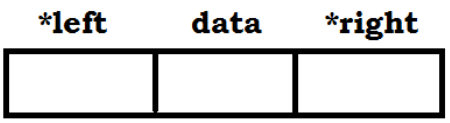
# Double linked List

- It is linear data structure.
- It is used store elements in node form.
- We can represent the node using user defined data type(Structure).
- In double linked list, every node has at most 3 fields.
    - Data field
    - Link to left node
    - Link to right node
- Double linked is bi-directional, hence we can process elements in both directions.
- Compare to Single linked list, it occupies memory (one extra link).

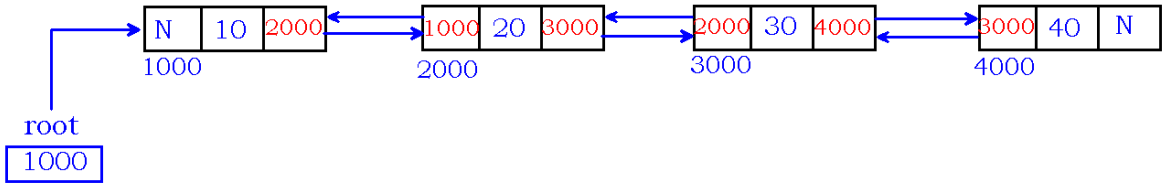
**Node structure:**

```
struct Node
{
    int data ;
    struct Node *left;
    struct Node *right;
};
```




**Linked List representation:**

- Every node has 2 pointers (left , right).
- First node left link is NULL
- Last node right link is NULL.
- Root node always points to First node in the list.



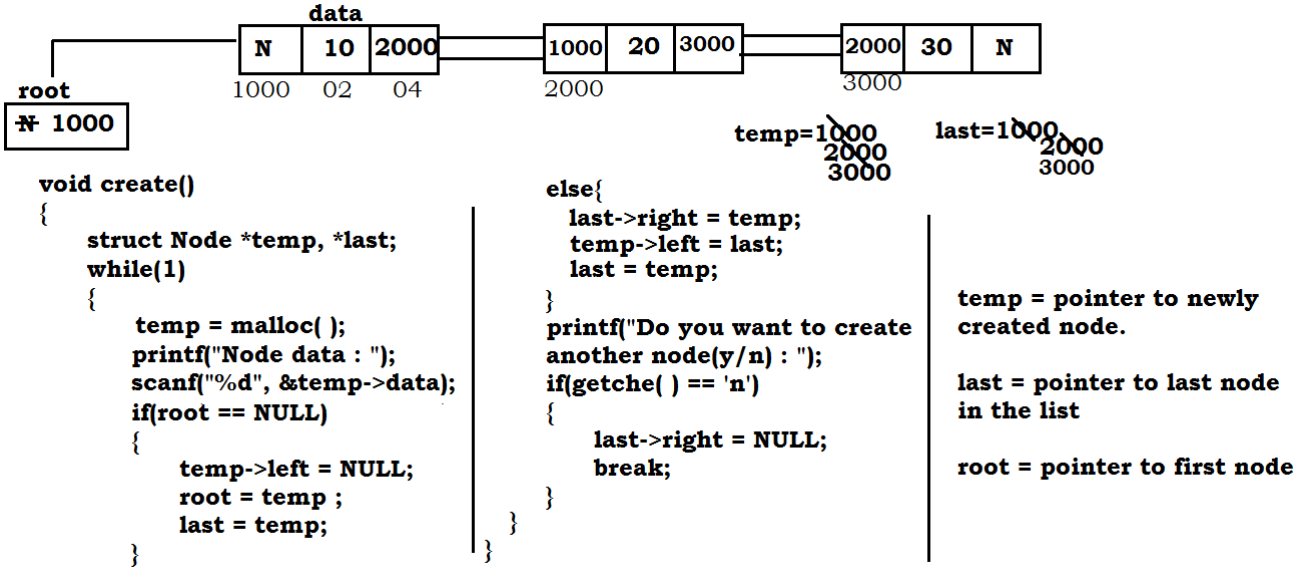

**Operations on double linked list :**

- Create()
- Length()
- Display()
- Append()
- addFirst()
- addAfter()
- remove()
- swap()
- sort()
- reverse()

**Create():**

- The function creates a list of elements using Iterators (loops).

