Display a Linked List

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

struct Node
{
    int data;
    struct Node *next;
}*first=NULL;

void create(int A[],int n)
{
    int i;
    struct Node *t,*last;
    first=(struct Node *)malloc(sizeof(struct Node));
    first->data=A[0];
    first->next=NULL;
    last=first;

    for(i=1;i<n;i++)
    {
        t=(struct Node*)malloc(sizeof(struct Node));
        t->data=A[i];
        t->next=NULL;
        last->next=t;
        last=t;
    }
}

void Display(struct Node *p)
{
    while(p!=NULL)
    {
        printf("%d ",p->data);
        p=p->next;
    }
```

```c
}

void RDisplay(struct Node *p)
{
    if(p!=NULL)
    {
        RDisplay(p->next);
        printf("%d ",p->data);

    }
}


int main()
{
    struct Node *temp;
    int A[]={3,5,7,10,25,8,32,2};
    create(A,8);


    Display(first);

    return 0;
}
```

Display a Linked List

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

struct Node
{
    int data;
    struct Node *next;
}*first=NULL;

void create(int A[],int n)
{
    int i;
    struct Node *t,*last;
    first=(struct Node *)malloc(sizeof(struct Node));
    first->data=A[0];
    first->next=NULL;
    last=first;

    for(i=1;i<n;i++)
    {
        t=(struct Node*)malloc(sizeof(struct Node));
        t->data=A[i];
        t->next=NULL;
        last->next=t;
        last=t;
    }
}

void Display(struct Node *p)
{
    while(p!=NULL)
    {
        printf("%d ",p->data);
        p=p->next;
    }
}

void RDisplay(struct Node *p)
{
    if(p!=NULL)
    {
        RDisplay(p->next);
        printf("%d ",p->data);

    }
}


int main()
{
    struct Node *temp;
    int A[]={3,5,7,10,25,8,32,2};
    create(A,8);


    Display(first);

    return 0;
}
```

Count and Sum Linked List

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

struct Node
{
    int data;
    struct Node *next;
}*first=NULL;

void create(int A[],int n)
{
    int i;
    struct Node *t,*last;
    first=(struct Node *)malloc(sizeof(struct Node));
    first->data=A[0];
    first->next=NULL;
    last=first;

    for(i=1;i<n;i++)
    {
        t=(struct Node*)malloc(sizeof(struct Node));
        t->data=A[i];
        t->next=NULL;
        last->next=t;
        last=t;
    }
}

int count(struct Node *p)
{
    int l=0;
    while(p)
    {
        l++;
        p=p->next;
    }
    return l;
}

int Rcount(struct Node *p)
{
    if(p!=NULL)
        return Rcount(p->next)+1;
    else
        return 0;
```

```
}
int sum(struct Node *p)
{
    int s=0;

    while(p!=NULL)
    {
        s+=p->data;
        p=p->next;
    }
    return s;
}

int Rsum(struct Node *p)
{
    if(p==NULL)
        return 0;
    else
        return Rsum(p->next)+p->data;
}


int main()
{

    int A[]={3,5,7,10,25,8,32,2};
    create(A,8);

    printf(“Count %d\n”,count(first));
    printf(“Sum %d\n”,sum(first);



    return 0;
}
```

Max element from Linked List

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

struct Node
{
    int data;
    struct Node *next;
}*first=NULL;

void create(int A[],int n)
{
    int i;
    struct Node *t,*last;
    first=(struct Node *)malloc(sizeof(struct Node));
    first->data=A[0];
    first->next=NULL;
    last=first;

    for(i=1;i<n;i++)
    {
        t=(struct Node*)malloc(sizeof(struct Node));
        t->data=A[i];
        t->next=NULL;
        last->next=t;
        last=t;
    }
}

int Max(struct Node *p)
{
    int max=INT32_MIN;

    while(p)
    {
        if(p->data>max)
            max=p->data;
        p=p->next;
    }
    return max;

}

int RMax(struct Node *p)
{
    int x=0;
    if(p==0)
        return INT32_MIN;
    x=RMax(p->next);
    if(x>p->data)
        return x;
    else
        return p->data;
}


int main()
{

    int A[]={3,5,7,10,25,8,32,2};
    create(A,8);

   printf(“Max %d\n”,Max(first);



    return 0;
}
```

