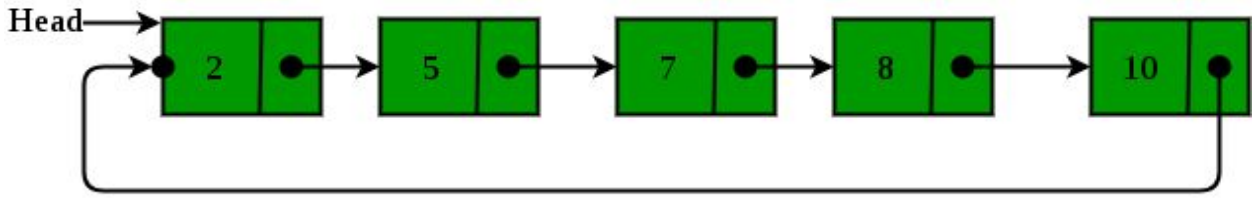
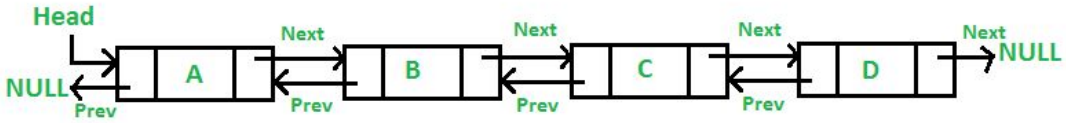
Circular Linked List

Circular linked list is a linked list where all nodes are connected to form a circle. There is no NULL at the end. A circular linked list can be a singly circular linked list or doubly circular linked list.



Doubly Linked List

A **D**oubly **L**inked **L**ist (DLL) contains an extra pointer, typically called *previous pointer*, together with next pointer and data which are there in a singly linked list.



Following is a representation of a DLL node in C language.

```
/* Node of a doubly linked list */
struct Node {
    int data;
    struct Node* next; // Pointer to next node in DLL
    struct Node* prev; // Pointer to previous node in DLL
};
```

Following are advantages/disadvantages of doubly linked list over singly linked list.

Advantages over singly linked list

1) A DLL can be traversed in both forward and backward direction.

2) The delete operation in DLL is more efficient if pointer to the node to be deleted is given.

3) We can quickly insert a new node before a given node.

In a singly linked list, to delete a node, pointer to the previous node is needed. To get this previous node, sometimes the list is traversed. In DLL, we can get the previous node using previous pointer.

Disadvantages over singly linked list

1) Every node of DLL Require extra space for a previous pointer. It is possible to implement DLL with single pointer though

2) All operations require an extra pointer previous to be maintained. For example, in insertion, we need to modify previous pointers together with next pointers. For example in following functions for insertions at different positions, we need 1 or 2 extra steps to set previous pointer.

Insertion

A node can be added in four ways

1) At the front of the DLL

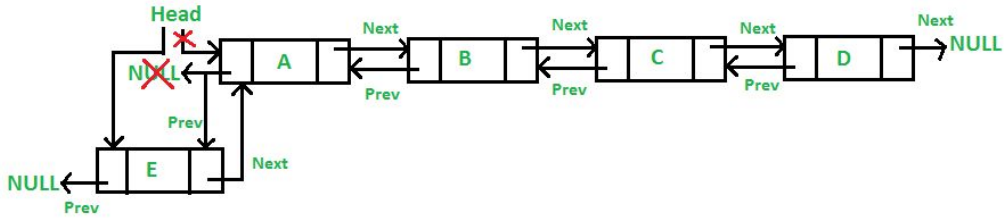
2) After a given node.

3) At the end of the DLL

4) Before a given node.

1) Add a node at the front: (A 5 steps process)

The new node is always added before the head of the given Linked List. And newly added node becomes the new head of DLL. For example, if the given Linked List is 10152025 and we add an item 5 at the front, then the Linked List becomes 510152025. Let us call the function that adds at the front of the list is push(). The push() must receive a pointer to the head pointer, because push must change the head pointer to point to the new node.



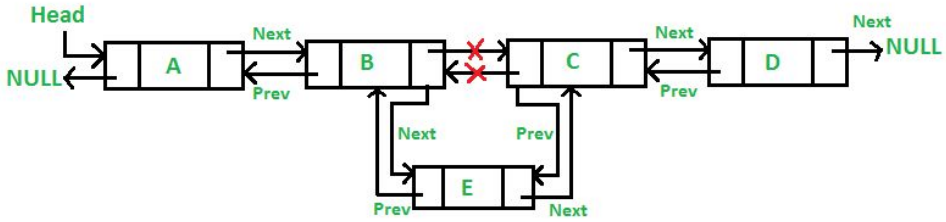
Following are the 5 steps to add node at the front.

```
/* Given a reference (pointer to pointer) to the head of a list
   and an int, inserts a new node on the front of the list. */

void push(struct Node** head_ref, int new_data)
{
    /* 1. allocate node */

    /* 2. put in the data  */

    /* 3. Make next of new node as head and previous as NULL */


    /* 4. change prev of head node to new node */


    /* 5. move the head to point to the new node */
}
```

Four steps of the above five steps are same as the 4 steps used for inserting at the front in singly linked list. The only extra step is to change previous of head.

