Syntax-Linked List

All types

Creation of Singly Linked list

```
struct node
{
int data;
struct node *next;
};

int main()
{
struct node *head,*newnode,*temp;
head=0;
int choice=1;
while(choice)
{
newnode=(structnode*)malloc(sizeof(struct node));
printf(“Enter data”);
Scanf(“%d”,&newnode->data);
```

```
newnode->next=0;
if(head==0)
{
head=temp=newnode;
}
else
{
temp->next=newnode;
temp=newnode;
}
printf("Do you want to continue(0,1)?");
scanf("%d",&choice);
}
//Traverse
temp=head;
while(temp!=0)
{
printf("%d",temp->data);
temp=temp->next;
}
getch();
}
```

Traversal(Logic)

```
Traverse(struct node *head)
{
if(head==NULL)
printf(“Linked List empty”);
struct node*ptr;
ptr=head;`
while(ptr!=NULL)
{
   printf(“%d”,ptr->data);
   ptr=ptr->next;
}
```

Traversal(Algorithm)

1.Set PTR=START

2.Repeat step 3 and 4 While PTR!=NULL

3.Apply Process to INFO[PTR]

4.Set PTR=LINK[PTR];

5.Exit

INSERTION(Beginning)

```
Insertbeg(struct node*head,int info)
{
 struct node *new;
new=(struct node*)malloc(sizeof(struct node));
new->data=info;
new->next=head;
head=new;
return head;
}
```

INSERTION(End)

```
Insertend(struct node*head,int info){
 struct node *ptr,*new;
new=(struct node*)malloc(sizeof(struct node));
new->data=info;
new->next=NULL;
ptr=head;
if(ptr!=NULL)
{
        While(ptr->next!=NULL)
        {
                Ptr=ptr->next;
        }
ptr->next=new;
}
else
head=new;
return head;
}
```

INSERTION(After a given node)

```
Insertend(struct node*head,int info,int x){
 struct node *ptr,*new;
new=(struct node*)malloc(sizeof(struct node));
new->data=info;
ptr=head;
While(ptr->data!=x && ptr!=NULL)
{
        ptr=ptr->next;
}
if(ptr->data==x)
{
new->next=ptr->next;
ptr->next=new;
}
Return head;
}
```

Deletion at beginning

```
delfirst(struct node *ptr)
{
   srruct node *ptr;
   if(head==NULL)
      printf(“List is already empty”);
else
 {
ptr=head;
head=head->next;
free(ptr);
}
return head;
```

Time complexity –O(1)

Deletion at end

```
del_last(struct node *head)
{
    struct node *ptr,*prep;
   if(head==NULL)
 printf("List is empty");
else if(head->next==NULL)
{
   free(head);
  head=null;
}
else
{
 ptr=head;
while(ptr->next!=NULL)
{
prep=ptr;
ptr=ptr->next;
}
prep->next=NULL;
free(ptr);
}
return head;
}
```

Time complexity-O(n)

Deletion after a given element

```
Delete_after(struct node *head,int key)
{
 struct node *ptr1,*ptr2;
ptr1=head;
 while(ptr1->next!=NULL)
{
   if(ptr1->data==key)
   {
        ptr2=ptr1->next;
ptr1->next=ptr2->next;
free(ptr2);
break;
}
ptr1=ptr1->next;
}
return head;
}
```

Time complexity-O(n)

Doubly Linked List

Creation of Doubly Linked list

```
#include<stdio.h>
#include<stdlib.h>

struct node
{
struct node *prev;
int data;
struct node *next;
};

int main()
{
struct node *head,*newnode,*temp;
head=0;
int choice=1;
While(choice)
{
newnode=(structnode*)malloc(sizeof(struct node));
printf(“Enter data”);
Scanf(“%d”,&newnode->data);
```

```
newnode->prev=0;
newnode->next=0;
if(head==0)
{
head=temp=newnode;
}
else
{
temp->next=newnode;
New->prev=temp;
Temp=temp->next;
}
```

```
Printf("Do you want to continue(0,1)?");
Scanf("%d",&choice);
}
//Traverse
temp=head;
while(temp!=0)
{
printf("%d",temp->data);
temp=temp->next;
}
getch();
}
```

Insertion in beginning

```
//new->data=info;
new->next=head;
head->prev=new;
head=new;
```

Insertion at end

```
struct node *temp;
temp=head;
while(temp->next!=NULL)
{
        temp=temp->next;
}
        temp->next=new;
        new->prev=temp;
```

Insertion at specific position

```
struct node *temp;
temp=head;
while(temp->data!=x)
{
temp=temp->next;
}
new->next=temp->next;
new->prev=temp;
temp->next=new;
new->next->prev=new;
```

Pre condition

```
If(head==NULL)
{
        printf(“linked list empty”);
}
If(head->next==NULL)
{
 free(head);
Head=NULL;
}
else
```

