

Doubly Linked List



Structure and Diagrammatic representation of doubly linked list

```
struct link
```

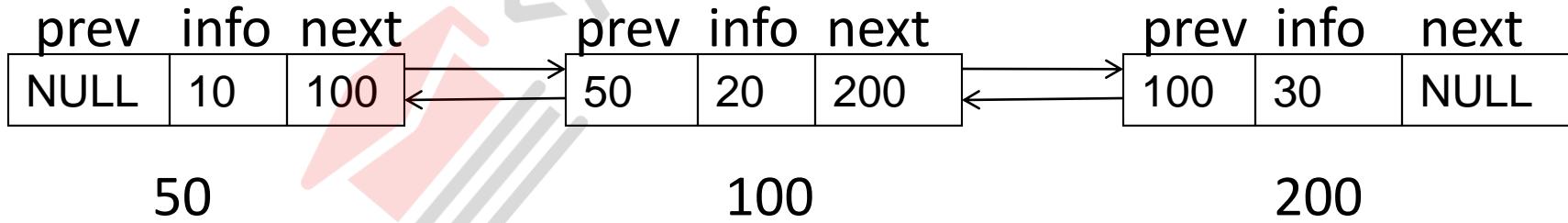
```
{
```

```
    struct link *prev;
```

```
    int info;
```

```
    struct link *next;
```

```
};
```



Doubly linked list

- In linear linked list and circular linked list, one cannot traverse the list backwards. The doubly linked list can be used in such operations.
- Algorithms

Creating the list

- Creating a linked list

Algorithm create_linked_list(struct link *header)

1. [initialize]

1. header->prev= NULL
2. header->next= NULL
3. Point the node to the header

2. Repeat while(ch!=‘n’)

1. node->next= create a dynamic list of type struct link
2. node->next->prev=node
3. node=node->next
4. Input the data for the node
5. node->next = null
6. Ask the user for continuing to create the list

Displaying a linked list

Algorithm display_linked_list(struct link *header)

1. node=point to the first node
2. Repeat while(node)
 1. Display node->info
 2. node=node-> next

Insert data at the beginning of the list

Algorithm

```
insert_at_beg(struct  
link *header)
```

1. [declare]

struct link *new1

2. [initialize]

1. node= point to the
first node

2. prevnode=address
of the header

3. new1 =create a
dynamic list of type
struct link

4.

point new1->next to
node

point new1->prev to
prevnode

point prevnode->next
to new1

node->prev=new1

Enter the value for the
new node new1

Insert data at the end of the list

- Algorithm insert_at_end(struct link *header)
1. [Initialize]
 struct link *new1
 node=point to the next node
 prevnode=address of the header
 2. if(node=NULL)
 display "LIST IS EMPTY"
 3. new1= create a dynamic list of type struct link
 4. Input the data for the new1 node
 5. Repeat while(node)
 1. node=node->next
 2. prevnode=prevnode->next
 6. Point new1->next to node
 7. Point new1->prev to prevnode
 8. Point prevnode->next to new1

Insert data in the middle of the list

Algorithm insert_in_the_middle(struct link *header)

1. [initialize]

int node_num=1;

int insert_node;

node=point to the next node

prevnode=address of the header

2. Input the location at which the node is to be inserted.

Insert data in the middle of the list

3. if(insert_node<=node_counts)
 1. Repeat while(node)
if(node_num= insert_node)
 1. new1 =create a
dynamic list of
type struct link
 2. point prevnode->next
to new1
 3. point new1->next to
node
 4. point new1->prev to
prevnode
 5. point node->prev to new1
 6. Enter the value for
the new node new1
 - else
 1. point node to the
next node
 2. point previous to the
next node
 - node_num++;
 - else
 - display “ THERE ARE ONLY
'node_counts' NODES IN
THE LIST”

Delete first node in the list

Algorithm delete_first_node(struct link *header)

1. [initialize]
 1. node=point to the next node
 2. prevnode=address of the header
2. If node=NULL
 1. display “ THERE ARE NO NODES”
 2. return
- else
 1. prevnode->next= node->next
 2. node->next->prev=prevnode
 3. free the node
3. node=header->next

Delete last node in the list

Algorithm

```
delete_last_node(struct link  
*header)
```

1. [initialize]

```
node_number=node_counts  
node=pointer to the first
```

node

```
prevnode=address of the  
header
```

2. If node=NULL

1. display “ THERE ARE
NO NODES”

2. return

else

1. Repeat while (node
and node_number != 1)
 1. node= point node to
the next node
 2. prevnode=point
previous to
the next node
 2. prevnode->prev->next=
node
 3. free the prevnode

Algorithm to delete desired node from the list

Algorithm

```
delete_desired_node(struct  
link *header)
```

1.[initialize]

 node_number=1

 delete_node=0

 node=point to the first
 node

 previous=address of the
 header

2.if(node=NULL)

 1. display "LIST IS

EMPTY"

3. Enter the node number you
want to delete

4. Repeat while(node)

 1. if(node_number=
 delete_node)

 1. previous->next=
 node->next

 2. node->next->prev
 =prevnode

 3. free(node)

 else

 1. node=point node
 to the next node

 2. previous=point previous
 to the next node

 node_number++