Display a Linked List

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

struct Node
{
    int data;
    struct Node *next;
}*first=NULL;

void create(int A[],int n)
{
    int i;
    struct Node *t,*last;
    first=(struct Node *)malloc(sizeof(struct Node));
    first->data=A[0];
    first->next=NULL;
    last=first;

    for(i=1;i<n;i++)
    {
        t=(struct Node*)malloc(sizeof(struct Node));
        t->data=A[i];
        t->next=NULL;
        last->next=t;
        last=t;
    }
}

struct Node * LSearch(struct Node *p,int key)
{
    struct Node *q;

    while(p!=NULL)
    {
        if(key==p->data)
        {
            q->next=p->next;
            p->next=first;
            first=p;
            return p;
        }
        q=p;
        p=p->next;
    }
    return NULL;

}

struct Node * RSearch(struct Node *p,int key)
{
    if(p==NULL)
        return NULL;
    if(key==p->data)
        return p;
    return RSearch(p->next,key);

}



int main()
{
    struct Node *temp;
    int A[]={3,5,7,10,25,8,32,2};
    create(A,8);

    temp=Search(first,8);
    printf(“%d”,temp->data);

    return 0;
}
```

Linked List

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

struct Node
{
    int data;
    struct Node *next;
}*first=NULL;

void create(int A[],int n)
{
    int i;
    struct Node *t,*last;
    first=(struct Node *)malloc(sizeof(struct Node));
    first->data=A[0];
    first->next=NULL;
    last=first;

    for(i=1;i<n;i++)
    {
        t=(struct Node*)malloc(sizeof(struct Node));
        t->data=A[i];
        t->next=NULL;
        last->next=t;
        last=t;
    }
}

void Display(struct Node *p)
{
    while(p!=NULL)
    {
        printf("%d ",p->data);
        p=p->next;
    }
```

```
}

void Insert(struct Node *p,int index,int x)
{
    struct Node *t;
    int i;

    if(index < 0 || index > count(p))
        return;
    t=(struct Node *)malloc(sizeof(struct Node));
    t->data=x;

    if(index == 0)
    {
        t->next=first;
        first=t;
    }
    else
    {
        for(i=0;i<index-1;i++)
            p=p->next;
        t->next=p->next;
        p->next=t;

    }


}




int main()
{

    int A[]={10,20,30,40,50};
    create(A,5);



    Insert(first,0,5);
    Display(first);
    return 0;
}
```

Insert and create a Linked List

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

struct Node
{
    int data;
    struct Node *next;
}*first=NULL;

void create(int A[],int n)
{
    int i;
    struct Node *t,*last;
    first=(struct Node *)malloc(sizeof(struct Node));
    first->data=A[0];
    first->next=NULL;
    last=first;

    for(i=1;i<n;i++)
    {
        t=(struct Node*)malloc(sizeof(struct Node));
        t->data=A[i];
        t->next=NULL;
        last->next=t;
        last=t;
    }
}

void Display(struct Node *p)
{
    while(p!=NULL)
    {
        printf("%d ",p->data);
        p=p->next;
    }
}

void Insert(struct Node *p,int index,int x)
{
    struct Node *t;
    int i;

    if(index < 0 || index > count(p))
        return;
    t=(struct Node *)malloc(sizeof(struct Node));
    t->data=x;

    if(index == 0)
    {
        t->next=first;
        first=t;
    }
    else
    {
        for(i=0;i<index-1;i++)
            p=p->next;
        t->next=p->next;
        p->next=t;

    }
}
int main()
{

    Insert(first,0,15);
    Insert(first,0,8);
    Insert(first,0,9);
    Insert(first,1,10);

    Display(first);



    return 0;
}
```

Inserting in a Sorted Linked List

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

struct Node
{
    int data;
    struct Node *next;
}*first=NULL,*second=NULL,*third=NULL;

void Display(struct Node *p)
{
    while(p!=NULL)
    {
        printf("%d ",p->data);
        p=p->next;
    }
}



void create(int A[],int n)
{
    int i;
    struct Node *t,*last;
    first=(struct Node *)malloc(sizeof(struct Node));
    first->data=A[0];
    first->next=NULL;
    last=first;

    for(i=1;i<n;i++)
    {
        t=(struct Node*)malloc(sizeof(struct Node));
        t->data=A[i];
        t->next=NULL;
        last->next=t;
        last=t;
    }
}


void SortedInsert(struct Node *p,int x)
{
    struct Node *t,*q=NULL;
    t=(struct Node*)malloc(sizeof(struct Node));
    t->data=x;
    t->next=NULL;

    if(first==NULL)
        first=t;
    else
    {
        while(p && p->data<x)
        {
            q=p;
            p=p->next;
        }
        if(p==first)
        {
            t->next=first;
            first=t;
        }
        else
        {
            t->next=q->next;
            q->next=t;
        }
    }

}

int main()
{

    int A[]={10,20,30,40,50};
    create(A,5);


    printf(“%d\n”,SortedInsert(first,15));
    Display(first);


    return 0;
}
```

