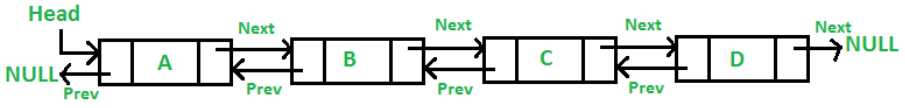
Insert at Begin of Doubly Linked List

A Doubly Linked List (DLL) contains an extra pointer, typically called the previous pointer, together with the next pointer and data which are there in a singly linked list.



Below are operations on the given DLL:

1. **Add a node at the front of DLL:** The new node is always added before the head of the given Linked List. And the newly added node becomes the new head of DLL & maintaining a global variable for counting the total number of nodes at that time.
2. Traversal of a Doubly linked list
3. **Insertion of a node:** This can be done in three ways:
    - **At the beginning:** The new created node is insert in before the head node and head points to the new node.
    - **At the end:** The new created node is insert at the end of the list and tail points to the new node.
    - **At a given position:** Traverse the given DLL to that position(**let the node be X**) then do the following:
        1. Change the next pointer of new Node to the next pointer of Node X.
        2. Change the prev pointer of next Node of Node X to the new Node.
        3. Change the next pointer of node X to new Node.
        4. Change the prev pointer of new Node to the Node X.
4. **Deletion of a node:** This can be done in three ways:
    - **At the beginning:** Move head to the next node to delete the node at the beginning and make previous pointer of current head to NULL .

- **At the last:** Move tail to the previous node to delete the node at the end and make next pointer of tail node to NULL.
- **At a given position:** Let the prev node of Node at position pos be Node X and next node be Node Y, then do the following:
    1. Change the next pointer of Node X to Node Y.
    2. Change the previous pointer of Node Y to Node X.

