Program:
Design and implement a dynamic list (Singly linked list/ doubly linked list) to store any
information which needs a linear data structure.

Aim:
To implement a Singly linked list

Algorithm:

Beginsert

- Firstly make a pointer and allocate some memory to it.
- If the pointer is NULL then display an memory error
- Then check if the list is full and display appropriate error
- Else set the data element of pointer to value which the user intends to insert
- Then set it’s next element to head
- Lastly set the head element to point to new pointer

Lstinsert

- Firstly make a pointer and allocate some memory to it.
- If the pointer is NULL then display an memory error
- Then check if the list is full and display appropriate error
- Else set the data element of pointer to value which the user intends to insert
- Now check if the list is empty
- If it is empty then set ptr’s next element to NULL and set head to ptr
- If it’s not empty then make a new pointer and traverse to the last element in the list
- Now set it’s next element to ptr and set ptr’s next element

RandInsert

- Firstly make a pointer and allocate some memory to it.
- If the pointer is NULL then display an memory error
- Then check if the list is full and display appropriate error

- Else set the data element of pointer to value which the user intends to insert
- Now take input from the user specifying the location where the node is to be inserted
- Now traverse to the location
- Set ptr’s next element to temp’s next element
- And now set temp’s next element to ptr

Begdelete

- Firstly check if the list is empty
- If its empty then print the underflow condition
- If it’s not empty then set head to it’s next element and free the block of memory that head was occupying previously

Lstdelete

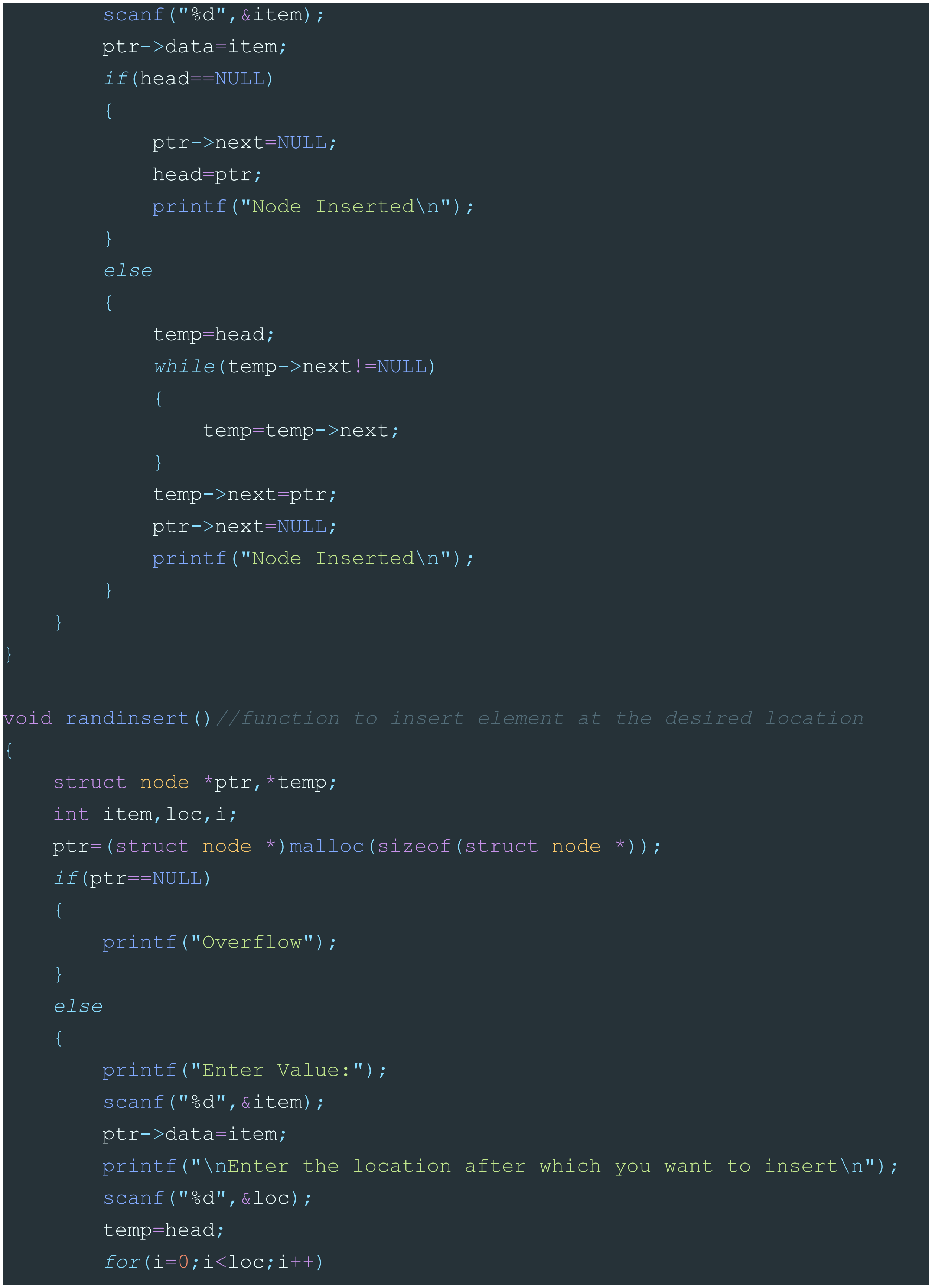
- Firstly check if the list is empty
- If its empty then print the underflow condition]
- Now check if head is the only element by checking if its next element is NULL
- If it’s NULL then free the memory and set head to NULL
- Else, traverse to the second last element of the list by keeping a pointer which points to the previous element of the current traversing pointer
- Now set the next element of this pointer to NULL and free the memory block from the traversing pointer