Linked List

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

struct Node
{
    int data;
    struct Node *next;
}*first=NULL;

void create(int A[],int n)
{
    int i;
    struct Node *t,*last;
    first=(struct Node *)malloc(sizeof(struct Node));
    first->data=A[0];
    first->next=NULL;
    last=first;

    for(i=1;i<n;i++)
    {
        t=(struct Node*)malloc(sizeof(struct Node));
        t->data=A[i];
        t->next=NULL;
        last->next=t;
        last=t;
    }
}


void Display(struct Node *p)
{
    while(p!=NULL)
    {
        printf("%d ",p->data);
        p=p->next;
    }
}

void RDisplay(struct Node *p)
{
    if(p!=NULL)
    {
        RDisplay(p->next);
        printf("%d ",p->data);

    }
}


int Delete(struct Node *p,int index)
{
    struct Node *q=NULL;
    int x=-1,i;

    if(index < 1 || index > count(p))
        return -1;
    if(index==1)
    {
        q=first;
        x=first->data;
        first=first->next;
        free(q);
        return x;
    }
    else
    {
        for(i=0;i<index-1;i++)
        {
            q=p;
            p=p->next;
        }
        q->next=p->next;
        x=p->data;
        free(p);
        return x;
    }


}

int main()
{

    int A[]={10,20,30,40,50};
    create(A,5);

    printf(“%d\n",Delete(first),2);
    Display(first);

    return 0;
}
```

Checking is a Linked List is Sorted

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

struct Node
{
    int data;
    struct Node *next;
}*first=NULL,*second=NULL,*third=NULL;

void Display(struct Node *p)
{
    while(p!=NULL)
    {
        printf("%d ",p->data);
        p=p->next;
    }
}



void create(int A[],int n)
{
    int i;
    struct Node *t,*last;
    first=(struct Node *)malloc(sizeof(struct Node));
    first->data=A[0];
    first->next=NULL;
    last=first;

    for(i=1;i<n;i++)
    {
        t=(struct Node*)malloc(sizeof(struct Node));
        t->data=A[i];
        t->next=NULL;
        last->next=t;
        last=t;
    }
}


int isSorted(struct Node *p)
{
    int x=-65536;
    while(p!=NULL)
    {
        if(p->data < x)
            return 0;
        x=p->data;
        p=p->next;
    }
    return 1;

}

int main()
{

    int A[]={10,20,30,40,50};
    create(A,5);


    printf("%d\n",isSorted(first));


    return 0;
}
```

Remove Duplicates from Linked List

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

struct Node
{
    int data;
    struct Node *next;
}*first=NULL,*second=NULL,*third=NULL;

void Display(struct Node *p)
{
    while(p!=NULL)
    {
        printf("%d ",p->data);
        p=p->next;
    }
}




void create(int A[],int n)
{
    int i;
    struct Node *t,*last;
    first=(struct Node *)malloc(sizeof(struct Node));
    first->data=A[0];
    first->next=NULL;
    last=first;

    for(i=1;i<n;i++)
    {
        t=(struct Node*)malloc(sizeof(struct Node));
        t->data=A[i];
        t->next=NULL;
        last->next=t;
        last=t;
    }
}


void RemoveDuplicate(struct Node *p)
{
    struct Node *q=p->next;
    while(q!=NULL)
    {
        if(p->data!=q->data)
        {
            p=q;
            q=q->next;
        }
        else
        {
            p->next=q->next;
            free(q);
            q=p->next;
        }
    }

}


int main()
{

    int A[]={10,20,20,40,50,50,50,60};
    create(A,8);


    RemoveDuplicate(frist);
    Display(frist);

    return 0;
}
```

Reverse a Linked List

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

struct Node
{
    int data;
    struct Node *next;
}*first=NULL,*second=NULL,*third=NULL;

void Display(struct Node *p)
{
    while(p!=NULL)
    {
        printf("%d ",p->data);
        p=p->next;
    }
}




void create(int A[],int n)
{
    int i;
    struct Node *t,*last;
    first=(struct Node *)malloc(sizeof(struct Node));
    first->data=A[0];
    first->next=NULL;
    last=first;

    for(i=1;i<n;i++)
    {
        t=(struct Node*)malloc(sizeof(struct Node));
        t->data=A[i];
        t->next=NULL;
        last->next=t;
        last=t;
    }
}


void Reverse1(struct Node *p)
{
    int *A,i=0;
    struct Node *q=p;
    A=(int *)malloc(sizeof(int)*count(p));

    while(q!=NULL)
    {
        A[i]=q->data;
        q=q->next;
        i++;
    }
    q=p;
    i--;
    while(q!=NULL)
    {
        q->data=A[i];
        q=q->next;
        i--;
    }
}

void Reverse2(struct Node *p)
{
    struct Node *q=NULL,*r=NULL;

    while(p!=NULL)
    {
        r=q;
        q=p;
        p=p->next;
        q->next=r;
    }
    first=q;
}

void Reverse3(struct Node *q,struct Node *p)
{
    if(p)
    {
        Reverse3(p,p->next);
        p->next=q;
    }
    else
        first=q;
}


int main()
{
    int A[]={10,20,40,50,60};
    create(A,5);


    Reverse1(frist);
    Display(frist);

    return 0;
}
```