2) Add a node after a given node.: (A 7 steps process)

We are given pointer to a node as prev_node, and the new node is inserted after the given node.



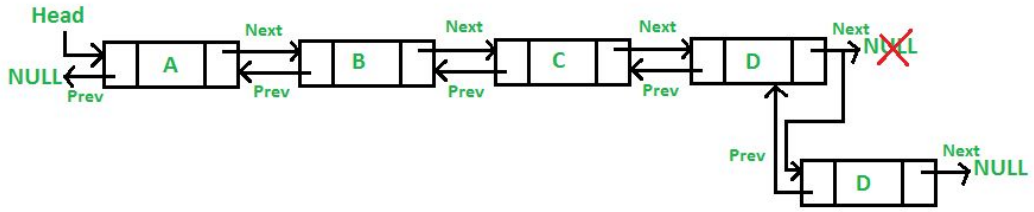
```
/* Given a node as prev_node, insert a new node after the given
node */
void insertAfter(struct Node* prev_node, int new_data)
{
    /*1. check if the given prev_node is NULL */

    /* 2. allocate new node */

    /* 3. put in the data  */


    /* 4. Make next of new node as next of prev_node */

    /* 5. Make the next of prev_node as new_node */


    /* 6. Make prev_node as previous of new_node */


    /* 7. Change previous of new_node's next node */
}
```

Five of the above steps step process are same as the 5 steps used for inserting after a given node in singly linked list. The two extra steps are needed to change previous pointer of new node and previous pointer of new node's next node.

3) Add a node at the end: (7 steps process)

The new node is always added after the last node of the given Linked List. For example if the given DLL is 510152025 and we add an item 30 at the end, then the DLL becomes 51015202530.

Since a Linked List is typically represented by the head of it, we have to traverse the list till end and then change the next of last node to new node.




```
/* Given a reference (pointer to pointer) to the head
   of a DLL and an int, appends a new node at the end  */
void append(struct Node** head_ref, int new_data)
{
    /* 1. allocate node */

    /* 2. put in the data  */

    /* 3. This new node is going to be the last node, so
          make next of it as NULL*/

    /* 4. If the Linked List is empty, then make the new
          node as head */

    /* 5. Else traverse till the last node */

    /* 6. Change the next of last node */


    /* 7. Make last node as previous of new node */


    return;
}
```

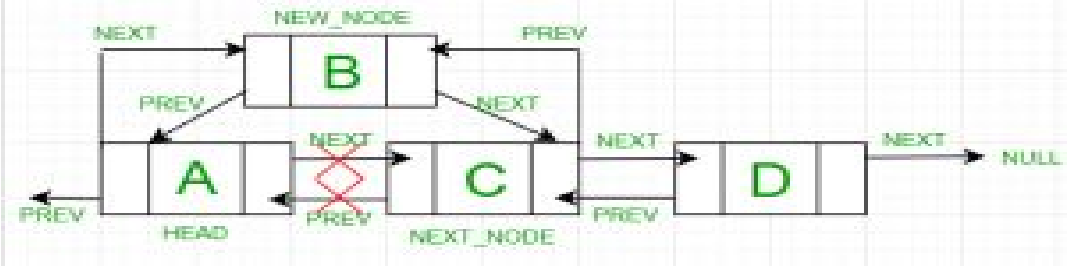
Six of the above 7 steps are the same as the 6 steps used for inserting after a given node in a singly linked list. The one extra step is needed to change previous pointer of new node.

4) Add a node before a given node:

Steps

Let the pointer to this given node be next_node and the data of the new node to be added as new_data.

1. Check if the next_node is NULL or not. If it's NULL, return from the function because any new node can not be added before a NULL
2. Allocate memory for the new node, let it be called new_node
3. Set new_node->data = new_data
4. Set the previous pointer of this new_node as the previous node of the next_node, new_node->prev = next_node->prev
5. Set the previous pointer of the next_node as the new_node, next_node->prev = new_node
6. Set the next pointer of this new_node as the next_node, new_node->next = next_node;
7. If the previous node of the new_node is not NULL, then set the next pointer of this previous node as new_node, new_node->prev->next = new_node
8. Else, if the prev of new_node is NULL, it will be the new head node. So, make (*head_ref) = new_node.