Randdelete

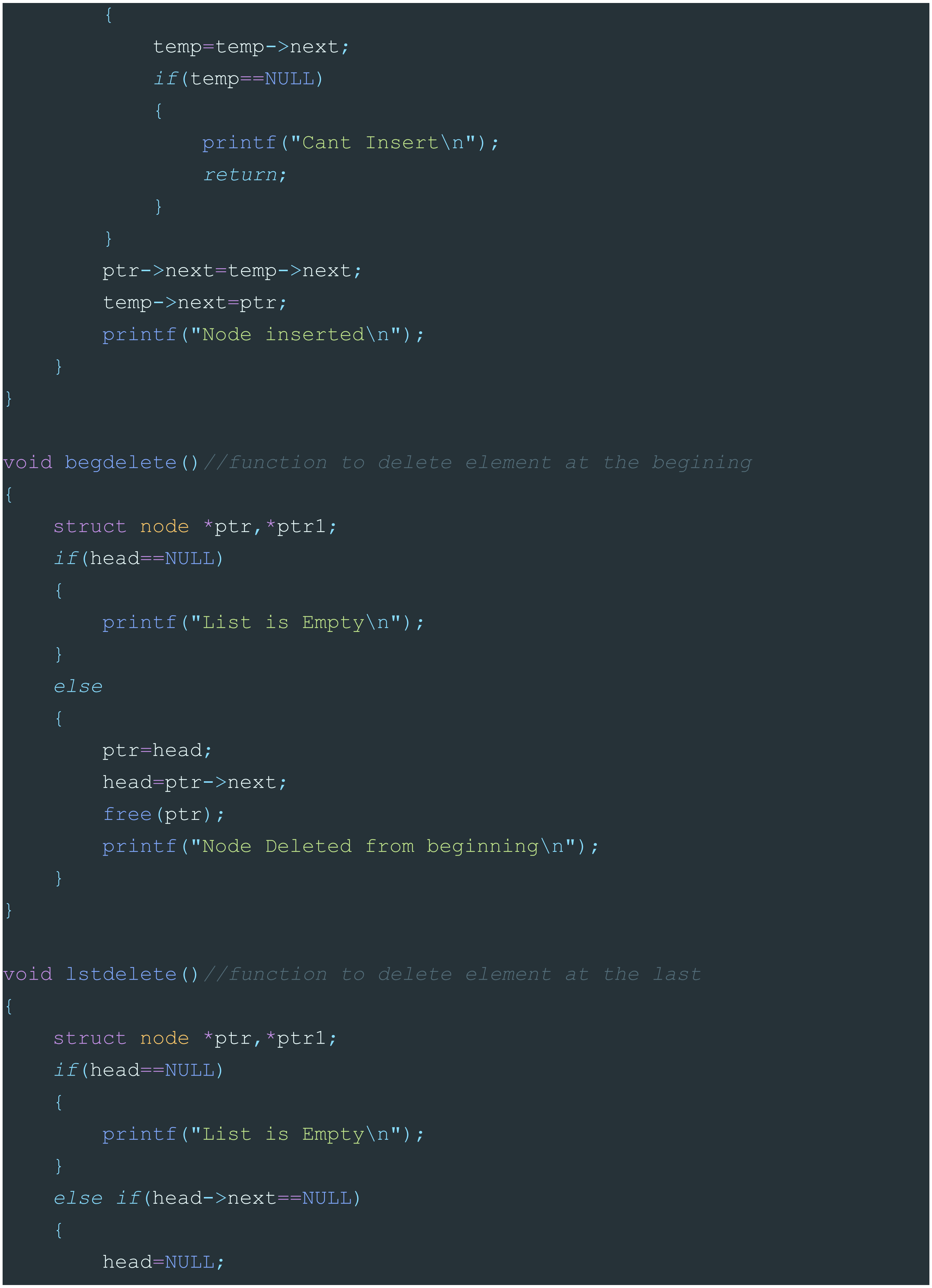
- Firstly take input from user about the location where the node is to be inserted
- Now traverse to the location while keeping another pointer that is set the element
- If while traversing the to the location the list ends, then return the control flow and display appropriate error
- If ptr successfully traverses to the specified location, set the previous pointer’s next element to the traversing pointers next element
- Finally free the memory block that the traversing pointer holds