Merge two Linked List

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

struct Node
{
    int data;
    struct Node *next;
}*first=NULL,*second=NULL,*third=NULL;

void Display(struct Node *p)
{
    while(p!=NULL)
    {
        printf("%d ",p->data);
        p=p->next;
    }
}



void create(int A[],int n)
{
    int i;
    struct Node *t,*last;
    first=(struct Node *)malloc(sizeof(struct Node));
    first->data=A[0];
    first->next=NULL;
    last=first;

    for(i=1;i<n;i++)
    {
        t=(struct Node*)malloc(sizeof(struct Node));
        t->data=A[i];
        t->next=NULL;
        last->next=t;
        last=t;
    }
}

void create2(int A[],int n)
{
    int i;
    struct Node *t,*last;
    second=(struct Node *)malloc(sizeof(struct Node));
    second->data=A[0];
    second->next=NULL;
    last=second;

    for(i=1;i<n;i++)
    {
        t=(struct Node*)malloc(sizeof(struct Node));
        t->data=A[i];
        t->next=NULL;
        last->next=t;
        last=t;
    }
}


void Merge(struct Node *p,struct Node *q)
{
    struct Node *last;
    if(p->data < q->data)
    {
        third=last=p;
        p=p->next;
        third->next=NULL;
    }
    else
    {
        third=last=q;
        q=q->next;
        third->next=NULL;
    }
    while(p && q)
    {
        if(p->data < q->data)
        {
            last->next=p;
            last=p;
            p=p->next;
            last->next=NULL;

        }
        else
        {
            last->next=q;
            last=q;
            q=q->next;
            last->next=NULL;

        }
    }
    if(p)last->next=p;
    if(q)last->next=q;

}

int main()
{

    int A[]={10,20,40,50,60};
    int B[]={15,18,25,30,55};
    create(A,5);
    create2(B,5);


    Merge(frist,second);
    Display(third);

    return 0;
}
```

Checking for Loop Linked List

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

struct Node
{
    int data;
    struct Node *next;
}*first=NULL,*second=NULL,*third=NULL;

void Display(struct Node *p)
{
    while(p!=NULL)
    {
        printf("%d ",p->data);
        p=p->next;
    }
}



void create(int A[],int n)
{
    int i;
    struct Node *t,*last;
    first=(struct Node *)malloc(sizeof(struct Node));
    first->data=A[0];
    first->next=NULL;
    last=first;

    for(i=1;i<n;i++)
    {
        t=(struct Node*)malloc(sizeof(struct Node));
        t->data=A[i];
        t->next=NULL;
        last->next=t;
        last=t;
    }
}

int isLoop(struct Node *f)
{
    struct Node *p,*q;
    p=q=f;
    do
    {
        p=p->next;
        q=q->next;
        q=q?q->next:q;
    }while(p && q && p!=q);

    if(p==q)
        return 1;
    else
        return 0;
}


int main()
{
    struct Node *t1,*t2;

    int A[]={10,20,30,40,50};
    create(A,5);

    t1=first->next->next;
    t2=first->next->next->next->next;
    t2->next=t1;

    printf("%d\n",isLoop(first));


    return 0;
}
```