Below is the implementation of all the operations:

```
// C program to implement all functions
// used in Doubly Linked List

#include <stdio.h>
#include <stdlib.h>
int i = 0;

// Node for Doubly Linked List
typedef struct node {
    int key;
    struct node* prev;
    struct node* next;

} node;

// Head, Tail, first & temp Node
node* head = NULL;
node* first = NULL;
node* temp = NULL;
node* tail = NULL;

// Function to add a node in the
// Doubly Linked List
void addnode(int k)
{

    // Allocating memory
    // to the Node ptr
    node* ptr
```

```
        = (node*)malloc(sizeof(node));

    // Assign Key to value k
    ptr->key = k;

    // Next and prev pointer to NULL
    ptr->next = NULL;
    ptr->prev = NULL;

    // If Linked List is empty
    if (head == NULL) {
        head = ptr;
        first = head;
        tail = head;
    }

    // Else insert at the end of the
    // Linked List
    else {
        temp = ptr;
        first->next = temp;
        temp->prev = first;
        first = temp;
        tail = temp;
    }

    // Increment for number of Nodes
    // in the Doubly Linked List
    i++;
}

// Function to traverse the Doubly
// Linked List
void traverse()
{
    // Nodes points towards head node
    node* ptr = head;
```

```
    // While pointer is not NULL,
    // traverse and print the node
    while (ptr != NULL) {

        // Print key of the node
        printf("%d ", ptr->key);
        ptr = ptr->next;
    }

    printf("\n");
}

// Function to insert a node at the
// beginning of the linked list
void insertatbegin(int k)
{

    // Allocating memory
    // to the Node ptr
    node* ptr
        = (node*)malloc(sizeof(node));

    // Assign Key to value k
    ptr->key = k;

    // Next and prev pointer to NULL
    ptr->next = NULL;
    ptr->prev = NULL;

    // If head is NULL
    if (head == NULL) {
        first = ptr;
        first = head;
        tail = head;
    }

    // Else insert at beginning and
    // change the head to current node
```

```
    else {
        temp = ptr;
        temp->next = head;
        head->prev = temp;
        head = temp;
    }
    i++;
}

// Function to insert Node at end
void insertatend(int k)
{

    // Allocating memory
    // to the Node ptr
    node* ptr
        = (node*)malloc(sizeof(node));

    // Assign Key to value k
    ptr->key = k;

    // Next and prev pointer to NULL
    ptr->next = NULL;
    ptr->prev = NULL;

    // If head is NULL
    if (head == NULL) {
        first = ptr;
        first = head;
        tail = head;
    }

    // Else insert at the end
    else {
        temp = ptr;
        temp->prev = tail;
        tail->next = temp;
        tail = temp;
```

```
    }
    i++;
}

// Function to insert Node at any
// position pos
void insertatpos(int k, int pos)
{

    // For Invalid Position
    if (pos < 1 || pos > i + 1) {
        printf("Please enter a"
               " valid position\n");
    }

    // If position is at the front,
    // then call insertatbegin()
    else if (pos == 1) {
        insertatbegin(k);
    }

    // Position is at length of Linked
    // list + 1, then insert at the end
    else if (pos == i + 1) {
        insertatend(k);
    }

    // Else traverse till position pos
    // and insert the Node
    else {
        node* src = head;

        // Move head pointer to pos
        while (pos--) {
            src = src->next;
        }

        // Allocate memory to new Node
```

```
        node **da, **ba;
        node* ptr
            = (node*)malloc(
                sizeof(node));
        ptr->next = NULL;
        ptr->prev = NULL;
        ptr->key = k;

        // Change the previous and next
        // pointer of the nodes inserted
        // with previous and next node
        ba = &src;
        da = &(src->prev);
        ptr->next = (*ba);
        ptr->prev = (*da);
        (*da)->next = ptr;
        (*ba)->prev = ptr;
        i++;
    }
}

// Function to delete node at the
// beginning of the list
void delatbegin()
{
    // Move head to next and
    // decrease length by 1
    head = head->next;
    i--;
}

// Function to delete at the end
// of the list
void delatend()
{
    // Mode tail to the prev and
    // decrease length by 1
    tail = tail->prev;
```

```
    tail->next = NULL;
    i--;
}

// Function to delete the node at
// a given position pos
void delatpos(int pos)
{

    // If invalid position
    if (pos < 1 || pos > i + 1) {
        printf("Please enter a"
               " valid position\n");
    }

    // If position is 1, then
    // call delatbegin()
    else if (pos == 1) {
        delatbegin();
    }

    // If position is at the end, then
    // call delatend()
    else if (pos == i) {
        delatend();
    }

    // Else traverse till pos, and
    // delete the node at pos
    else {
        // Src node to find which
        // node to be deleted
        node* src = head;
        pos--;

        // Traverse node till pos
        while (pos--) {
            src = src->next;
```

```
        }

        // previous and after node
        // of the src node
        node **pre, **aft;
        pre = &(src->prev);
        aft = &(src->next);

        // Change the next and prev
        // pointer of pre and aft node
        (*pre)->next = (*aft);
        (*aft)->prev = (*pre);

        // Decrease the length of the
        // Linked List
        i--;
    }
}

// Driver Code
int main()
{
    // Adding node to the linked List
    addnode(2);
    addnode(4);
    addnode(9);
    addnode(1);
    addnode(21);
    addnode(22);

    // To print the linked List
    printf("Linked List: ");
    traverse();

    printf("\n");

    // To insert node at the beginning
    insertatbegin(1);
```

```
    printf("Linked List after"
           " inserting 1 "
           "at beginning: ");
    traverse();

    // To insert at the end
    insertatend(0);
    printf("Linked List after "
           "inserting 0 at end: ");
    traverse();

    // To insert Node with
    // value 44 after 3rd Node
    insertatpos(44, 3);
    printf("Linked List after "
           "inserting 44 "
           "after 3rd Node: ");
    traverse();

    printf("\n");

    // To delete node at the beginning
    delatbegin();
    printf("Linked List after "
           "deleting node "
           "at beginning: ");
    traverse();

    // To delete at the end
    delatend();
    printf("Linked List after "
           "deleting node at end: ");
    traverse();

    // To delete Node at position 5
    printf("Linked List after "
           "deleting node "
           "at position 5: ");
```

```
    delatpos(5);
    traverse();

    return 0;
}
```

Output:

Linked List: 2 4 9 1 21 22

Linked List after inserting 1 at beginning: 1 2 4 9 1 21 22

Linked List after inserting 0 at end: 1 2 4 9 1 21 22 0

Linked List after inserting 44 after 3rd Node: 1 2 4 44 9 1 21 22 0

Linked List after deleting node at beginning: 2 4 44 9 1 21 22 0

Linked List after deleting node at end: 2 4 44 9 1 21 22

Linked List after deleting node at position 5: 2 4 44 9 21 22