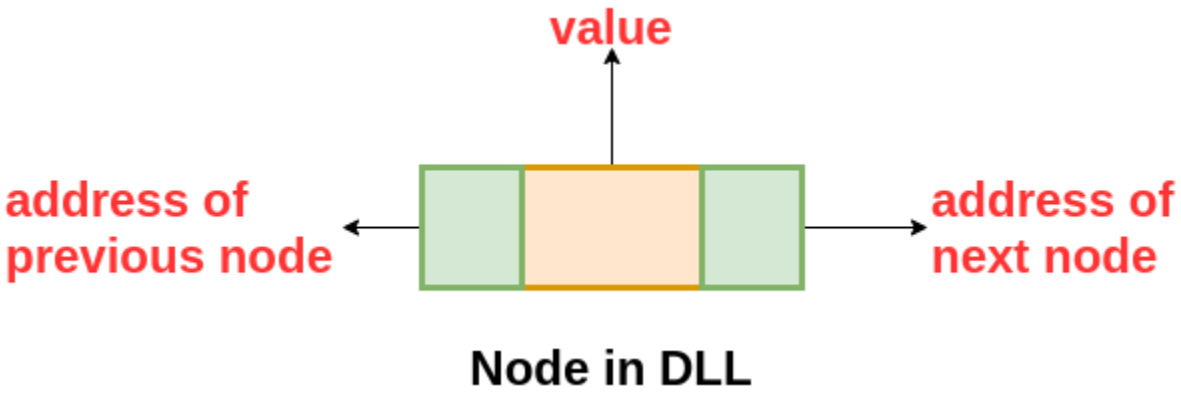
Doubly Linked List

Doubly Linked List: A Doubly-linked list is a collection of nodes linked together in a sequential way where each node is divided into three parts:

i. **Data Field:** Contains the data.

ii. **Previous Pointer:** address of the previous node in the list.

iii. **Next Pointer:** address of the next node in the list.



Basic Structure of Singly Linked List

```
Struct Node{

int data;

struct Node *prev;

struct Node *next;

};
```

Advantages over Singly Linked List-

- It can be traversed both forward and backward direction.
- The delete operation is more efficient if the node to be deleted is given.
- The insert operation is more efficient if the node is given before which insertion should take place.

Disadvantages over Singly Linked List-

- It will require more space as each node has an extra memory to store the address of the previous node.
- The number of modification increase while doing various operations like insertion, deletion, etc.

C program to create and Traverse Doubly Linked List.

```
#include <stdio.h>

#include <stdlib.h>

struct node {

    int data;

    struct node * prev;

    struct node * next;

}*head, *last;
```

```
void createList(int n);

void displayListFromFirst();

void displayListFromEnd();

void main()

{

    int n, choice;

    head = NULL;

    last = NULL;

 printf("Enter the number of nodes you want to create: ");

    scanf("%d", &n);

createList(n);

    printf("\nPress 1 to display list from First");

    printf("\nPress 2 to display list from End : ");

    scanf("%d", &choice);

    if(choice==1)

    {

        displayListFromFirst();

    }

    else if(choice == 2)
```

```
    {
        displayListFromEnd();
    }
getch();
 }
void createList(int n)
{
    int i, data;
    struct node *newNode;

    if(n >= 1)
    {
        head = (struct node *)malloc(sizeof(struct node));

        if(head != NULL)
        {
            printf("Enter data of 1 node: ");
            scanf("%d", &data);
```

```
head->data = data;
head->prev = NULL;
head->next = NULL;

last = head;
for(i=2; i<=n; i++)
{
    newNode = (struct node *)malloc(sizeof(struct node));

    if(newNode != NULL)
    {
        printf("Enter data of %d node: ", i);
        scanf("%d", &data);

        newNode->data = data;
        newNode->prev = last;
        newNode->next = NULL;

        last->next = newNode;
```

```
            last = newNode;
        }
        else
        {
            printf("Unable to allocate memory.");
            break;
        }
    }

    printf("\nDOUBLY LINKED LIST CREATED SUCCESSFULLY\n");
  }
  else
  {
    printf("Unable to allocate memory");
  }
 }
}
```

```
void displayListFromFirst()
{
    struct node * temp;
    int n = 1;

    if(head == NULL)
    {
        printf("List is empty.");
    }
    else
    {
        temp = head;
        printf("\n\nDATA IN THE LIST:\n");

        while(temp != NULL)
        {
            printf("DATA of %d node = %d\n", n, temp->data);

            n++;
```

```
            temp = temp->next;
        }
    }
}
void displayListFromEnd()
{
    struct node * temp;
    int n = 0;

    if(last == NULL)
    {
        printf("List is empty.");
    }
    else
    {
        temp = last;
        printf("\n\nDATA IN THE LIST:\n");
```

```
        while(temp != NULL)
        {
            printf("DATA of last-%d node = %d\n", n, temp->data);


            n++;
                    temp = temp->prev;
        }
    }
}
```