Linked List CPP

```
#include <iostream>

using namespace std;

class Node
{
public:
    int data;
    Node *next;
};

class LinkedList
{
private:
    Node *first;
public:
    LinkedList(){first=NULL;}
    LinkedList(int A[],int n);
    ~LinkedList();

    void Display();
    void Insert(int index,int x);
    int Delete(int index);
    int Length();

};

LinkedList::LinkedList(int A[],int n)
{
    Node *last,*t;
    int i=0;

    first=new Node;
    first->data=A[0];
    first->next=NULL;
    last=first;
    for(i=1;i<n;i++)
    {
        t=new Node;
        t->data=A[i];
        t->next=NULL;
        last->next=t;
        last=t;
    }
}
LinkedList::~LinkedList()
{
    Node *p=first;
    while(first)
    {
        first=first->next;
        delete p;
        p=first;
    }
}
void LinkedList::Display()
{
    Node *p=first;

    while(p)
    {
        cout<<p->data<<" ";
        p=p->next;
    }
    cout<<endl;
}
int LinkedList::Length()
{
    Node *p=first;
    int len=0;

    while(p)
    {
        len++;
        p=p->next;
    }
```

```cpp
    return len;
}

void LinkedList::Insert(int index,int x)
{
    Node *t,*p=first;

    if(index <0 || index > Length())
        return;
    t=new Node;
    t->data=x;
    t->next=NULL;

    if(index==0)
    {
        t->next=first;
        first=t;
    }
    else
    {
        for(int i=0;i<index-1;i++)
            p=p->next;
        t->next=p->next;
        p->next=t;
    }
}

int LinkedList::Delete(int index)
{
    Node *p,*q=NULL;
    int x=-1;

    if(index < 1 || index > Length())
        return -1;
    if(index==1)
    {
        p=first;
        first=first->next;
        x=p->data;
        delete p;
    }
    else
    {
        p=first;
        for(int i=0;i<index-1;i++)
        {
            q=p;
            p=p->next;
        }
        q->next=p->next;
        x=p->data;
        delete p;
    }
    return x;
}


int main()
{
    int A[]={1,2,3,4,5};
    LinkedList l(A,5);

    l.Insert(3,10);

    l.Display();

    return 0;
}
```

Circular Linked List

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

struct Node
{
    int data;
    struct Node *next;
}*Head;


void create(int A[],int n)
{
    int i;
    struct Node *t,*last;
    Head=(struct Node*)malloc(sizeof(struct Node));
    Head->data=A[0];
    Head->next=Head;
    last=Head;

    for(i=1;i<n;i++)
    {
        t=(struct Node*)malloc(sizeof(struct Node));
        t->data=A[i];
        t->next=last->next;
        last->next=t;
        last=t;
    }
}

void Display(struct Node *h)
{
    do
    {
        printf("%d ",h->data);
        h=h->next;
    }while(h!=Head);
    printf("\n");
}

void RDisplay(struct Node *h)
{
    static int flag=0;
    if(h!=Head || flag==0)
    {
        flag=1;
        printf("%d ",h->data);
        RDisplay(h->next);
    }
    flag=0;
}

int Length(struct Node *p)
{
    int len=0;
    do
    {
        len++;
        p=p->next;

    }while(p!=Head);
    return len;
}

void Insert(struct Node *p,int index, int x)
{
    struct Node *t;
    int i;
    if(index<0 || index > Length(p))
        return;

    if(index==0)
    {
        t=(struct Node *)malloc(sizeof(struct Node));
        t->data=x;
        if(Head==NULL)
        {
            Head=t;
            Head->next=Head;
        }
        else
        {
            while(p->next!=Head)p=p->next;
            p->next=t;
            t->next=Head;
            Head=t;
        }

    }
    else
    {
        for(i=0;i<index-1;i++)p=p->next;
        t=(struct Node *)malloc(sizeof(struct Node));
        t->data=x;
        t->next=p->next;
        p->next=t;

    }
}

int Delete(struct Node *p,int index)
{
    struct Node *q;
    int i,x;

    if(index <0 || index >Length(Head))
        return -1;
    if(index==1)
    {
        while(p->next!=Head)p=p->next;
        x=Head->data;
        if(Head==p)
        {
            free(Head);
            Head=NULL;
        }
        else
        {
            p->next=Head->next;
            free(Head);
            Head=p->next;
        }
    }
    else
    {
        for(i=0;i<index-2;i++)
            p=p->next;
        q=p->next;
        p->next=q->next;
        x=q->data;
        free(q);
    }
    return x;
}

int main()
{
    int A[]={2,3,4,5,6};
    create(A,5);

    Delete(Head,8);