Below is the implementation of the above approach:

```
// A complete working C program to demonstrate all
// insertion before a given node
#include <stdio.h>
#include <stdlib.h>

// A linked list node
struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
};

/* Given a reference (pointer to pointer) to the head of a list
and an int, inserts a new node on the front of the list. */
void push(struct Node** head_ref, int new_data)
{
    struct Node* new_node = (struct Node*)malloc(sizeof(struct
Node));

    new_node->data = new_data;

    new_node->next = (*head_ref);
    new_node->prev = NULL;

    if ((*head_ref) != NULL)
        (*head_ref)->prev = new_node;

    (*head_ref) = new_node;
}

/* Given a node as next_node, insert a new node before the
given node */
void insertBefore(struct Node** head_ref, struct Node*
next_node, int new_data)
{
    /*1. check if the given next_node is NULL */


    /* 2. allocate new node */


    /* 3. put in the data */


    /* 4. Make prev of new node as prev of next_node */
```

```
    /* 5. Make the prev of next_node as new_node */


    /* 6. Make next_node as next of new_node */


    /* 7. Change next of new_node's previous node */

    /* 8. If the prev of new_node is NULL, it will be
       the new head node */
}

// This function prints contents of linked list starting from
the given node
void printList(struct Node* node)
{
   }

/* Driver program to test above functions*/
int main()
{
    /* Start with the empty list */
    struct Node* head = NULL;
    push(&head, 7);

    push(&head, 1);

    push(&head, 4);

    // Insert 8, before 1. So linked list becomes
4->8->1->7->NULL
    insertBefore(&head, head->next, 8);

    printf("Created DLL is: ");
    printList(head);

    getchar();
    return 0;
}
```

Exercises

Question - 01

Given a Linked List of integers, write a function to modify the linked list such that all even numbers appear before all the odd numbers in the modified linked list. Also, keep the order of even and odd numbers same. Do this using **Circular Linked List**
Examples:

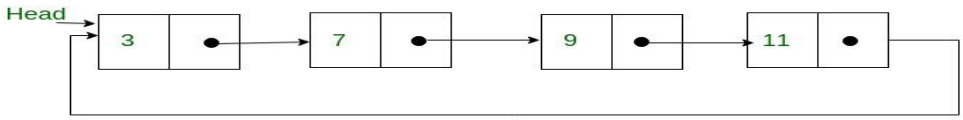
```
Input: 17->15->8->12->10->5->4->1->7->6->NULL
Output: 8->12->10->4->6->17->15->5->1->7->NULL

Input: 8->12->10->5->4->1->6->NULL
Output: 8->12->10->4->6->5->1->NULL

// If all numbers are even then do not change the list
Input: 8->12->10->NULL
Output: 8->12->10->NULL

// If all numbers are odd then do not change the list
Input: 1->3->5->7->NULL
Output: 1->3->5->7->NULL
```

Question - 02

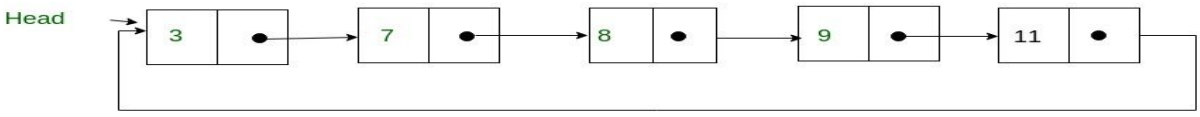
Perform Bubble sort on Doubly Linked List.

Question - 03

Sorted insert for circular linked list. Write a C function to insert a new value in a sorted Circular Linked List (CLL). For example, if the input CLL is following.



After inserting 8, the above CLL should be changed to the following



Question - 04

Given a singly linked list, swap kth node from beginning with kth node from end. **Swapping of data is not allowed, only pointers should be changed.** This requirement may be logical in many situations where the linked list data part is huge (For example student details line Name, RollNo, Address, ..etc). The pointers are always fixed (4 bytes for most compilers).

The problem seems simple at first look, but it has many interesting cases.

Let X be the kth node from beginning and Y be the kth node from end. Following are the interesting cases that must be handled.

1) Y is next to X

2) X is next to Y

3) X and Y are same

4) X and Y don't exist (k is more than the number of nodes in the linked list

Question - 05

Given a sorted doubly linked list of positive distinct elements, the task is to find pairs in doubly linked list whose sum is equal to given value x, without using any extra space ?

Example:

```
Input : head : 1 <-> 2 <-> 4 <-> 5 <-> 6 <-> 8 <-> 9
```

```
        x = 7
```

```
Output: (6, 1), (5,2)
```

Expected time complexity is O(n) and auxiliary space is O(1).