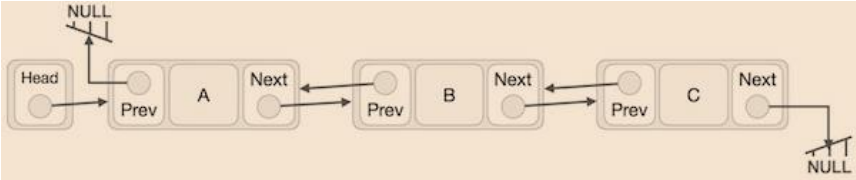
Doubly Linked List

Doubly Linked List is a variation of Linked list in which navigation is possible in both ways, either forward and backward easily as compared to Single Linked List. Following are the important terms to understand the concept of doubly linked list.

- Data− Each node of a linked list can store a data called an element.
- Next− Each link of a linked list contains a link to the next link called Next.
- Prev − Each link of a linked list contains a link to the previous link called Prev.
- LinkedList − A Linked List contains the connection link to the first link called First and to the last link called Last.

Doubly Linked List Representation :



As per the above illustration, following are the important points to be considered.

- Doubly Linked List contains a link element called first and last.
- Each node carries a data field(s) and two link fields called next and prev.
- Each node is linked with its next link using its next link.
- Each node is linked with its previous link using its previous link.
- The last node carries a link as null to mark the end of the list.

Basic Operations :

Following are the basic operations supported by a doubly linked list.

- Insertion − Adds an element at the beginning of the list.
- Deletion − Deletes an element at the beginning of the list.
- Insert Last − Adds an element at the end of the list.
- Delete Last − Deletes an element from the end of the list.
- Insert After − Adds an element after an item of the list.
- Delete − Deletes an element from the list using the key.
- Display forward − Displays the complete list in a forward manner.
- Display backward − Displays the complete list in a backward manner.

Insertion Operation :

The following code demonstrates the insertion operation at the beginning of a doubly linked list.

```
//insert node at the first location

void insertFirst(int key, int data)
{

   //create a node
   struct node *link = (struct node*) malloc(sizeof(struct node));
   link->key = key;
   link->data = data;

   if(isEmpty())
 {
    last = link;  //make it the last link
 }
else
 {
    head->prev = link;  //update first prev link
 }

 link->next = head;  //point it to old first link

 head = link;  //point first to new first link
}
```

Deletion Operation :

Following code demonstrates the deletion operation at the beginning of a doubly linked list.

```
//delete first item
struct node* deleteFirst()
{

   struct node *tempLink = head;   //save reference to first link

      if(head->next == NULL)   //if only one link
       {
         last = NULL;
       }
      else
       {
         head->next->prev = NULL;
       }
      head = head->next;
```

```
    return tempLink; //return the deleted link

  }
```

Insertion at the End of list :

Following code demonstrates the insertion operation at the last position of a doubly linked list.

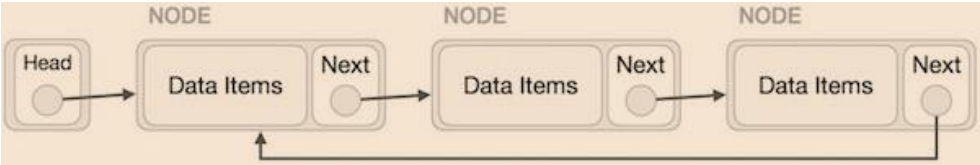
```
//insert node at the last location
void insertLast(int key, int data)
 {

   //create a node
 struct node *link = (struct node*) malloc(sizeof(struct node));
   link->key = key;
   link->data = data;

   if(isEmpty())
         {
          last = link; //make it the last link
         }
   else
     {
      last->next = link; //make link a new last link

      link->prev = last; //mark old last node as prev of new link
     }

   last = link;  //mark old last node as prev of new link
}
```

Circular Linked List

Circular Linked List is a variation of Linked list in which the first element points to the last element and the last element points to the first element. Both Singly Linked List and Doubly Linked List can be made into a circular linked list.

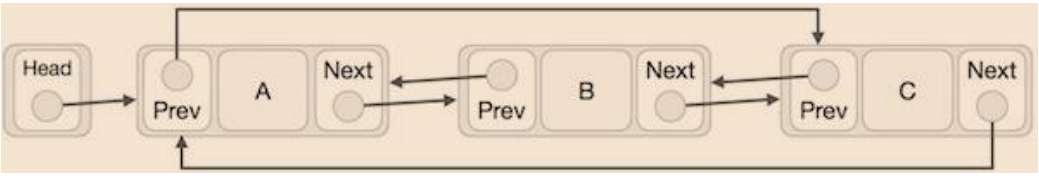
Singly Linked List as Circular :

In singly linked list, the next pointer of the last node points to the first node.



Doubly Linked List as Circular :

In doubly linked list, the next pointer of the last node points to the first node and the previous pointer of the first node points to the last node making the circular in both directions.



As per the above illustration, following are the important points to be considered.

- The last link's next points to the first link of the list in both cases of singly as well as doubly linked list.
- The first link's previous points to the last of the list in case of doubly linked list.

Basic Operations :

Following are the important operations supported by a circular list.

- insert − Inserts an element at the start of the list.
- delete − Deletes an element from the start of the list.
- display − Displays the list.

Insertion Operation :

The following algorithm demonstrates the insertion operation in a circular linked list based on single linked list.

```
insertFirst(data):
Begin
  create a new node
  node -> data := data
  if the list is empty, then
    head := node
    next of node = head
  else
    temp := head
    while next of temp is not head, do
    temp := next of temp
    done
    next of node := head
    next of temp := node
    head := node
  end if
End
```

Deletion Operation :

The following code demonstrates the deletion operation in a circular linked list based on single linked list.

```
deleteFirst():
Begin
  if head is null, then
    it is Underflow and return
  else if next of head = head, then
    head := null
    deallocate head
  else
    ptr := head
    while next of ptr is not head, do
      ptr := next of ptr
    next of ptr = next of head
    deallocate head
    head := next of ptr
  end if
End
```

Display List :

The following code demonstrates the display list operation in a circular linked list.

```
display():
Begin
  if head is null, then
    Nothing to print and return
  else
    ptr := head
    while next of ptr is not head, do
      display data of ptr
      ptr := next of ptr
    display data of ptr
  end if
End
```