Linked Lists

Advantages of Linked Lists over Arrays

1. **Efficient Memory Utilization:** The memory of a linked list is not pre-allocated. Memory is allocated whenever required and de-allocated when it is no longer required.
2. Insertion and deletion operations are easier and efficient.
3. **Extensive manipulations:** Without any prior idea of memory space available we can perform the computations. Example, there is no "overflow condition" while implementing stacks or queues.
4. Data can be stored in non-contiguous blocks of memory which exhibits logical adjacency i.e. they are linked by pointers.

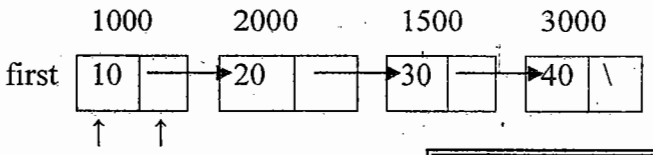
Disadvantages of Linked Lists

1. It requires extra space because each node requires to hold the address of the next node within it (i.e. a pointer field).
2. Accessing a particular node in the list may take more time compared with arrays because the list needs to be traversed from the beginning.

Singly Linked Lists and Chains

Linked list is a linear collection of data elements called nodes and there exists a logical relationship between nodes (i.e. given the address of first node, any node in that list can be obtained)

The pictorial representation of a singly linked list is as shown below.



Info field Link field

Each and every node has 2 fields.

- ✓ **Info field** – Here some useful information can be stored.
- ✓ **Link field** – This field contains address of the next node. So this field should be of type pointer.

Note: A chain is a singly linked list that is compromised of zero or more nodes. When the number of nodes is zero, the chain is empty. The nodes of a chain are ordered so that the first node links to the second, second to the third and so on. The last node of a chain has a zero link (NULL).

Representing Chains in C

The following capabilities are needed to make linked representation possible

- ✓ A mechanism to define the nodes structure. This is done using self-referential structures.
- ✓ A way to create new nodes when we need them. This is done using malloc( ) function.
- ✓ A way to remove nodes that we no longer need. This is done using free( ) function.

The following is the structure definition for each in the list.

```
struct node
{
        int info;
        struct node *link;
};
```


In the above structure definition the type of field or member (link) is same as the structure name and therefore known as self referential structure.

Note:

- ✓ The variable 'first' contains the address of first node (initially NULL)
- ✓ All functions require first
- ✓ Functions that manipulate the linked list should return the address of the first node.
- ✓ first = NULL => List is empty.
- ✓ first ->link = NULL => There is one element in the List
- ✓ first ->link ! = NULL => There is more than one element in the List

Fundamental Operations of a Linked List

In the following functions we use the structure definition as shown below.

```
struct node
{
        int info;
        struct node *link;
};
typedef struct node * NODE;
```

Function to create a 2-node list

```
NODE create2( )
{
        NODE first, second;
        first = (NODE)malloc(sizeof(struct node));
        second = (NODE)malloc(sizeof(struct node));

        first->info = 10;
        first->link = second;

        second->info = 20;
        second->link = NULL;

        return first;
}
```


Function to insert an element at the front end

```
NODE insert_front(int item, NODE first)
{
        NODE temp;
        temp = (NODE)malloc(sizeof(struct node));
        temp->info = item;
        temp->link = first;
        return temp;
}
```

Function to delete an element at the front end

```
NODE delete_front(NODE first)
{
        NODE temp;

        if(first = = NULL)   // no node
        {
                printf("list is empty\n");
                return first;
        }
```

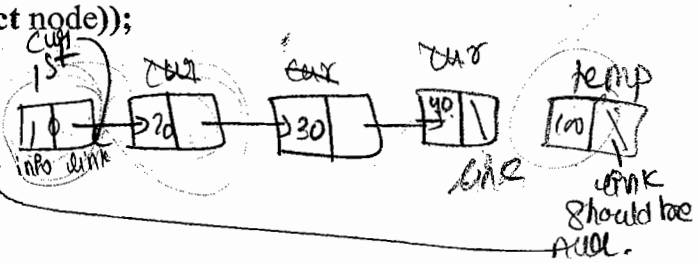
```
        temp = first;
        first = first->link;
        printf("the deleted item is %d", temp->info);
        free(temp);
        return(first);
}
```

address of 20 is stored in 1st link.

physically delete memory.