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

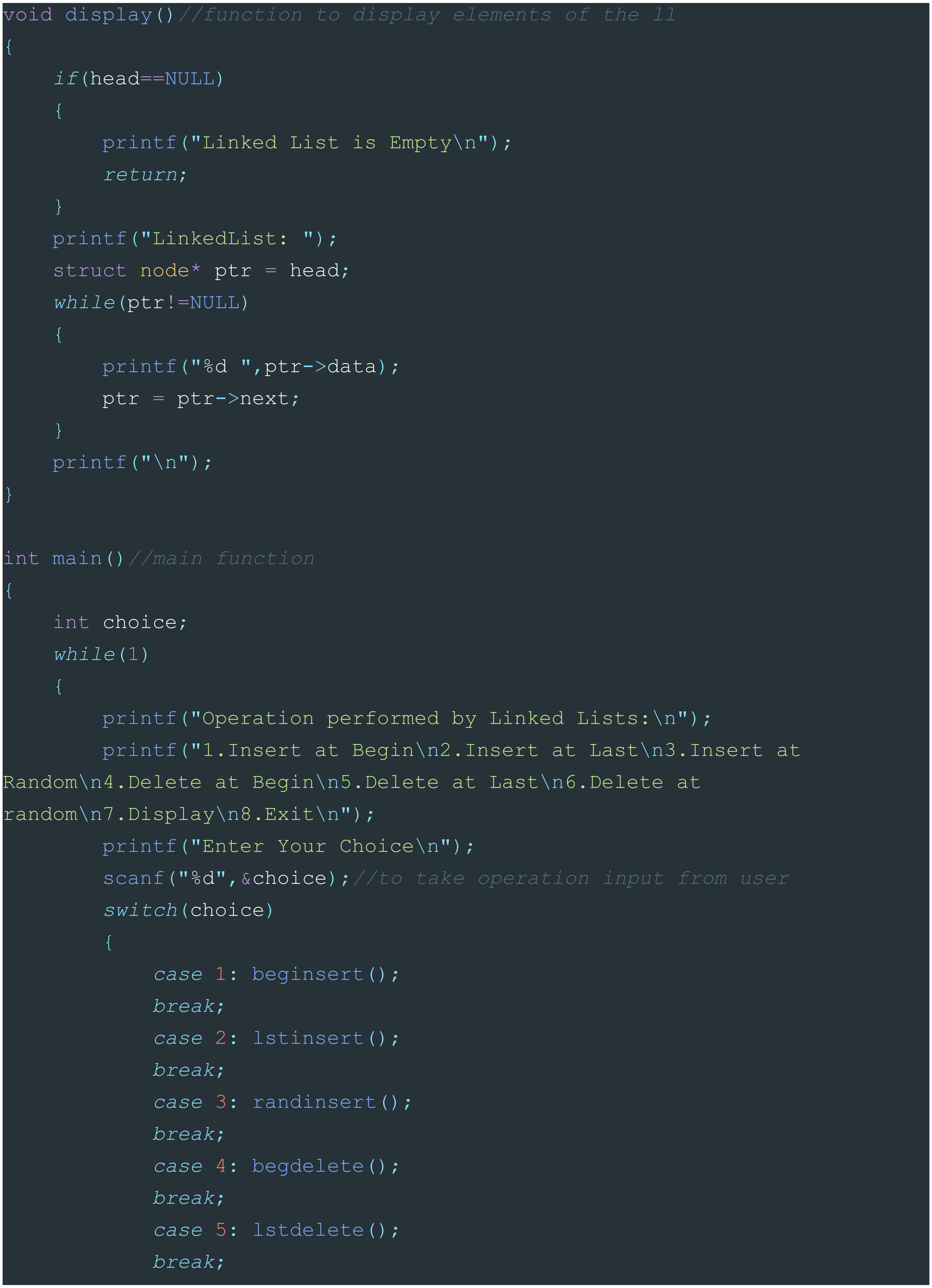
```
struct node
{
    int data;
    struct node *next;
};
struct node *head;

void beginsert()//function to insert element at the begining
{
    struct node *ptr;
    int item;
    ptr=(struct node *)malloc(sizeof(struct node *));
    if(ptr==NULL)
    {
        printf("Overflow\n");
    }
    else
    {
        printf("Enter Value:");
        scanf("%d",&item);
        ptr->data=item;
        ptr->next=head;
        head=ptr;
        printf("Node Inserted\n");
    }
}

void lstinsert()//function to insert element at the last
{
    struct node *ptr,*temp;
    int item;
    ptr=(struct node *)malloc(sizeof(struct node *));
    if(ptr==NULL)
    {
        printf("OverFlow\n");
    }
    else
    {
        printf("Enter Value:");
```



