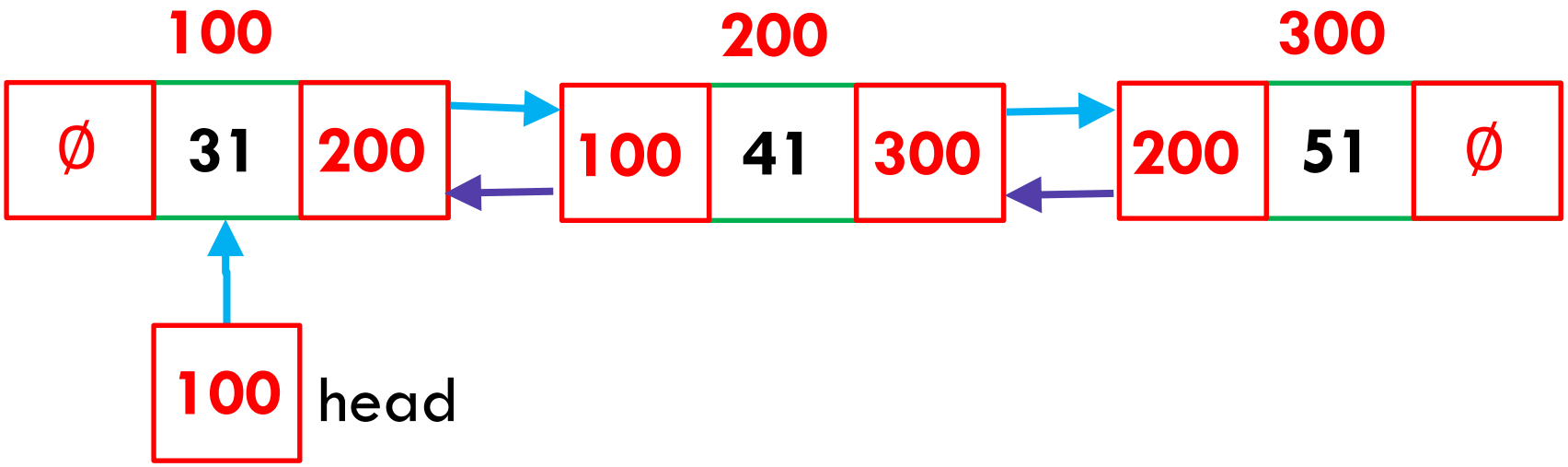
Why Doubly Linked List

- A DLL can be traversed in both forward and backward direction.
- The delete operation in DLL is more efficient if pointer to the node to be deleted is given.
- Similarly, we can quickly insert a new node before a given node.

Example: Deletion in DLL

The delete operation in DLL is more efficient if pointer to the node to be deleted is given. For example, a pointer to the specific node is given,

```
temp // 200 pointer to the node to be deleted
temp->next    // 300
temp->prev    // 100
```



Example: Deletion in DLL

- In **singly linked list**, we need two pointers in order to delete the node,
  - One pointer to the node to be deleted
  - Second pointer to the previous node

- In **doubly linked list**, we need just only one pointer to the node to be deleted. As a result, reverse look up is very useful.

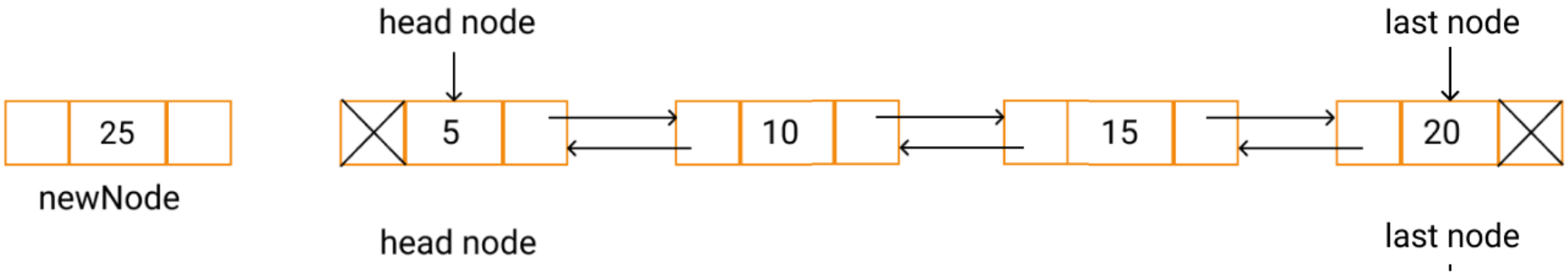
Disadvantages of DLL

- Every node of DLL requires extra memory space for a previous pointer **"prev"** .
- All operations require an extra pointer **"prev"** to be maintained.
- For example:
  - A linked list of integer, integer takes 4 bytes, pointer also takes 4 bytes.
  - In singly linked list, each node is of 8 bytes, while for doubly linked list every node is of 12 bytes.

OPERATIONS ON DLL

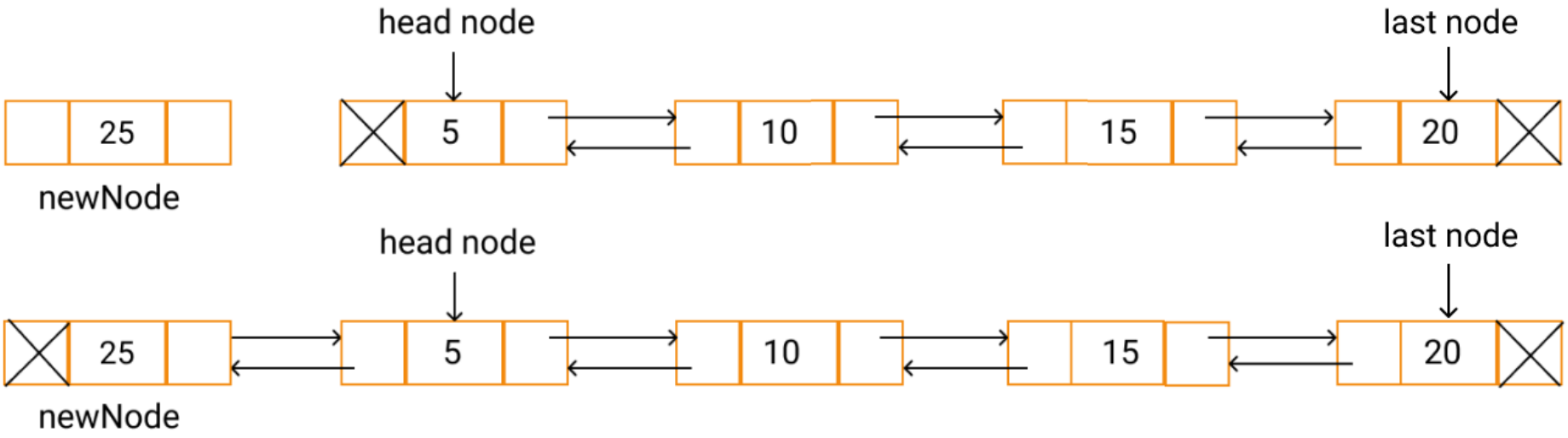
Insertion in DLL (1)

