WEEK-5

Date:23-07-2025

List of programs:

1. Write a C program to insert a node at the beginning in a single linked list.
2. Write a C program to insert a node at the end in a single linked list.
3. Write a C program to insert a node after a given node(middle case) in a single linked list.
4. Write a C program to delete a node at the beginning in a single linked list.
5. Write a C program to delete a node at the end in a single linked list
6. Write a C program to delete a node after a given node(middle case) in a single linked list.

1. **Aim:** To write a C program to insert a node at the beginning in a single linked list.

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *next;
};
struct node *head=NULL,*temp,*new=NULL;
struct node *getnode(int x)
{
struct node *temp=(struct node*)malloc(sizeof(struct node));
temp->data=x;
temp->next=NULL;
return temp;
```

```
}
void insert_begin()
{
if(head==NULL)
   head=new;
else
{
new->next=head;
head=new;
}
}
void display()
{
if(head==NULL)
{
printf("list empty\n");
}
else
{
temp=head;
while(temp!=NULL)
{
printf("%d->",temp->data);
temp=temp->next;
}
}
printf("null\n");
}
```

```
int main()
{
int n,i,x;
printf("enter the number of node:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter the data in %d node:",i+1);
scanf("%d",&x);
new=getnode(x);
if(head==NULL)
{
head=new;
temp=head;
}
else
{
temp->next=new;
temp=new;
}
}
printf("enter the data to insert at beginning");
scanf("%d",&x);
new=getnode(x);
insert_begin();
printf("the list after inserting at beginning\n");
display();
return 0;
```

```
}
```

Output:

```
enter the number of node:5
enter the data in 1 node:100
enter the data in 2 node:200
enter the data in 3 node:300
enter the data in 4 node:400
enter the data in 5 node:500
enter the data to insert at beginning10
the list after inserting at beginning
10->100->200->300->400->500->null


=== Code Execution Successful ===
```

2. Aim: Write a C program to insert a node at the end in a single linked list.

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *next;
};
struct node *head=NULL,*temp,*new=NULL;
struct node *getnode(int x)
{
struct node *temp=(struct node*)malloc(sizeof(struct node));
temp->data=x;
temp->next=NULL;
return temp;
}
void insert_end(int x)
{
new=getnode(x);
if(head==NULL)
{
head=new;
temp=head;
}
else
{
```

```
temp->next=new;
temp=new;
}
}
void display()
{
if(head==NULL)
{
printf("list is empty\n");
}
else
{
temp=head;
while(temp!=NULL)
{
printf("%d->",temp->data);
temp=temp->next;
}
printf("null\n");
}
}
int main()
{
int n,i,x;
printf("enter the number of node:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
```

```
printf("enter the data in %d node:",i+1);
scanf("%d",&x);
insert_end;
}
printf("the list after insertion:\n");
display();
return 0;
}
```

Output:

```
Enter the number of nodes: 4
Enter the data for node 1: 11
Enter the data for node 2: 22
Enter the data for node 3: 33
Enter the data for node 4: 44
The list after insertion:
11->22->33->44->NULL


=== Code Execution Successful ===
```

3. **Aim:** Write a C program to insert a node after a given node(middle case) in a single linked list.

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *next;
};
struct node *head=NULL,*temp,*new;
struct node *getnode(int x)
{
struct node *temp=(struct node*)malloc(sizeof(struct node));
temp->data=x;
temp->next=NULL;
return temp;
}
void insert_middle(int val,int y)
{
struct node *temp=head;
while(temp->data!=val)
{
temp=temp->next;
}
struct node * new=getnode(y);
```

```
new->next=temp->next;
temp->next=new;
}
void display()
{
if(head==NULL)
{
printf("list empty\n");
}
else
{
temp=head;
while(temp!=NULL)
{
printf("%d->",temp->data);
temp=temp->next;
}
printf("null\n");
}
}
int main()
{
int n,i,x,val,y;
printf("enter the number of node:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter the data in %d node:",i+1);
```

```
scanf("%d",&x);
new=getnode(x);
if(head==NULL)
{
head=new;
temp=head;
}
else
{
temp->next=new;
temp=temp->next;
}}
printf("the list before insertion:\n");
display();
printf("enter the value in which node after you want to insert:");
scanf("%d",&val);
printf("enter the value you want to insert");
scanf("%d",&y);
insert_middle(val,y);
printf("the list after insertion:\n");
display();
return 0;
}
```

Output:

```
enter the number of node:5
enter the data in 1 node:9
enter the data in 2 node:18
enter the data in 3 node:27
enter the data in 4 node:36
enter the data in 5 node:45
the list before insertion:
9->18->27->36->45->null
enter the value in which node after you want to insert:18
enter the value you want to insert20
the list after insertion:
9->18->20->27->36->45->null