    RDisplay(Head);
    return 0;
}
```

Doubly Linked List

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


struct Node
{
    struct Node *prev;
    int data;
    struct Node *next;
}*first=NULL;

void create(int A[],int n)
{
    struct Node *t,*last;
    int i;

    first=(struct Node *)malloc(sizeof(struct Node));
    first->data=A[0];
    first->prev=first->next=NULL;
    last=first;

    for(i=1;i<n;i++)
    {
        t=(struct Node *)malloc(sizeof(struct Node));
        t->data=A[i];
        t->next=last->next;
        t->prev=last;
        last->next=t;
        last=t;
    }
}

void Display(struct Node *p)
{
    while(p)
    {
        printf("%d ",p->data);
        p=p->next;
    }
    printf("\n");
}

int Length(struct Node *p)
{
    int len=0;

    while(p)
    {
        len++;
        p=p->next;
    }
    return len;
}

void Insert(struct Node *p,int index,int x)
{
    struct Node *t;
    int i;

    if(index < 0 || index > Length(p))
        return;
    if(index==0)
    {
        t=(struct Node *)malloc(sizeof(struct Node));
        t->data=x;
        t->prev=NULL;
        t->next=first;
        first->prev=t;
        first=t;
    }
    else
    {
        for(i=0;i<index-1;i++)
            p=p->next;
        t=(struct Node *)malloc(sizeof(struct Node));
        t->data=x;
        t->prev=p;
        t->next=p->next;
        if(p->next)p->next->prev=t;
        p->next=t;

    }
}

int Delete(struct Node *p,int index)
{
    //struct Node *q;
    int x=-1,i;

    if(index < 1 || index > Length(p))
        return -1;

    if(index==1)
    {
        first=first->next;
        if(first)first->prev=NULL;

        x=p->data;
        free(p);
    }
    else
    {
        for(i=0;i<index-1;i++)
            p=p->next;
        p->prev->next=p->next;
        if(p->next)
            p->next->prev=p->prev;
        x=p->data;
        free(p);
    }
    return x;

}

void Reverse(struct Node *p)
```

```c
{
    struct Node *temp;

    while(p!=NULL)
    {
        temp=p->next;
        p->next=p->prev;
        p->prev=temp;
        p=p->prev;
        if(p!=NULL && p->next==NULL)
            first=p;
    }
}

int main()
{
    int A[]={10,20,30,40,50};
    create(A,5);

    Reverse(first);

    Display(first);

    return 0;
}
```

Polynomial Linked List

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

struct Node
{
    int coeff;
    int exp;
    struct Node *next;
}*poly=NULL;

void create()
{
    struct Node *t,*last=NULL;
    int num,i;

    printf("Enter number of terms");
    scanf("%d",&num);
    printf("Enter each term with coeff and exp\n");

    for(i=0;i<num;i++)
    {
        t=(struct Node *)malloc(sizeof(struct Node));
        scanf("%d%d",&t->coeff,&t->exp);
        t->next=NULL;
        if(poly==NULL)
        {
            poly=last=t;
        }
        else
        {
            last->next=t;
            last=t;
        }
    }
}
```

```c
void Display(struct Node *p)
{
    while(p)
    {
        printf("%dx%d +",p->coeff,p->exp);
        p=p->next;
    }
    printf("\n");
}

long Eval(struct Node *p, int x)
{
    long val=0;

    while(p)
    {
        val+=p->coeff*pow(x,p->exp);
        p=p->next;
    }
    return val;
}

int main()
{
    create();
    Display(poly);
    printf("%ld\n",Eval(poly,1));

    return 0;
}
```

Stack Class

```
#include <iostream>
using namespace std;

template<class T>
class Stack
{
private:
    T *st;
    int size;
    int top;
public:
    Stack(){size=10;top=-1;st=new T[size];}
    Stack(int size){this->size=size;top=-1;st=new
T[this->size];}

    void push(T x);
    T pop();
    T peek(int index);
    int stacktop();
    int isEmpty();
    int isFull();
    void Display();
};

template<class T>
void Stack<T>::push(T x)
{
    if(isFull())
        cout<<"Stack Overflow"<<endl;
    else
    {
        top++;
        st[top]=x;
    }
}
```

```
template<class T>
T Stack<T>::pop()
{
    T x=-1;
    if(isEmpty())
        cout<<"Stack underlfow"<<endl;
    else
    {
        x=st[top];
        top--;
    }
    return x;
}

template<class T>
T Stack<T>::peek(int index)
{
    T x=-1;
    if(top-index+1<0)
        cout<<"Invalid Index"<<endl;
    else
        x=st[top-index+1];

    return x;
}

template<class T>
int Stack<T>::stacktop()
{
    if(isEmpty())
        return -1;
    return st[top];
}

template<class T>
int Stack<T>::isFull()
{
    return top==size-1;
}
```

```cpp
template<class T>
int Stack<T>::isEmpty()
{
    return top==-1;
}

template<class T>
void Stack<T>::Display()
{
    for(int i=top;i>=0;i--)
        cout<<st[i]<<" ";
    cout<<endl;
}

int main()
{
    Stack<char> stk(5);
    stk.push('a');
    stk.push('b');
    stk.push('c');
    stk.push('d');
    stk.push('e');
    stk.push('f');

    stk.Display();

    cout<<stk.peek(1)<<endl;