Insertion at Start



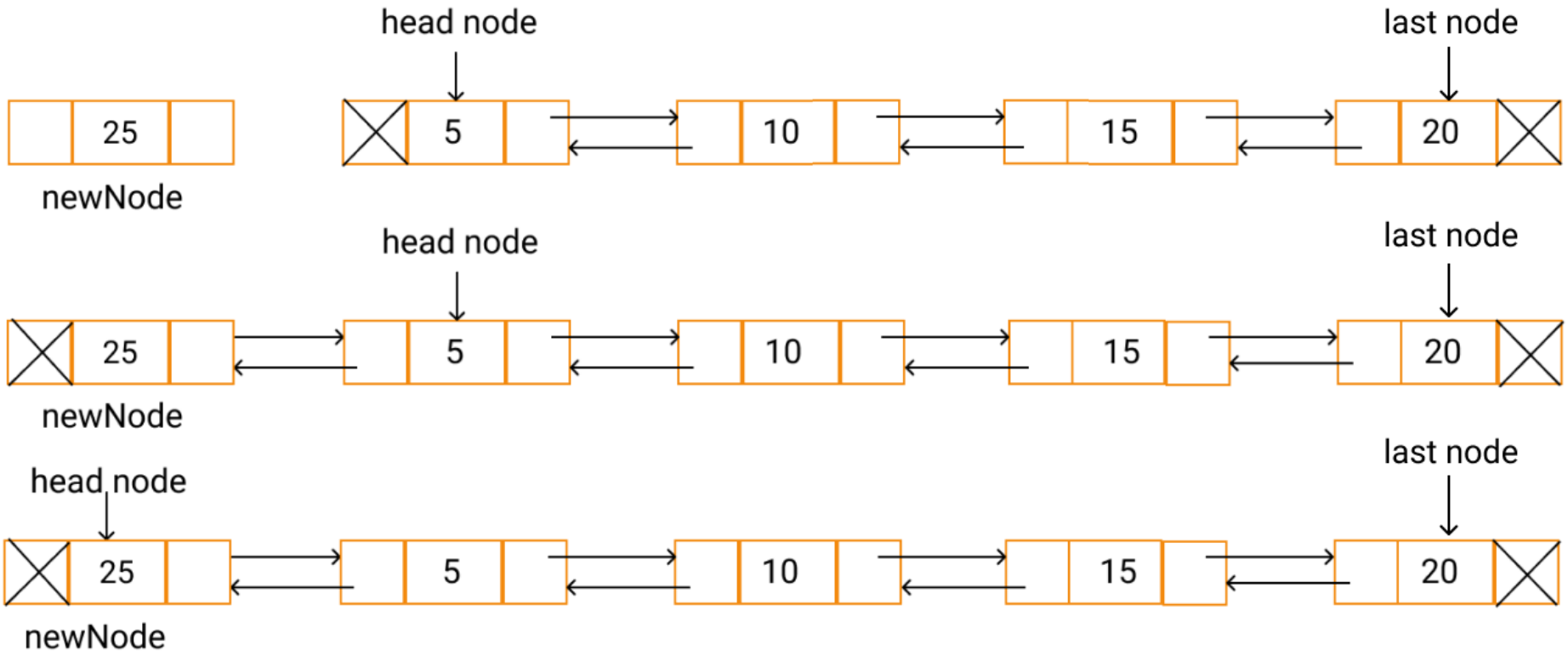
Insertion in DLL (1)

Insertion at Start

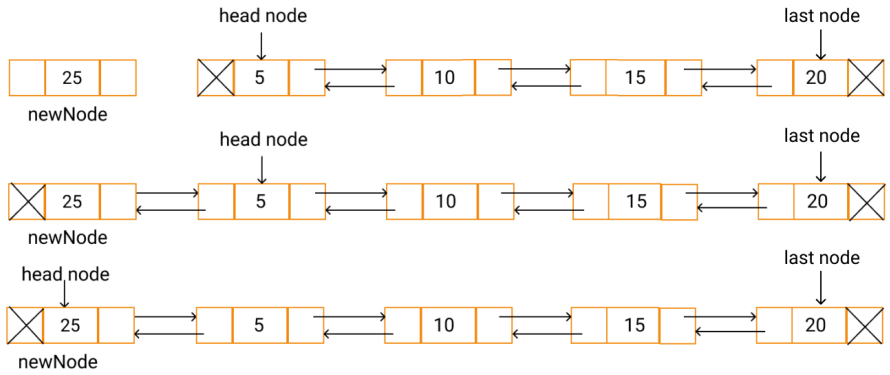


Insertion in DLL (1)

Insertion at Start



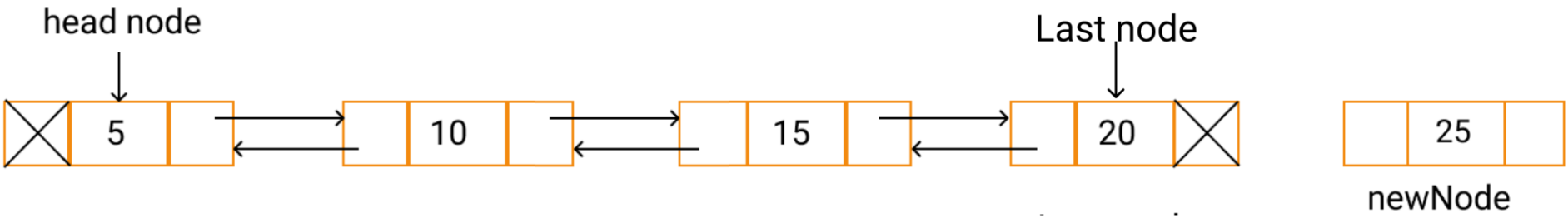
Insertion in DLL (1)




```
void insertAtStart(int data){
struct node * newNode;
if(head == NULL) {
   printf("Error, List is Empty!\n");
}
else {
   newNode->data = data;
   newNode->next = head; // Point to next node which is currently head
   newNode->prev = NULL; // Previous node of first node is NULL
   /* Link previous address field of head with newNode */
   head->prev = newNode;
   /* Make the new node as head node */
   head = newNode;
}
}
```

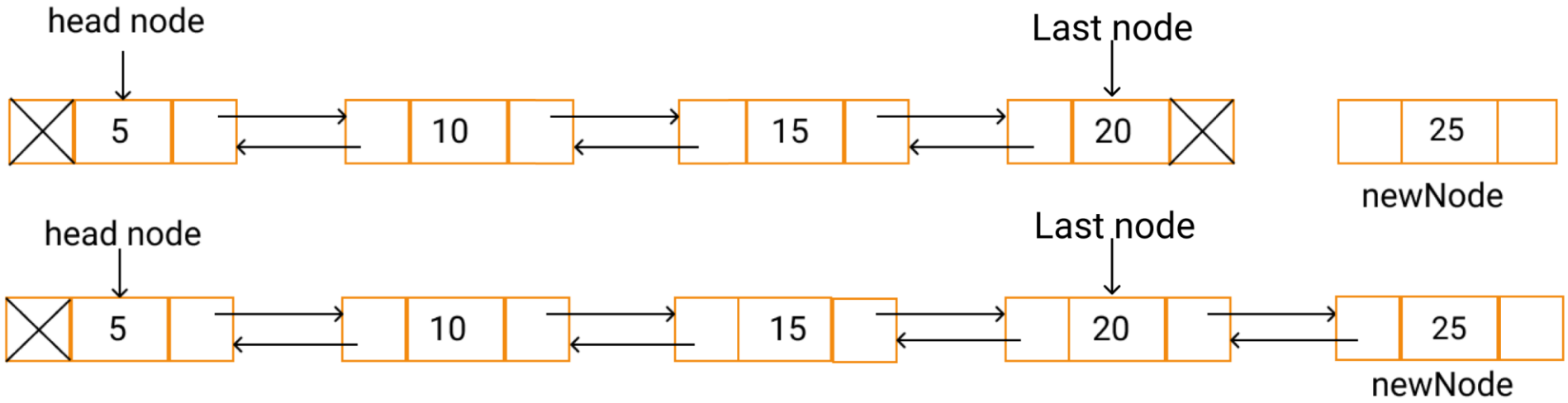
Insertion in DLL (2)