=== Code Execution Successful ===
```

4. Aim: To write a C program to delete a node at the beginning in a single linked list.

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *next;
};
struct node *head=NULL,*temp,*new,*last;
struct node *getnode(int x)
{
struct node *temp=(struct node*)malloc(sizeof(struct node));
temp->data=x;
temp->next=NULL;
return temp;
}
void delete_begin()
{
if(head==NULL)
{
printf("deletion is not possible");
}
else
{
```

```
temp=head;
head=head->next;
free(temp);
printf("node deleted from beginning");
}
}
void display()
{
if(head==NULL)
{
printf("list empty\n");
}
else
{
temp=head;
while(temp!=NULL)
{
printf("%d->",temp->data);
temp=temp->next;
}
printf("null\n");
}
}
int main()
{
int n,i,x;
printf("enter the number of node:");
scanf("%d",&n);
```

```
for(i=0;i<n;i++)
{
printf("enter the data in %d node:",i+1);
scanf("%d",&x);
new=getnode(x);
if(head==NULL)
{
head=last=new;
}
else
{
last->next=new;
last=new;
}
}
printf("the list after creation:\n");
display();
delete_begin();
printf(" list after deleting first node:\n");
display();
return 0;
}
```

Output:

```
enter the number of node:5
enter the data in 1 node:3
enter the data in 2 node:6
enter the data in 3 node:9
enter the data in 4 node:12
enter the data in 5 node:15
the list after creation:
3->6->9->12->15->null
node deleted from beginning list after deleting first node:
6->9->12->15->null


=== Code Execution Successful ===
```

5. Aim: Write a C program to delete a node at the end in a single linked list.

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *next;
};
struct node *head=NULL,*temp=NULL,*new;
struct node *getnode(int x)
{
struct node *temp=(struct node*)malloc(sizeof(struct node));
temp->data=x;
temp->next=NULL;
return temp;
}
void delete_end()
{
if(head==NULL)
{
printf("deletion is not possible");
}
else if(head->next==NULL)
{
temp=head;
head=NULL;
```

```
free(temp);
}
else
{
struct node*temp1=NULL;
temp=head;
while(temp->next!=NULL)
{
temp1=temp;
temp=temp->next;
}
temp1->next=NULL;
free(temp);
}
}
void display()
{
if(head==NULL)
{
printf("list empty\n");
}
else
{
temp=head;
while(temp!=NULL)
{
printf("%d->",temp->data);
temp=temp->next;
```

```
}
printf("null\n");
}
}
int main()
{
int n,i,x;
printf("enter the number of node:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter the data in %d node:",i+1);
scanf("%d",&x);
new=getnode(x);
if(head==NULL)
{
head=new;
}
else
{
temp=head;
while(temp->next!=NULL)
{
temp=temp->next;
}
temp->next=new;
}
}
```

```
printf(" list after deleting end node:\n");

delete_end();

display();

return 0;

}
```

Output:

```
enter the number of node:5
enter the data in 1 node:2
enter the data in 2 node:4
enter the data in 3 node:6
enter the data in 4 node:8
enter the data in 5 node:10
 list after deleting end node:
2->4->6->8->null


=== Code Execution Successful ===
```

6. **Aim:** Write a C program to delete a node after a given node(middle case) in a single linked list.

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *next;
};
struct node *head=NULL,*temp,*new;
struct node *getnode(int x)
{
struct node *temp=(struct node*)malloc(sizeof(struct node));
temp->data=x;
temp->next=NULL;
return temp;
}
void delete_middle(int val)
{
if(head==NULL)
{
printf("deletion is not possible");
}
else if(head->next==NULL)
{
printf("deletion from middle is not possible");
```

```
}
else
{
struct node*temp1=NULL;
temp=head;
while(temp->data!=val)
{
temp1=temp;
temp=temp->next;
}
temp1->next=temp->next;
free(temp);
}
}
void display()
{
if(head==NULL)
{
printf("list empty\n");
}
else
{
temp=head;
while(temp!=NULL)
{
printf("%d->",temp->data);
temp=temp->next;
}
```

```
printf("null\n");
}
}
int main()
{
int n,i,x,val;
printf("enter the number of node:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter the data in %d node:",i+1);
scanf("%d",&x);
new=getnode(x);
if(head==NULL)
{
head=new;
}
else
{
temp=head;
while(temp->next!=NULL)
{
temp=temp->next;
}
temp->next=new;
}
}
printf("enter the value that you want to delete:\n");
```

```
scanf("%d",&val);
printf(" list after deleting middle node:\n");
delete_middle(val);
display();
return 0;
}
```

Output:

```
enter the number of node:5
enter the data in 1 node:4
enter the data in 2 node:8
enter the data in 3 node:12
enter the data in 4 node:16
enter the data in 5 node:20
enter the value that you want to delete:
12
 list after deleting middle node:
4->8->16->20->null


=== Code Execution Successful ===
```

Inferences:

- List size is **dynamic** that is nodes created at runtime using `malloc`.
- Nodes are **not stored in contiguous memory** unlike arrays.
- Traversal is **one-way only** that is forward, from `head` to `NULL`.
- Last node's `next` pointer is always **NULL** → end of list.
- **Head pointer is important** because losing it means losing the list.
- Insertion and deletion are **easier and faster** than arrays.
- Searching/traversing is **slower** → O(n) time complexity.
- Each node requires **extra memory** for the pointer (`struct node* next`).
- Cannot access elements randomly. It has only sequential access.
- Commonly used in **stacks, queues, graphs, and dynamic memory structures**.