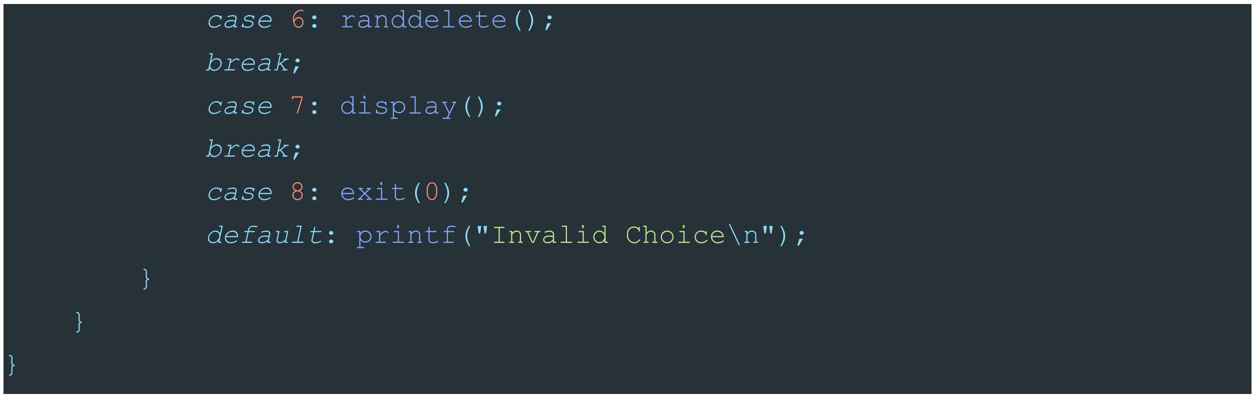
```
        scanf("%d",&item);
        ptr->data=item;
        if(head==NULL)
        {
            ptr->next=NULL;
            head=ptr;
            printf("Node Inserted\n");
        }
        else
        {
            temp=head;
            while(temp->next!=NULL)
            {
                temp=temp->next;
            }
            temp->next=ptr;
            ptr->next=NULL;
            printf("Node Inserted\n");
        }
    }
}

void randinsert()//function to insert element at the desired location
{
    struct node *ptr,*temp;
    int item,loc,i;
    ptr=(struct node *)malloc(sizeof(struct node *));
    if(ptr==NULL)
    {
        printf("Overflow");
    }
    else
    {
        printf("Enter Value:");
        scanf("%d",&item);
        ptr->data=item;
        printf("\nEnter the location after which you want to insert\n");
        scanf("%d",&loc);
        temp=head;
        for(i=0;i<loc;i++)
```



```
        {
            temp=temp->next;
            if(temp==NULL)
            {
                printf("Cant Insert\n");
                return;
            }
        }
        ptr->next=temp->next;
        temp->next=ptr;
        printf("Node inserted\n");
    }
}

void begdelete()//function to delete element at the begining
{
    struct node *ptr,*ptr1;
    if(head==NULL)
    {
        printf("List is Empty\n");
    }
    else
    {
        ptr=head;
        head=ptr->next;
        free(ptr);
        printf("Node Deleted from beginning\n");
    }
}

void lstdelete()//function to delete element at the last
{
    struct node *ptr,*ptr1;
    if(head==NULL)
    {
        printf("List is Empty\n");
    }
    else if(head->next==NULL)
    {
        head=NULL;
        free(head);
        printf("Node deleted\n");
    }
    else
    {
        ptr=head;
        while(ptr->next!=NULL)
        {
            ptr1=ptr;
            ptr=ptr->next;
        }
        ptr1->next=NULL;
        free(ptr);
        printf("Deleted node from the last\n");
    }
}

void randdelete()//function to delete element at the desired location
{
    struct node *ptr,*ptr1;
    int loc,i;
    printf("Enter the location of the node after which it has to be 
deleted\n");
    scanf("%d",&loc);
    ptr=head;
    for(i=0;i<loc;i++)
    {
        ptr1=ptr;
        ptr=ptr->next;
        if(ptr==NULL)
        {
            printf("deletion isn't possible \n");
            return;
        }
    }
    ptr1->next=ptr->next;
    free(ptr);
    printf("Delete Node %d ",loc+1);
}
```