Insertion at End



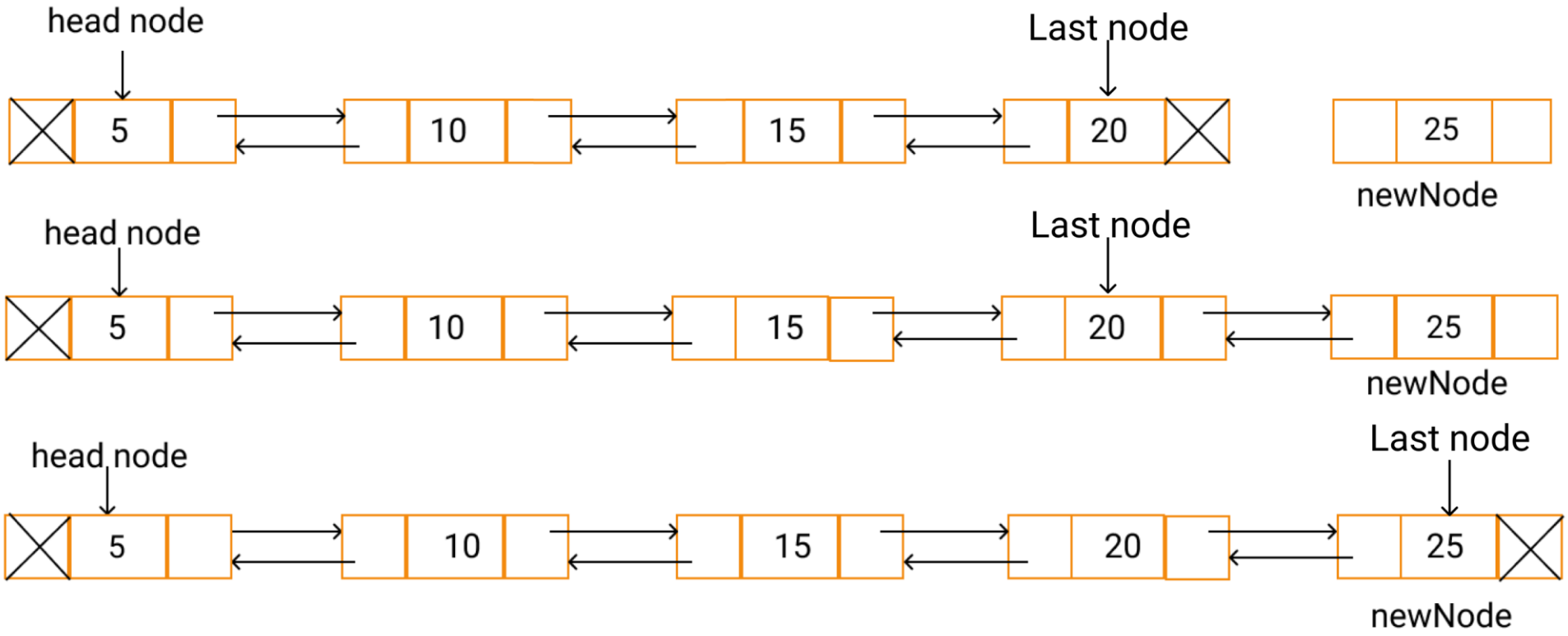
Insertion in DLL (2)

Insertion at End

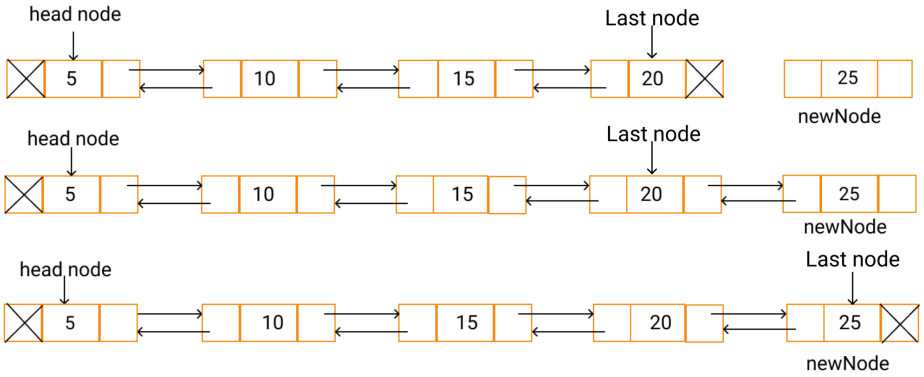


Insertion in DLL (2)

Insertion at End

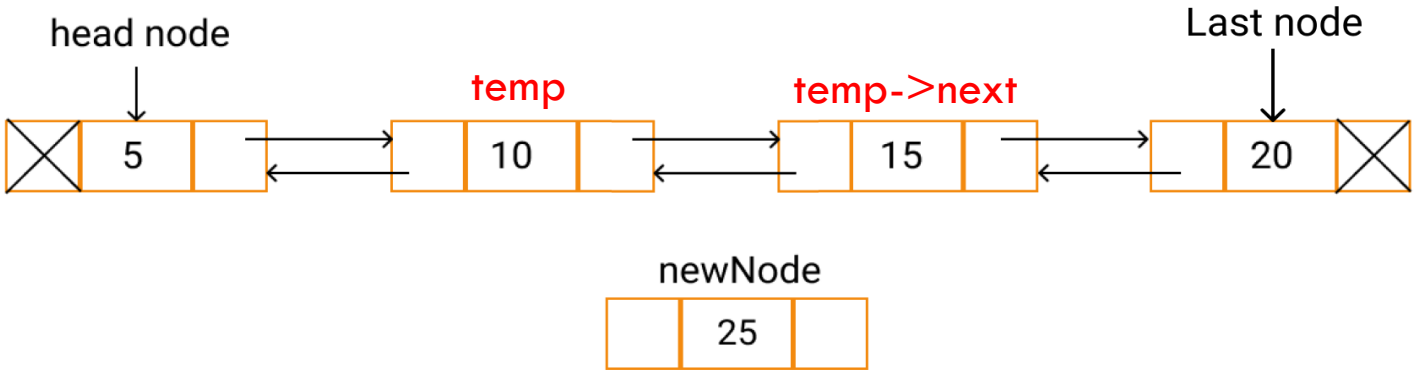


Insertion in DLL (2)

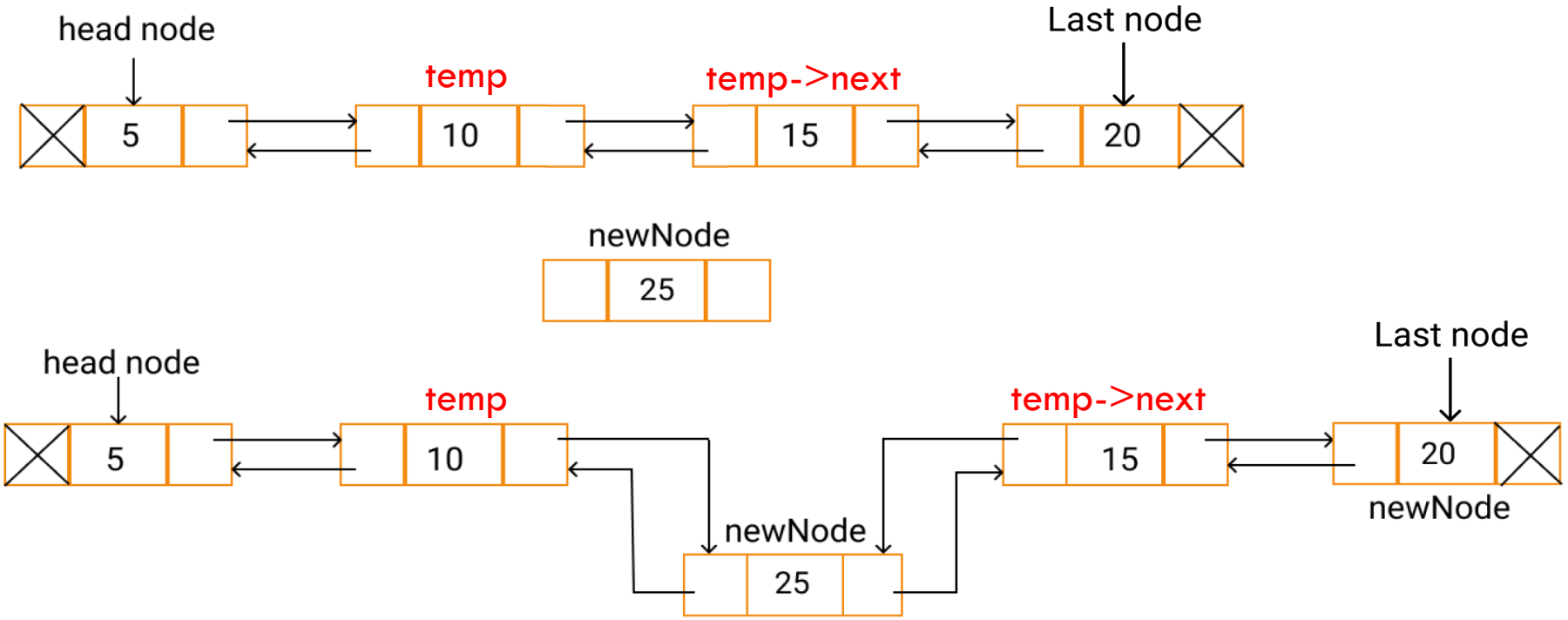



```
void insertAtEnd(int data){
struct node * newNode;
if(last == NULL) {
   printf("Error, List is Empty!\n");
}
else {
   newNode->data = data;
   newNode->next = NULL; // next pointer of newNode is NULL
   newNode->prev = last; // prev pointer of newNode is referenced to
                         //  the last node
   /* next pointer of the last node is referenced to the newNode*/
   last->next = newNode;
   /* newNode is made as the last node */
   last = newNode;
}
}
```