C program to insert new node in a Doubly Linked List

```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int data;
    struct node * prev;
    struct node * next;
}*head, *last;
void createList(int n);
void displayList();
void insertAtBeginning(int data);
void insertAtEnd(int data);
```

```
void insertAtN(int data, int position);
void  main()
{
   int n, data, choice=1;

   head = NULL;
   last = NULL;


   while(choice != 0)
   {
            printf("DOUBLY LINKED LIST PROGRAM\n");
      printf("1. Create List\n");
      printf("2. Insert node - at beginning\n");
      printf("3. Insert node - at end\n");
      printf("4. Insert node - at N\n");
      printf("5. Display list\n");
      printf("0. Exit\n");
      printf("Enter your choice : ");

      scanf("%d", &choice);


      switch(choice)
      {
```

```
case 1:
    printf("Enter the total number of nodes in list: ");
    scanf("%d", &n);

    createList(n);
    break;
case 2:
    printf("Enter data of first node : ");
    scanf("%d", &data);

    insertAtBeginning(data);
    break;
case 3:
    printf("Enter data of last node : ");
    scanf("%d", &data);

    insertAtEnd(data);
    break;
case 4:
    printf("Enter the position where you want to insert new node: ");
    scanf("%d", &n);
    printf("Enter data of %d node : ", n);
    scanf("%d", &data);

    insertAtN(data, n);
```

```
            break;
        case 5:
            displayList();
            break;
        case 0:
            break;
        default:
            printf("Error! Invalid choice. Please choose between 0-5");
      }

      printf("\n\n\n\n\n");
   }

getch();
}
void createList(int n)
{
   int i, data;
   struct node *newNode;

   if(n >= 1)
   {

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

```
printf("Enter data of 1 node: ");
scanf("%d", &data);

head->data = data;
head->prev = NULL;
head->next = NULL;

last = head;

for(i=2; i<=n; i++)
{
   newNode = (struct node *)malloc(sizeof(struct node));

   printf("Enter data of %d node: ", i);
   scanf("%d", &data);

   newNode->data = data;
   newNode->prev = last;
   newNode->next = NULL;

   last->next = newNode;
   last = newNode;
}

printf("\nDOUBLY LINKED LIST CREATED SUCCESSFULLY\n");
```

```
    }
}
void displayList()
{
    struct node * temp;
    int n = 1;

    if(head == NULL)
    {
        printf("List is empty.\n");
    }
    else
    {
        temp = head;
        printf("DATA IN THE LIST:\n");

        while(temp != NULL)
        {
            printf("DATA of %d node = %d\n", n, temp->data);

            n++;
  temp = temp->next;
        }
    }
}
```

```
void insertAtBeginning(int data)
{
   struct node * newNode;

   if(head == NULL)
   {
      printf("Error, List is Empty!\n");
   }
   else
   {
      newNode = (struct node *)malloc(sizeof(struct node));

      newNode->data = data;
      newNode->next = head;
      newNode->prev = NULL;

      head->prev = newNode;

      head = newNode;

      printf("NODE INSERTED SUCCESSFULLY AT THE BEGINNING OF THE LIST\n");
   }
}
```

```
void insertAtEnd(int data)
{
   struct node * newNode;

   if(last == NULL)
   {
      printf("Error, List is empty!\n");
   }
   else
   {
      newNode = (struct node *)malloc(sizeof(struct node));

      newNode->data = data;
      newNode->next = NULL;
      newNode->prev = last;

      last->next = newNode;
      last = newNode;

      printf("NODE INSERTED SUCCESSFULLY AT THE END OF LIST\n");
   }
}
```

```
void insertAtN(int data, int position)
{
    int i;
    struct node * newNode, *temp;

    if(head == NULL)
    {
        printf("Error, List is empty!\n");
    }
    else
    {
        temp = head;
        i=1;

        while(i<position-1 && temp!=NULL)
        {
            temp = temp->next;
            i++;
        }

        if(position == 1)
        {
            insertAtBeginning(data);
        }
        else if(temp == last)
```

```
    {
        insertAtEnd(data);
    }
    else if(temp!=NULL)
    {
        newNode = (struct node *)malloc(sizeof(struct node));

        newNode->data = data;
        newNode->next = temp->next;
        newNode->prev = temp;
        if(temp->next != NULL)
        {

            temp->next->prev = newNode;
        }

        temp->next = newNode;

        printf("NODE INSERTED SUCCESSFULLY AT %d POSITION\n", position);
    }
    else
    {
        printf("Error, Invalid position\n");
    }
}
```

}