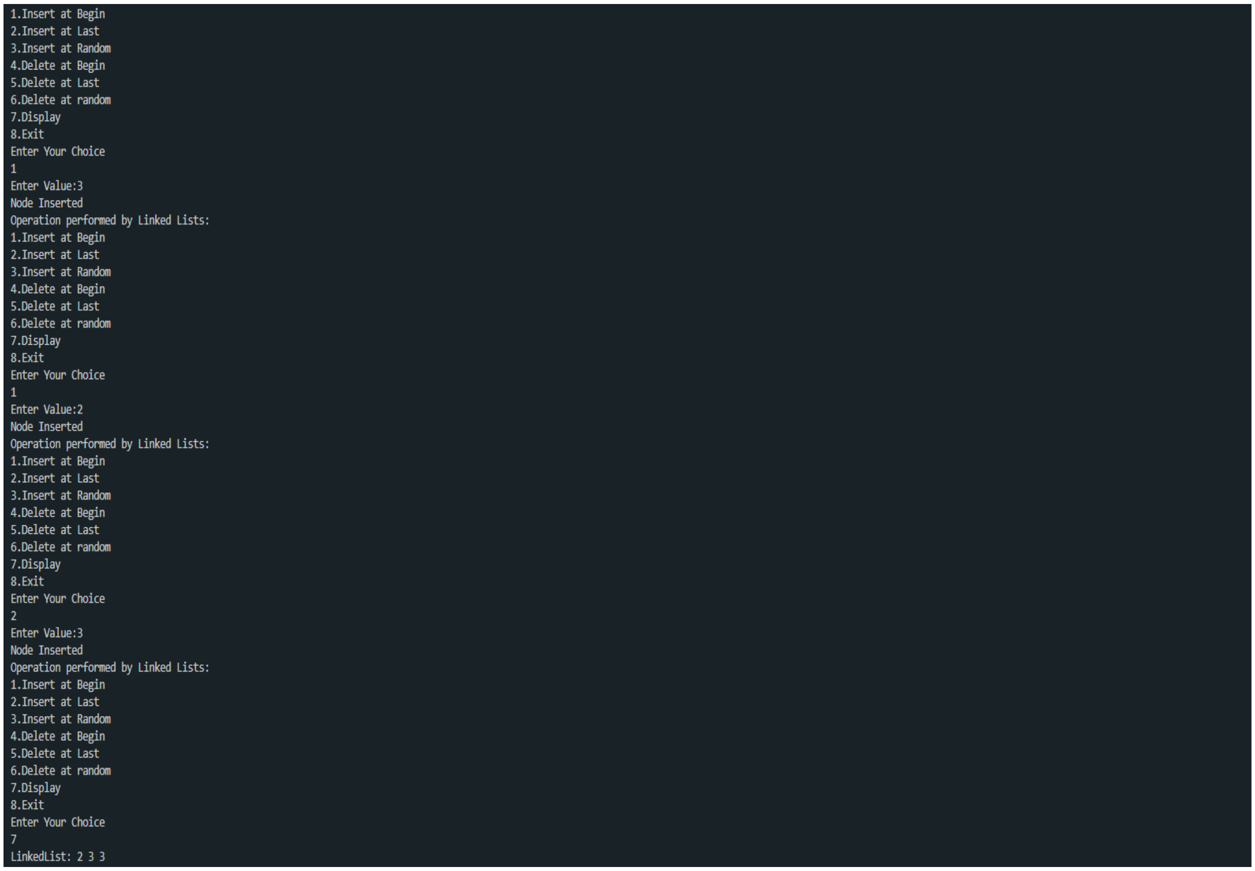


```
void display()//function to display elements of the ll
{
    if(head==NULL)
    {
        printf("Linked List is Empty\n");
        return;
    }
    printf("LinkedList: ");
    struct node* ptr = head;
    while(ptr!=NULL)
    {
        printf("%d ",ptr->data);
        ptr = ptr->next;
    }
    printf("\n");
}

int main()//main function
{
    int choice;
    while(1)
    {
        printf("Operation performed by Linked Lists:\n");
        printf("1.Insert at Begin\n2.Insert at Last\n3.Insert at
Random\n4.Delete at Begin\n5.Delete at Last\n6.Delete at
random\n7.Display\n8.Exit\n");
        printf("Enter Your Choice\n");
        scanf("%d",&choice);//to take operation input from user
        switch(choice)
        {
            case 1: beginsert();
            break;
            case 2: lstinsert();
            break;
            case 3: randinsert();
            break;
            case 4: begdelete();
            break;
            case 5: lstdelete();
            break;
```



```
                case 6: randdelete();
                break;
                case 7: display();
                break;
                case 8: exit(0);
                default: printf("Invalid Choice\n");
            }
        }
}
```

Output



```
1.Insert at Begin
2.Insert at Last
3.Insert at Random
4.Delete at Begin
5.Delete at Last
6.Delete at random
7.Display
8.Exit
Enter Your Choice
1
Enter Value:3
Node Inserted
Operation performed by Linked Lists:
1.Insert at Begin
2.Insert at Last
3.Insert at Random
4.Delete at Begin
5.Delete at Last
6.Delete at random
7.Display
8.Exit
Enter Your Choice
1
Enter Value:2
Node Inserted
Operation performed by Linked Lists:
1.Insert at Begin
2.Insert at Last
3.Insert at Random
4.Delete at Begin
5.Delete at Last
6.Delete at random
7.Display
8.Exit
Enter Your Choice
2
Enter Value:3
Node Inserted
Operation performed by Linked Lists:
1.Insert at Begin
2.Insert at Last
3.Insert at Random
4.Delete at Begin
5.Delete at Last
6.Delete at random
7.Display
8.Exit
Enter Your Choice
7
LinkedList: 2 3 3
```