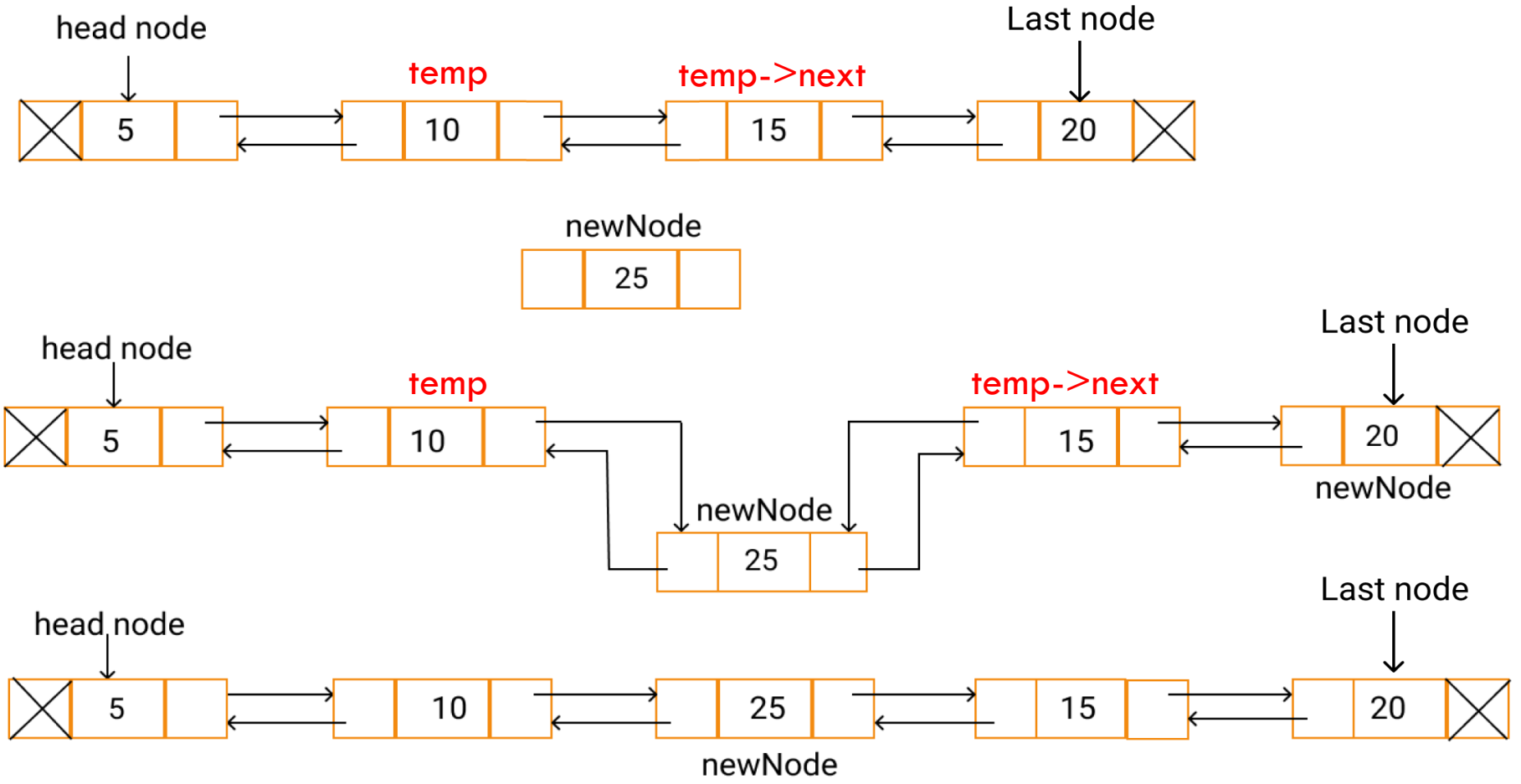
Insertion in DLL (3)



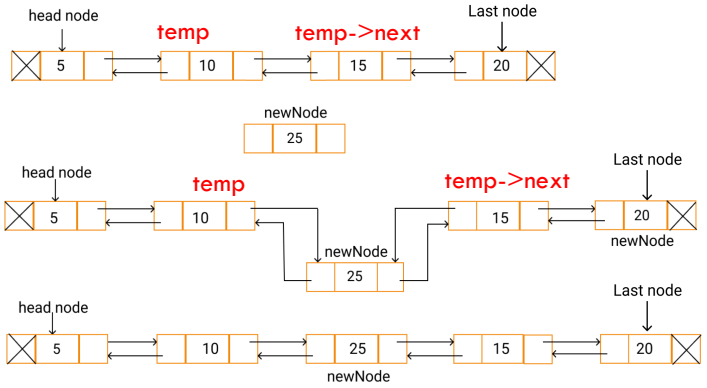
Insertion in DLL (3)



Insertion in DLL (3)

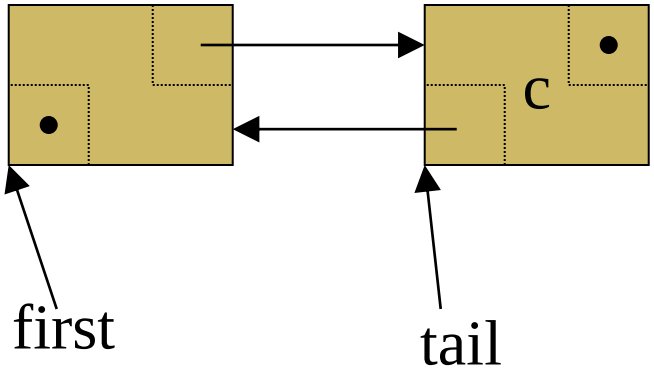


```
void insertGivenPos(int data, int position){
int i;
struct node * newNode, *temp;
if(head == NULL) { printf("Error, List is empty!\n"); }
else {
  temp = head; i=0;
  while(i<position-1 && temp!=NULL){
    temp = temp->next;
    i++;
  }
  if(temp!=NULL){
    newNode->data = data;
    newNode->next = temp->next;   // Connect new node with n+1 node
    newNode->prev = temp;         // Connect new node with n-1 node
    if(temp->next != NULL){
      temp->next->prev = newNode;// Connect pos+1 node with new node
    }
    temp->next = newNode;         // Connect pos-1 node with new node
  }
  else {printf("Error, Invalid position\n");
}}}
```