    return 0;
}
```

Stack using Array

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

struct Stack
{
    int size;
    int top;
    int *S;
};

void create(struct Stack *st)
{
    printf("Enter Size");
    scanf("%d",&st->size);
    st->top=-1;
    st->S=(int *)malloc(st->size*sizeof(int));
}

void Display(struct Stack st)
{
    int i;
    for(i=st.top;i>=0;i--)
        printf("%d ",st.S[i]);
    printf("\n");

}

void push(struct Stack *st,int x)
{
    if(st->top==st->size-1)
        printf("Stack overflow\n");
    else
    {
        st->top++;
        st->S[st->top]=x;
    }
```

```
}

int pop(struct Stack *st)
{
    int x=-1;

    if(st->top==-1)
        printf("Stack Underflow\n");
    else
    {
        x=st->S[st->top--];
    }
    return x;
}

int peek(struct Stack st,int index)
{
    int x=-1;
    if(st.top-index+1<0)
        printf("Invalid Index \n");
    x=st.S[st.top-index+1];

    return x;
}

int isEmpty(struct Stack st)
{
    if(st.top==-1)
        return 1;
    return 0;
}

int isFull(struct Stack st)
{
    return st.top==st.size-1;
}

int stackTop(struct Stack st)
{
    if(!isEmpty(st))
```

```c
        return st.S[st.top];
    return -1;
}


int main()
{
    struct Stack st;
    create(&st);

    push(&st,10);
    push(&st,20);
    push(&st,30);
    push(&st,40);

    printf("%d \n",peek(st,2));


    Display(st);

    return 0;
}
```

Stack using Linked List

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

struct Node
{
    int data;
    struct Node *next;
}*top=NULL;


void push(int x)
{
    struct Node *t;
    t=(struct Node*)malloc(sizeof(struct Node));

    if(t==NULL)
        printf("stack is full\n");
    else
    {
        t->data=x;
        t->next=top;
        top=t;
    }

}

int pop()
{
    struct Node *t;
    int x=-1;

    if(top==NULL)
        printf("Stack is Empty\n");
    else
    {
        t=top;
        top=top->next;
        x=t->data;
        free(t);
    }
    return x;
}

void Display()
{
    struct Node *p;
    p=top;
    while(p!=NULL)
    {
        printf("%d ",p->data);
        p=p->next;
    }
    printf("\n");
}



int main()
{
    push(10);
    push(20);
    push(30);

    Display();

    printf("%d ",pop());

    return 0;
}
```

Stack Linked List CPP

```
#include <iostream>
using namespace std;
class Node
{
public:
    int data;
    Node *next;
};

class Stack
{
private:
    Node *top;
public:
    Stack(){top=NULL;}
    void push(int x);
    int pop();
    void Display();
};

void Stack::push(int x)
```

```
{
    Node *t=new Node;
    if(t==NULL)
        cout<<"Stak is
Full\n";
    else
    {
        t->data=x;
        t->next=top;
        top=t;
    }
}

int Stack::pop()
{
    int x=-1;
    if(top==NULL)
        cout<<"Stack is
Empty\n";
    else
    {
        x=top->data;
        Node *t=top;
        top=top->next;
        delete t;
    }
    return x;

}

void Stack::Display()
{
    Node *p=top;
    while(p!=NULL)
    {
        cout<<p->data<<" ";
        p=p->next;
    }
    cout<<endl;
}

int main()
{

    Stack stk;
```

```cpp
    stk.push(10);
    stk.push(20);
    stk.push(30);

    stk.Display();
    cout<<stk.pop();
    return 0;
}
```

Parenthesis Matching

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

struct Node
{
    char data;
    struct Node *next;
}*top=NULL;


void push(char x)
{
    struct Node *t;
    t=(struct Node*)malloc(sizeof(struct Node));

    if(t==NULL)
        printf("stack is full\n");
    else
    {
        t->data=x;
        t->next=top;
        top=t;
    }

}

char pop()
{
    struct Node *t;
    char x=-1;

    if(top==NULL)
        printf("Stack is Empty\n");
    else
    {
        t=top;
        top=top->next;
```

```c
        x=t->data;
        free(t);
    }
    return x;
}

void Display()
{
    struct Node *p;
    p=top;
    while(p!=NULL)
    {
        printf("%d ",p->data);
        p=p->next;
    }
    printf("\n");
}

int isBalanced(char *exp)
{
    int i;

    for(i=0;exp[i]!='\0';i++)
    {
        if(exp[i]=='(')
            push(exp[i]);
        else if(exp[i]==')')
        {
            if(top==NULL)
                return 0;
            pop();
        }
    }
    if(top==NULL)
        return 1;
    else
        return 0;
}
```

```c
int main()
{
    char *exp="((a+b)*(c-d)))";

    printf("%d ",isBalanced(exp));

    return 0;
}
```

Infix to Postfix Conversion

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

struct Node
{
    char data;
    struct Node *next;
}*top=NULL;


void push(char x)
{
    struct Node *t;
    t=(struct Node*)malloc(sizeof(struct Node));

    if(t==NULL)
        printf("stack is full\n");
    else
    {
        t->data=x;
        t->next=top;
        top=t;
    }