Deletion at beginning

```
struct node *p=head;
head=head->next;
head->prev=NULL;
free(p);
P=NULL;
```

Deletion at end

```
struct node *p=head;
while(p->next!=NULL)
{
  p=p->next;
}
p->prev->next=NULL;
free(p);
P=NULL;
```

Deletion of a given node

```
Struct node *p=head;
While(p->data!=x)
{
        p=p->next;
}
p->prev->next=p->next;
P->next->prev=p->prev;
free(p);
p=NULL;
```

Circular Singly linked list

Creation of Singly circular Linked list

```
struct node
{
int data;
struct node *next;
};

int main()
{
struct node *head,*newnode,*temp;
head=0;
int choice=1;
While(choice)
{
newnode=(structnode*)malloc(sizeof(struct node));
printf(“Enter data”);
scanf(“%d”,&newnode->data);
```

```
newnode->next=0;
if(head==0)
{
head=temp=newnode;
}
else
{
temp->next=newnode;
temp=newnode;
}
temp->next=head;
printf("Do you want to continue(0,1)?");
scanf("%d",&choice);
}
//Traverse
temp=head;
while(temp->next!=head)
{
printf("%d",temp->data);
temp=temp->next;
}
printf("%d",temp->data);
getch();
}
```

Insertion-In beginning

```
if(head==0)
{
        head=new;
        new->next=head;
}
```

Insertion-In beginning

```
Struct node *p=head;
While(p->next!=head)
{
        p=p->next;
}
p->next=new
new->next=head;
head=new;
```

Insertion-In End

```
struct node *p=head;
while(p->next!=head)
{
        p=p->next;
}
p->next=new;
new->next=head;
```

Insertion-After a given node

```
struct node *p=head;
while(p->data!=x)
{
        p=p->next;
}
new->next=p->next;
p->next=new;
```

Pre condition-Deletion

```
If(head==NULL)
{
        printf(“linked list empty”);
}
If(head->next==NULL)
{
 free(head);
Head=NULL;
}
else
```

Deletion-In Beginning

```
struct node *t;
t=head;
while(t->next!=head)
{
        t=t->next;
}
p=head;
head=head->next;
t->next=head;
Free(p);
p=NULL;
```

Deletion at end

```
struct node *t,*p;
p=NULL,t=head;
while(t->next!=head)
{
        p=t;
        t=t->next;
}
p->next=head;
free(t);
t=NULL;
```

Deletion of a given node

```
struct node *t,*p;
p=NULL,t=head;
while(t->data!=x)
{
        p=t;
        t=t->next;
}
p->next=t->next;
free(t);
t=NULL;
```

Circular Doubly Linked list

Creation of Circular Doubly Linked list

```
#include<stdio.h>
#include<stdlib.h>

struct node
{
struct node *prev;
int data;
struct node *next;
};

int main()
{
struct node *head,*newnode,*temp;
head=0;
int choice=1;
While(choice)
{
newnode=(structnode*)malloc(sizeof(struct node));
printf(“Enter data”);
Scanf(“%d”,&newnode->data);
```

```
newnode->prev=0;
newnode->next=0;
if(head==0)
{
head=temp=newnode;
head->next=new;
head->prev=new;
}
else
{
temp->next=newnode;
newnode->prev=temp;
newnode->next=head;
head->prev=new;
temp=temp->next;
}
```

```
Printf("Do you want to continue(0,1)?");
Scanf("%d",&choice);
}
//Traverse
temp=head;
while(temp->next!=head)
{
printf("%d",temp->data);
temp=temp->next;
}
printf("%d",temp->data);

getch();
}
```

Insertion(pre-condition)

```
if(head==NULL)
{
head=new;
head->next=head;
head->prev=head;
}
```

Insertion-Beginning

```
struct node *p;
p=head->prev;
head->prev=new;
new->next=head;
p->next=new;
new->prev=p;
head=new;
```

Insertion at end

```
struct node *p;
p=head->prev;
p->next=new;
new->prev=p;
new->next=head;
head->prev=new;
```

Insertion after a given node

```
struct node *p=head;
while(p->data!=x)
{
        p=p->next;
}
new->next=p->next;
new->prev=p;
p->next=new;
new->next->prev=new;
```

Deletion (pre condition)

```
if(head==NULL)
{
    printf(“circular doubly linked list is empty:”);
}
if(head->next==head)
{
        free(head);
        head=NULL;
}
else
{}
```

Deletion at beginning

```
Struct node *p,*t;
p=head->prev;
t=head;
head=head->next;
head->prev=p;
p->next=head;
free(t);
t-=NULL;
```

Deletion at end

```
struct node *p;
p=head->prev;
p->prev->next=head;
head->prev=p->prev;
free(p);
p=NULL;
```

Delete a given node

```
struct node *p=head;
while(p->data!=x)
{
        p=p->next;
}
p->prev->next=p->next;
p->next->prev=p->prev;
free(p);
p=NULL;
```