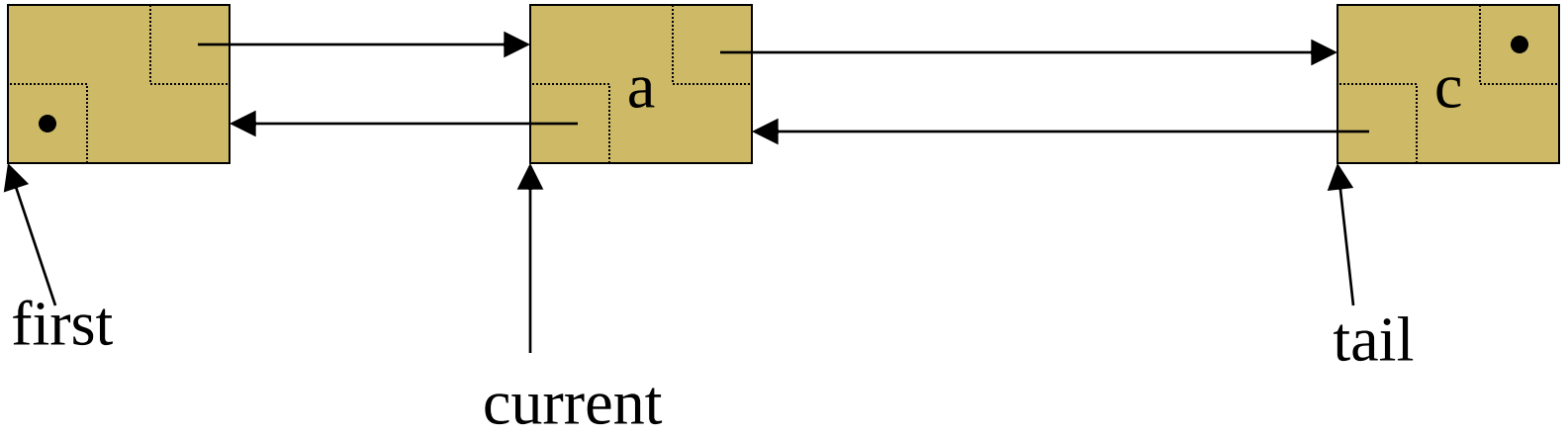
DETAILS IN INSERTIONS

Doubly Linked List



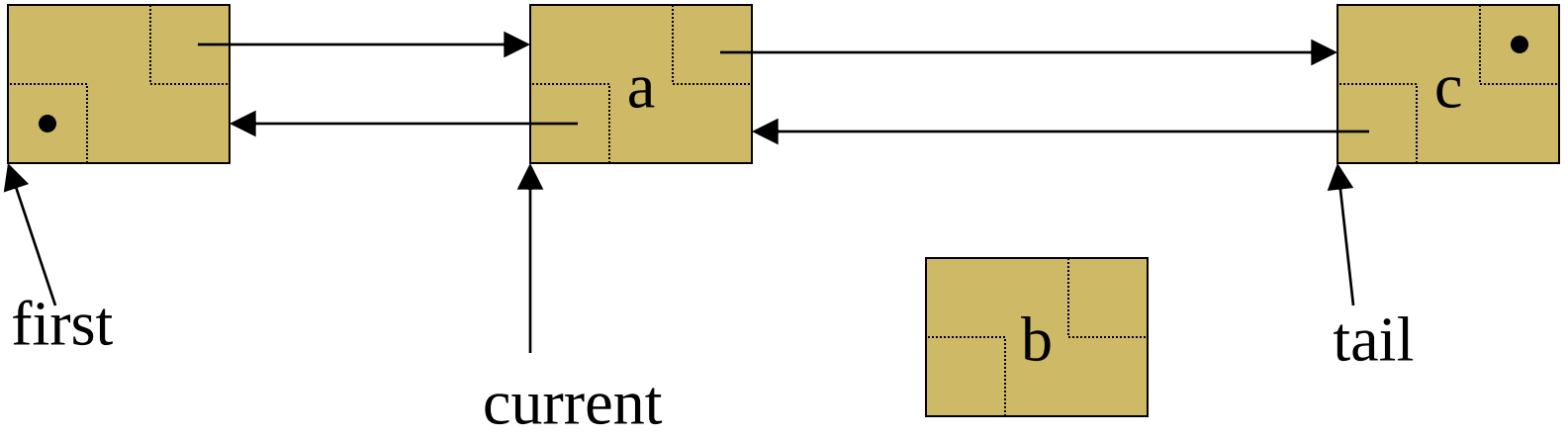

```
// constructor
DoubleList(){
    first = new DoubleListNode ();
    tail = new DoubleListNode ();
    first->next = tail;
    tail->prev = first;
}
```

Insertion



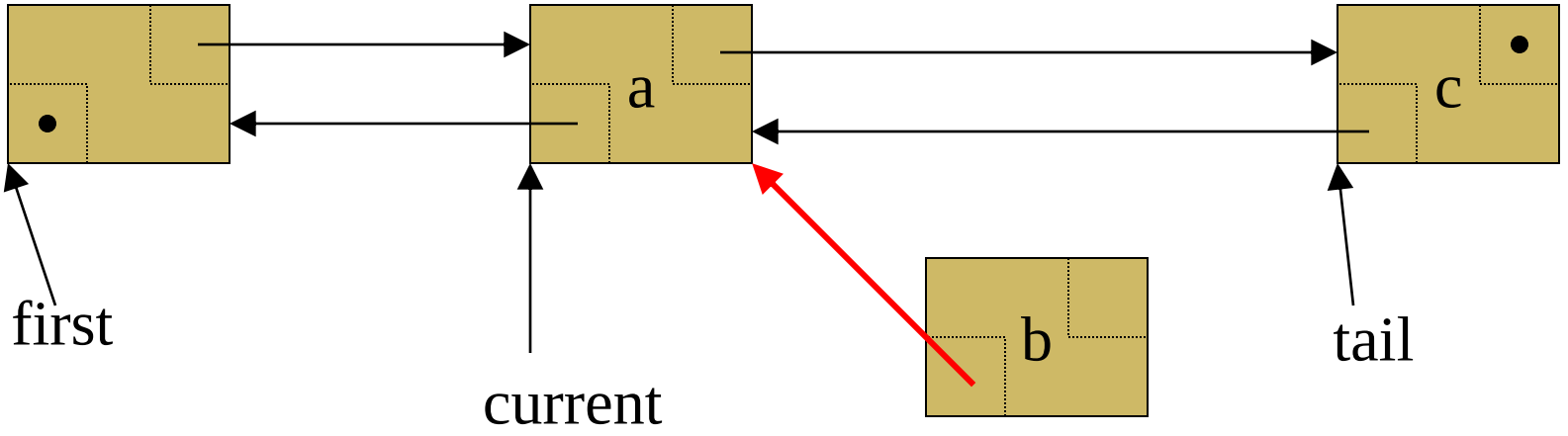

```
newNode = new DoublyLinkedListNode()
newNode->prev = current;
newNode->next = current->next;
newNode->prev->next = newNode;
newNode->next->prev = newNode;
current = newNode
```

Insertion



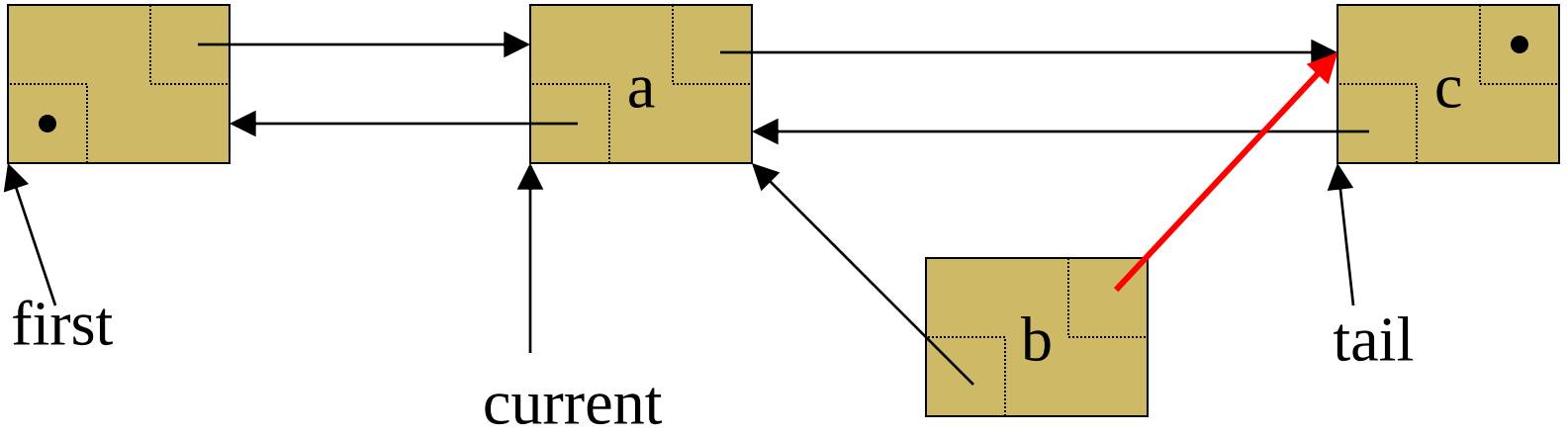

```
newNode = new DoublyLinkedListNode()
newNode->prev = current;
newNode->next = current->next;
newNode->prev->next = newNode;
newNode->next->prev = newNode;
current = newNode
```