}

char pop()
{
    struct Node *t;
    char x=-1;

    if(top==NULL)
        printf("Stack is Empty\n");
    else
    {
        t=top;
        top=top->next;
```

```c
        x=t->data;
        free(t);
    }
    return x;
}

void Display()
{
    struct Node *p;
    p=top;
    while(p!=NULL)
    {
        printf("%d ",p->data);
        p=p->next;
    }
    printf("\n");
}

int isBalanced(char *exp)
{
    int i;

    for(i=0;exp[i]!='\0';i++)
    {
        if(exp[i]=='(')
            push(exp[i]);
        else if(exp[i]==')')
        {
            if(top==NULL)
                return 0;
            pop();
        }
    }
    if(top==NULL)
        return 1;
    else
        return 0;
}

int pre(char x)
```

```
{
    if(x=='+' || x=='-')
        return 1;
    else if(x=='*' || x=='/')
            return 2;
    return 0;
}

int isOperand(char x)
{
    if(x=='+' || x=='-' || x=='*' || x=='/')
       return 0;
    else
       return 1;

}

char * InToPost(char *infix)
{
    int i=0,j=0;
    char *postfix;
    int len=strlen(infix);
    postfix=(char *)malloc((len+2)*sizeof(char));

    while(infix[i]!='\0')
    {
        if(isOperand(infix[i]))
            postfix[j++]=infix[i++];
        else
        {
            if(pre(infix[i])>pre(top->data))
                push(infix[i++]);
            else
            {
                postfix[j++]=pop();
            }
        }
    }
    while(top!=NULL)
        postfix[j++]=pop();
```

```c
    postfix[j]='\0';
    return postfix;
}

int main()
{
    char *infix="a+b*c-d/e";
    push('#');

    char *postfix=InToPost(infix);

    printf("%s ",postfix);


    return 0;
}
```

Evaluation of Postfix

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

struct Node
{
    int data;
    struct Node *next;
}*top=NULL;


void push(int x)
{
    struct Node *t;
    t=(struct Node*)malloc(sizeof(struct Node));

    if(t==NULL)
        printf("stack is full\n");
    else
    {
        t->data=x;
        t->next=top;
        top=t;
    }

}

int pop()
{
    struct Node *t;
    int x=-1;

    if(top==NULL)
        printf("Stack is Empty\n");
    else
    {
        t=top;
        top=top->next;
        x=t->data;
        free(t);
    }
    return x;
}

void Display()
{
    struct Node *p;
    p=top;
    while(p!=NULL)
    {
        printf("%d ",p->data);
        p=p->next;
    }
    printf("\n");
}

int isBalanced(char *exp)
{
    int i;

    for(i=0;exp[i]!='\0';i++)
    {
        if(exp[i]=='(')
            push(exp[i]);
        else if(exp[i]==')')
        {
            if(top==NULL)
                return 0;
            pop();
        }
    }
    if(top==NULL)
        return 1;
    else
        return 0;
}
```

```
int pre(char x)
{
    if(x=='+' || x=='-')
        return 1;
    else if(x=='*' || x=='/')
            return 2;
    return 0;
}

int isOperand(char x)
{
    if(x=='+' || x=='-' || x=='*' || x=='/')
       return 0;
    else
       return 1;

}

char * InToPost(char *infix)
{
    int i=0,j=0;
    char *postfix;
    long len=strlen(infix);
    postfix=(char *)malloc((len+2)*sizeof(char));

    while(infix[i]!='\0')
    {
        if(isOperand(infix[i]))
            postfix[j++]=infix[i++];
        else
        {
            if(pre(infix[i])>pre(top->data))
                push(infix[i++]);
            else
            {
                postfix[j++]=pop();
            }
        }
    }
    while(top!=NULL)
```

```
        postfix[j++]=pop();
    postfix[j]='\0';
    return postfix;
}

int Eval(char *postfix)
{
    int i=0;
    int x1,x2,r=0 ;

    for(i=0;postfix[i]!='\0';i++)
    {
        if(isOperand(postfix[i]))
        {
            push(postfix[i]-'0');
        }
        else
        {
            x2=pop();x1=pop();
            switch(postfix[i])
            {
                case '+':r=x1+x2; break;
                case '-':r=x1-x2; break;
                case '*':r=x1*x2; break;
                case '/':r=x1/x2; break;
            }
            push(r);
        }
    }
    return top->data;
}


int main()
{
    char *postfix="234*+82/-";

    printf("Result is %d\n",Eval(postfix));

    return 0;
```

```
}
```