Linked List

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

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

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

    while(p!=NULL)
    {
        if(key==p->data)
        {
            if(p!=first)
            {
                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);

}

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;

    }
```

```
}

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 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 isSorted(struct Node *p)
{
    int x=-65536;

    while(p!=NULL)
    {
        if(p->data < x)
            return 0;
        x=p->data;
        p=p->next;
    }
    return 1;

}
```

```
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;
        }
    }

}

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;
}

void Concat(struct Node *p,struct Node *q)
{
    third=p;

    while(p->next!=NULL)
        p=p->next;
    p->next=q;

}
```

```
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;

}
```

```c
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;
}
```