Insertion



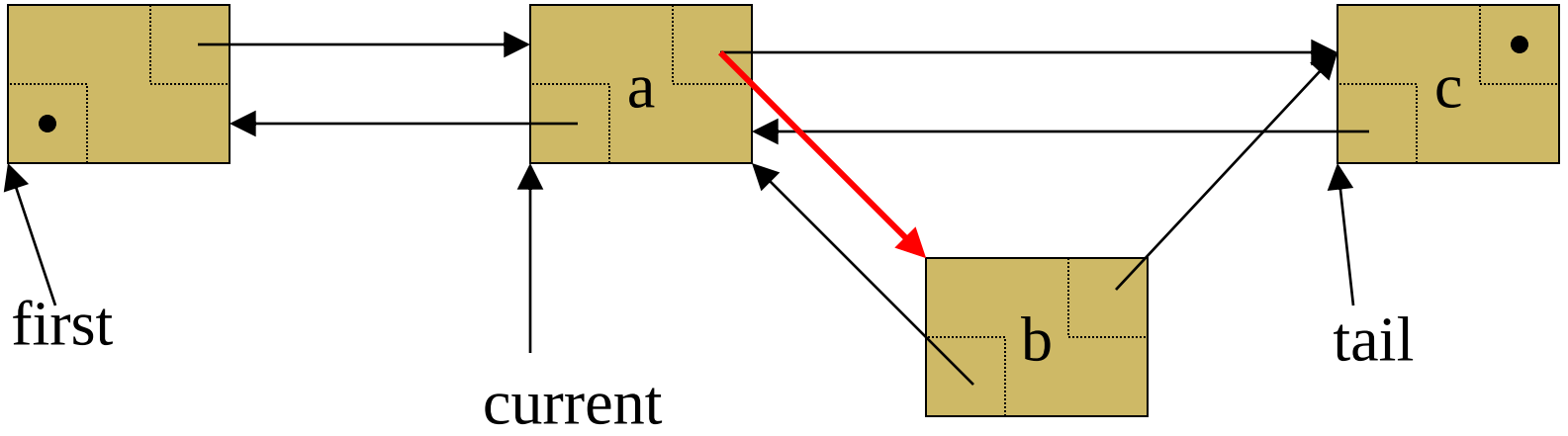

```
newNode = new DoublyLinkedListNode()
newNode->prev = current;
newNode->next = current->next;
newNode->prev->next = newNode;
newNode->next->prev = newNode;
current = newNode
```

Insertion



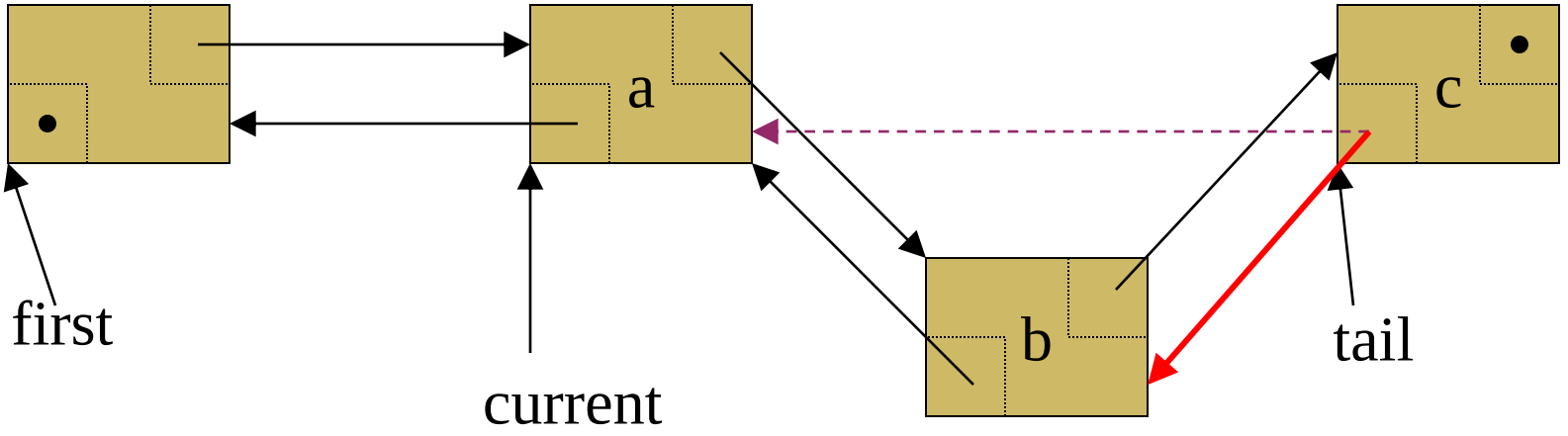

```
newNode = new DoublyLinkedListNode()
newNode->prev = current;
newNode->next = current->next;
newNode->prev->next = newNode;
newNode->next->prev = newNode;
current = newNode
```

Insertion



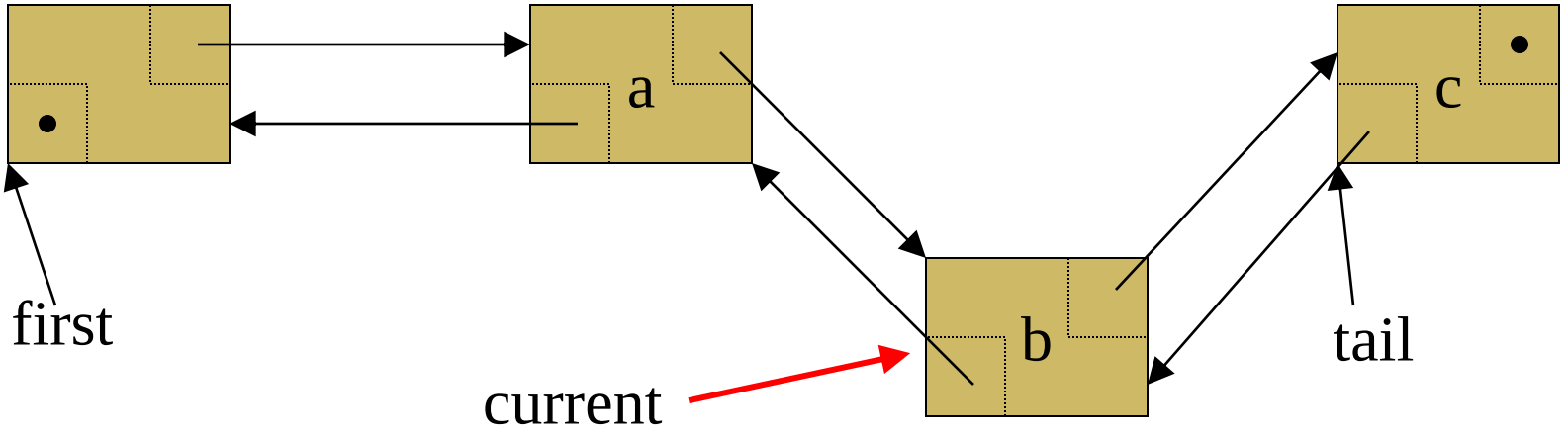

```
newNode = new DoublyLinkedListNode()
newNode->prev = current;
newNode->next = current->next;
newNode->prev->next = newNode;
newNode->next->prev = newNode;
current = newNode
```

Insertion




```
newNode = new DoublyLinkedListNode()
newNode->prev = current;
newNode->next = current->next;
newNode->prev->next = newNode;
newNode->next->prev = newNode;
current = newNode
```