- In the creation of list we use local variables (pointer type).
    - Temp points to newly created node
- Last points to last node in the list.




```
void create()
{
    struct Node *temp, *last;
    while(1)
    {
        temp = malloc( );
        printf("Node data : ");
        scanf("%d", &temp->data);
        if(root == NULL)
        {
            temp->left = NULL;
            root = temp ;
            last = temp;
        }
    else{
      last->right = temp;
      temp->left = last;
      last = temp;
    }
    printf("Do you want to create
    another node(y/n) : ");
    if(getche( ) == 'n')
    {
        last->right = NULL;
        break;
    }
  }
}
```

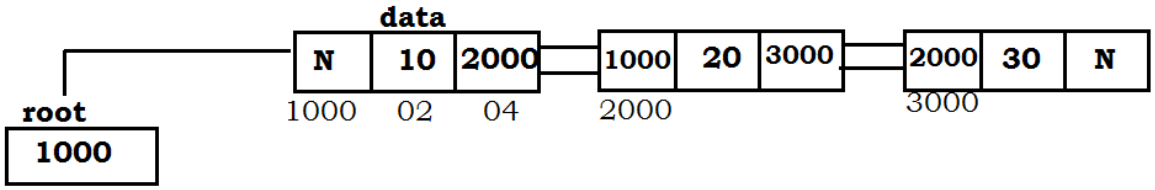
```
temp = pointer to newly
created node.

last = pointer to last node
in the list

root = pointer to first node
```

**Length():**

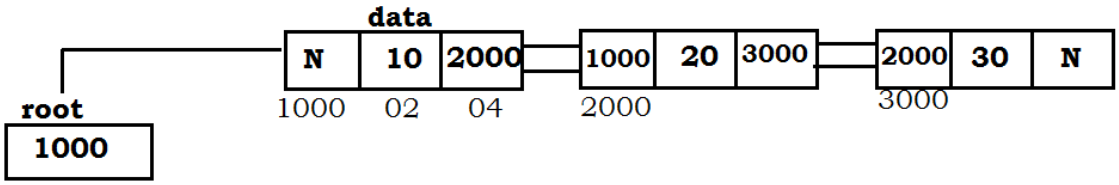
- Returns the number of nodes in the list.
- If list is empty, returns 0
- Return type is integer.




```
int length( )
{
    int count=0;
    struct Node *temp = root ;
    while(temp)
    {
        ++count ;
        temp = temp->right ;
    }
    return count ;
}
```

**Display():**

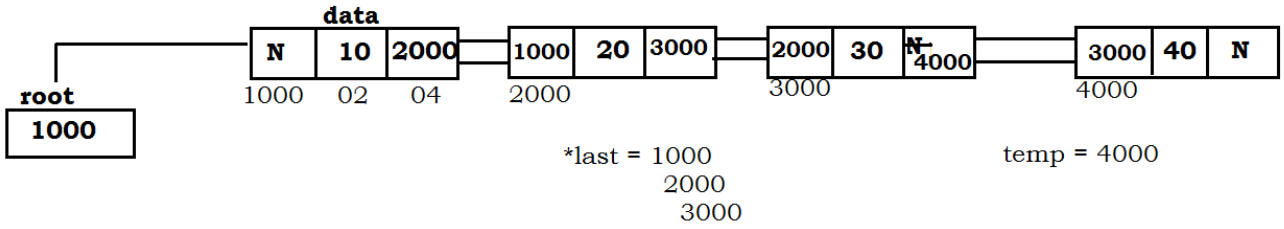
- Used to traverse all elements in the list.
- If the list has no nodes – results message “List is empty”




```
void display()
{
    if(root==NULL){
        printf("List is empty \n");
    }
    else{
        struct Node *temp = root;
        while(temp){
            printf("%d \n", temp->data);
            temp = temp->right;
        }
    }
}
```

**Append():**

- It is used append(add a node at end) a node to list.
- If the list has no elements, new node becomes root node.
- If list has elements, we use iterator to reach the last node in the list.
- We connect the new node to last node.




```
void append()
{
    struct Node *temp, *last;
    temp = malloc( );

    printf("Node data : ");
    scanf("%d" , &temp->data);
    temp->right = NULL;

    if(root == NULL)
    {
        temp->left = NULL;
        root = temp ;
    }
    else
    {
        last = root ;

        while(last->right)
        {
            last = last->right;
        }
        last->right = temp ;
        temp->left = last ;
    }
}
```