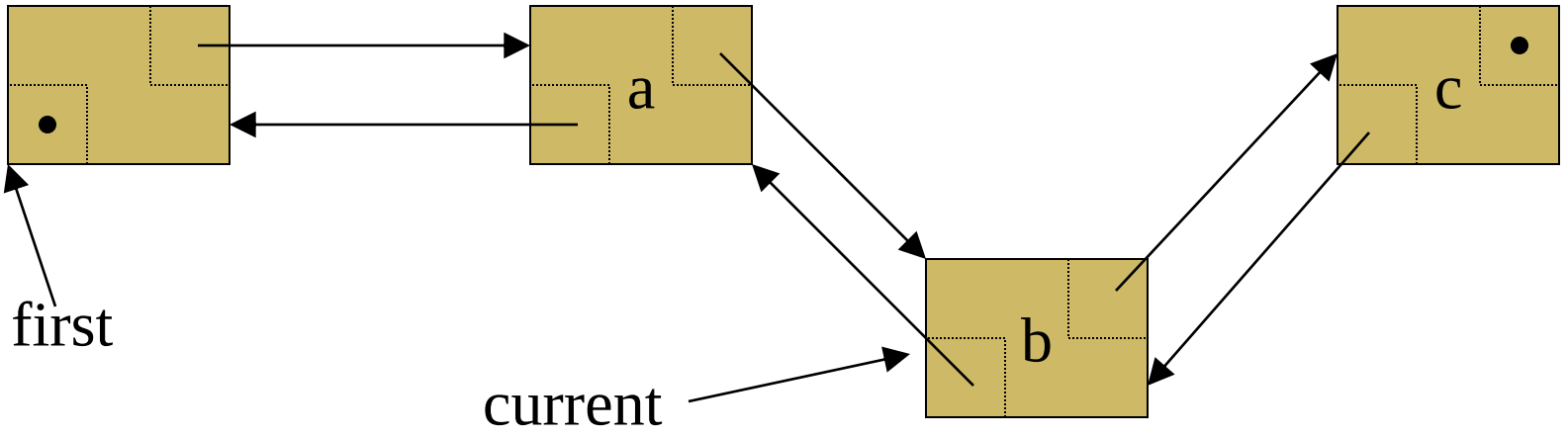
Insertion



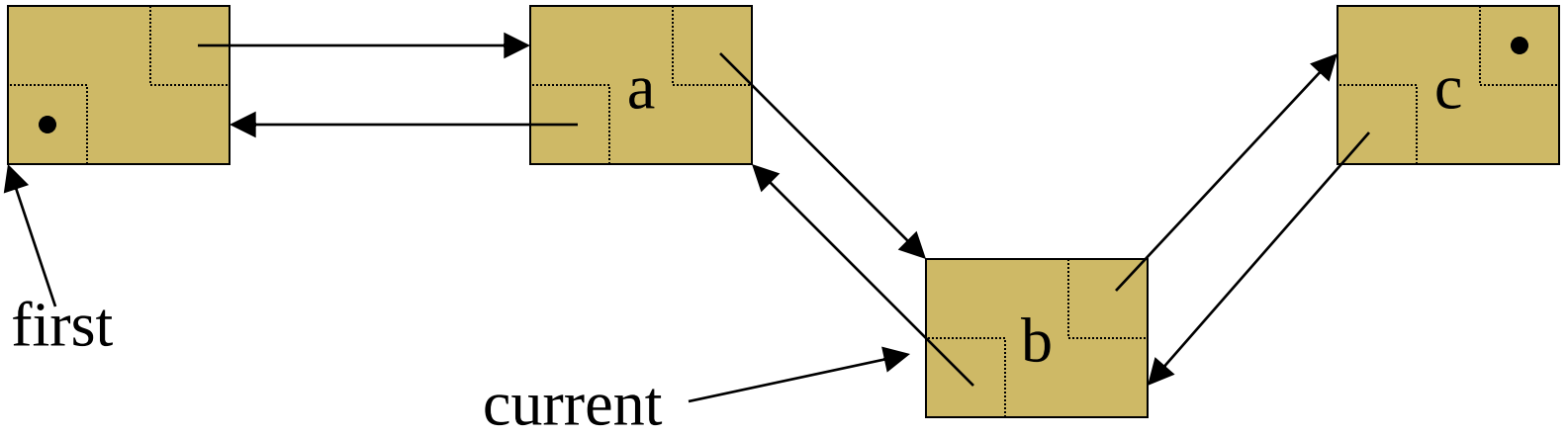

```
newNode = new DoublyLinkedListNode()
newNode->prev = current;
newNode->next = current->next;
newNode->prev->next = newNode;
newNode->next->prev = newNode;
current = newNode
```

DELETION

Deletion

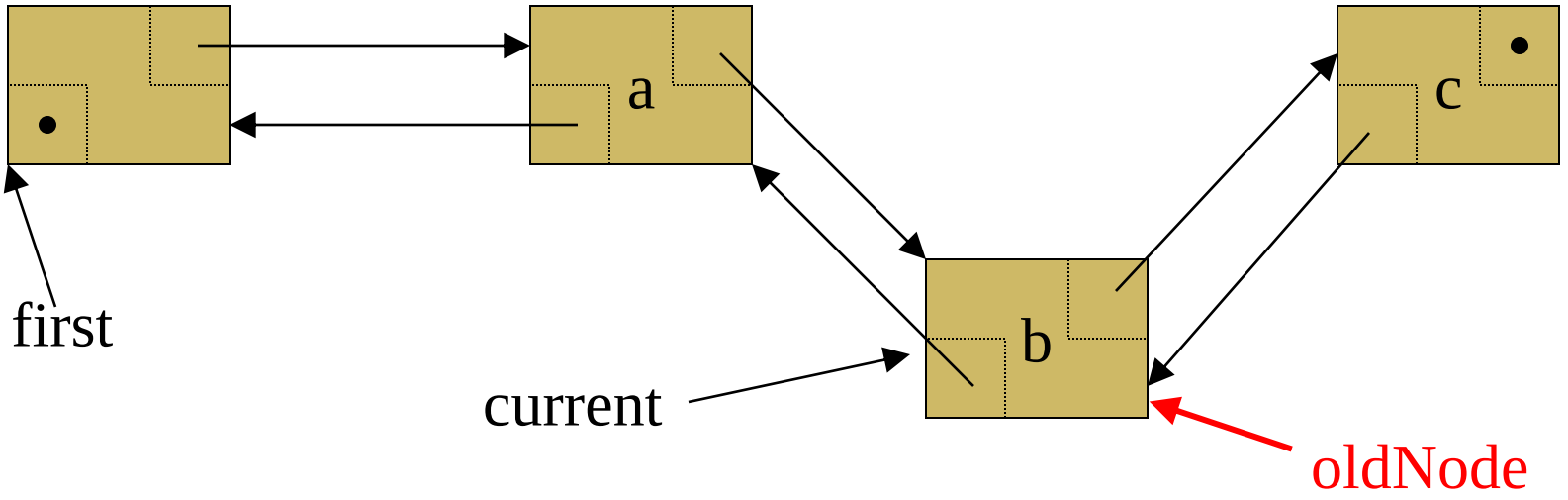


Deletion



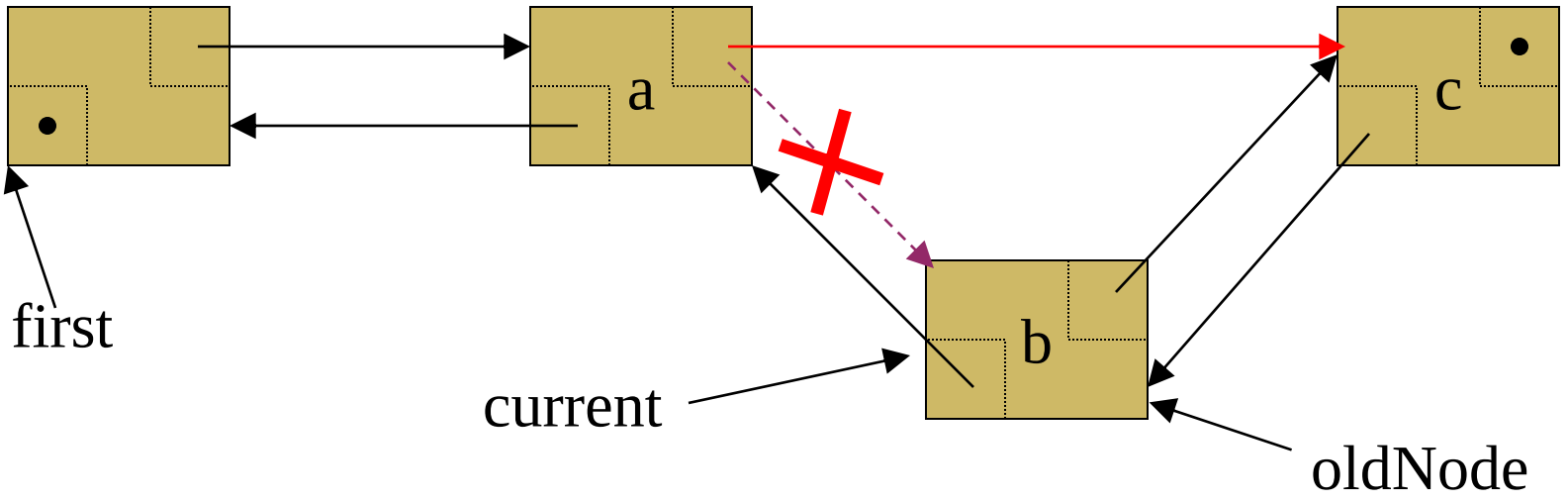

```
oldNode=current;
oldNode->prev->next = oldNode->next;
oldNode->next->prev = oldNode->prev;
current = oldNode->prev;
delete oldNode;
```

Deletion



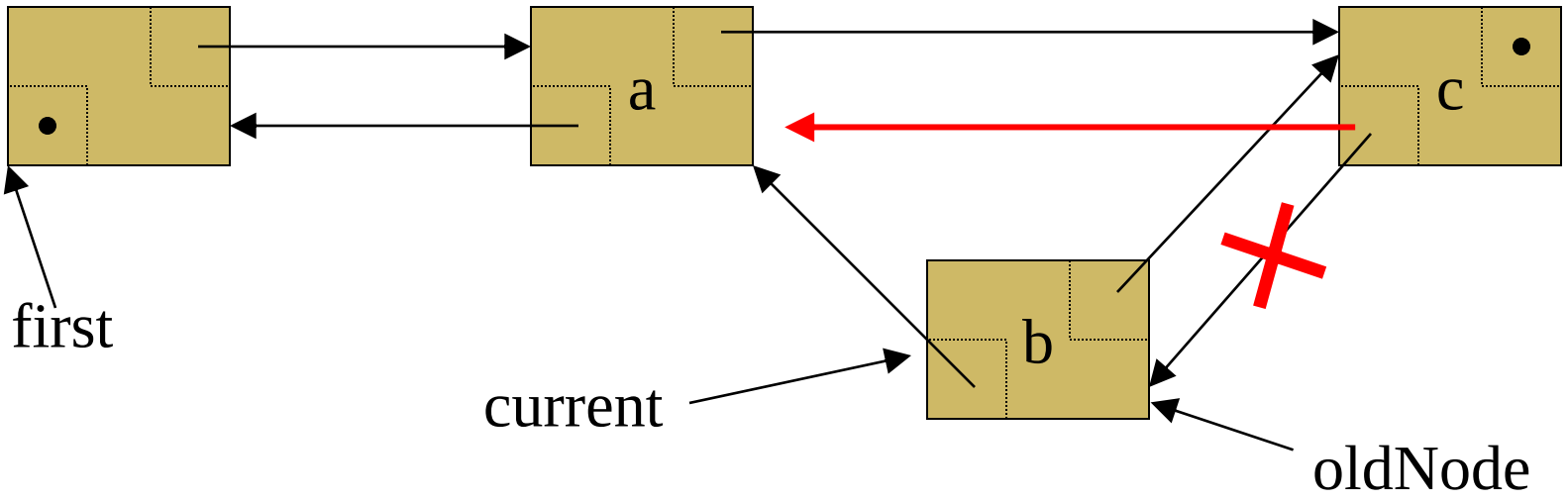

```
oldNode=current;
oldNode->prev->next = oldNode->next;
oldNode->next->prev = oldNode->prev;
current = oldNode->prev;
delete oldNode;
```

Deletion



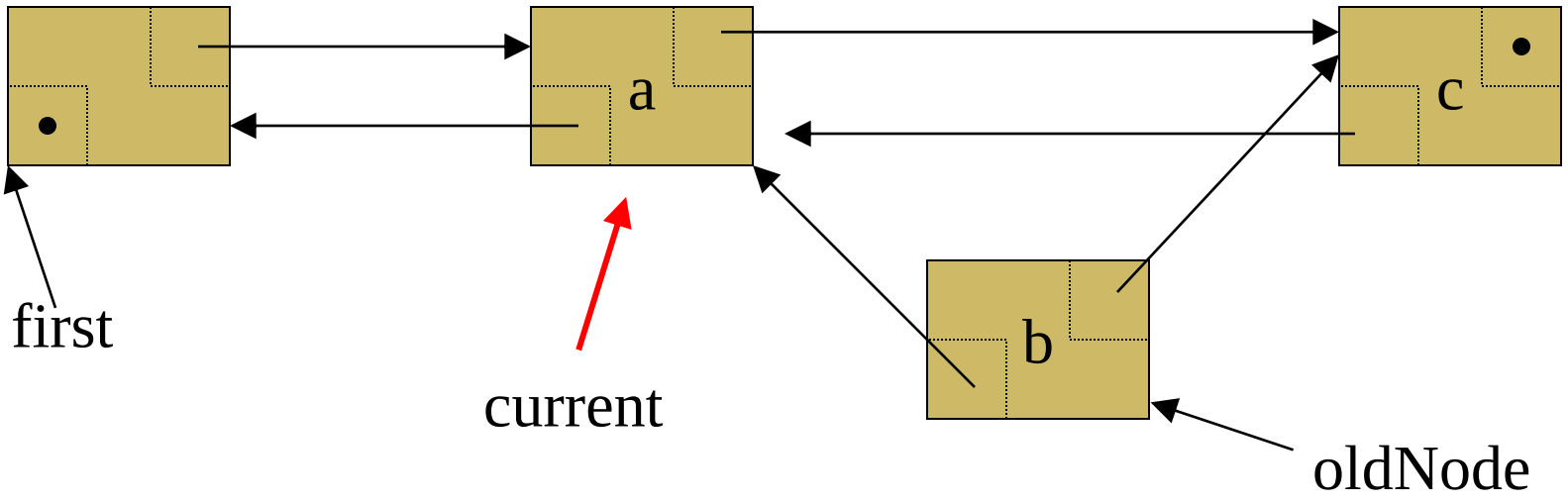

```
oldNode=current;
oldNode->prev->next = oldNode->next;
oldNode->next->prev = oldNode->prev;
current = oldNode->prev;
delete oldNode;
```

Deletion



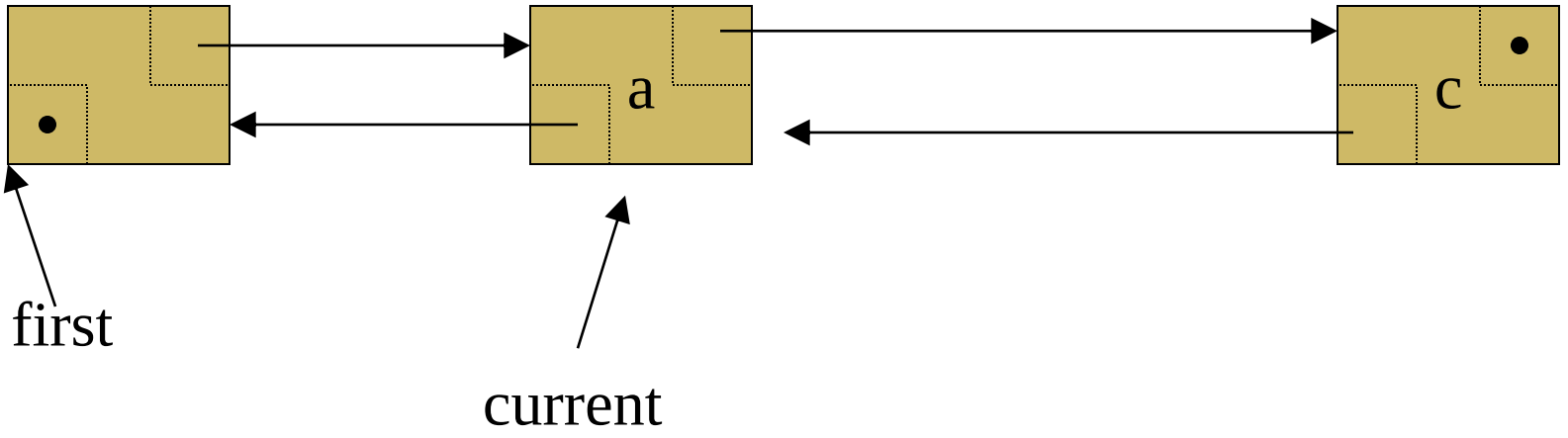

```
oldNode=current;
oldNode->prev->next = oldNode->next;
oldNode->next->prev = oldNode->prev;
current = oldNode->prev;
delete oldNode;
```

Deletion




```
oldNode=current;
oldNode->prev->next = oldNode->next;
oldNode->next->prev = oldNode->prev;
current = oldNode->prev;
delete oldNode;
```

Deletion




```
oldNode=current;
oldNode->prev->next = oldNode->next;
oldNode->next->prev = oldNode->prev;
current = oldNode->prev;
delete oldNode;
```

Reading Materials

- Schaum's Outlines: Chapter # 5
- D. S. Malik: Chapter # 5
- Mark A. Weiss: Chapter # 3
- Chapter 6, ADT, Data Structure and Problem solving with C++ by Larry Nyhoff.
- Chapter 5, Nell Dale 3rd Edition
- Chapter 8, C++ an introduction to Data Structures by Larry Nyhoff