Function to insert an element at the rear end

```
NODE insert_rear(int item,NODE first)
{
        NODE temp, cur;
        temp=(NODE)malloc(sizeof(struct node));
        temp->info = item;
        temp->link = NULL;
        if(first = = NULL)   //no node
        {
                return temp;
        }
        cur = first;              // node exists
        while(cur->link != NULL)
        {
                cur = cur->link;
        }
        cur->link = temp;
        return first;
}
```

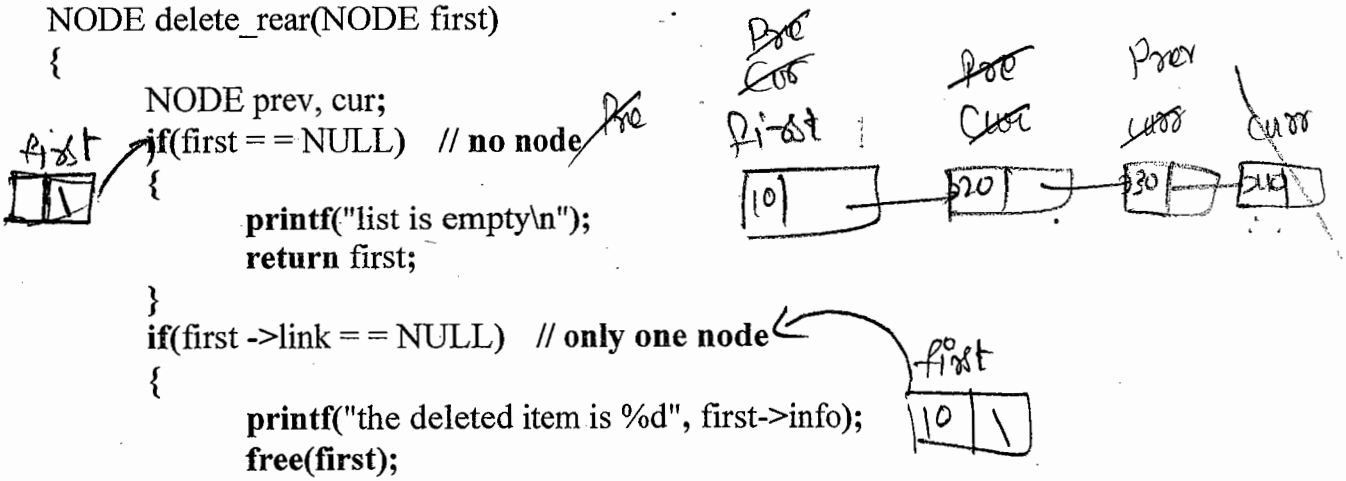


current node = cur

Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

Function to delete an element at the rear end

```
NODE delete_rear(NODE first)
{
        NODE prev, cur;
        if(first = = NULL)   // no node
        {
                printf("list is empty\n");
                return first;
        }
        if(first ->link = = NULL)   // only one node
        {
                printf("the deleted item is %d", first->info);
                free(first);
                return( NULL );
        }
        prev = NULL;          // more than one node
        cur = first;
```



```
        while(cur->link != NULL)
        {
                prev = cur;
                cur = cur->link;
        }
        printf("the deleted item is %d", cur->info);
        free(cur);
        prev->link = NULL;
        return(first);
}
```

Function to display the contents of the list

```
void display(NODE first)
{
        NODE cur;
        if(first == NULL)
        {
                printf("the list is empty\n");
                return;
        }
        cur = first;
        printf("the contents of the list are:\n");
        while(cur != NULL)
        {
                printf("%d\n",cur->info);
                cur = cur->link;
        }
}
```

Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

Note:

1. To implement stacks using linked lists the functions required are
   - ✓ Insert_front( ), Delete_front and Display( ) OR
   - ✓ Insert_rear( ), Delete_rear and Display( )

2. To implement queues using linked lists the function required are
   - ✓ Insert_front( ), Delete_rear and Display( ) OR
   - ✓ Insert_rear( ), Delete_front and Display( )

3. To implement De-queues using linked lists the function required are
   - ✓ Insert_front( ), Delete_front, Insert_rear( ), Delete_rear and Display( )

// Main function for Stacks using Linked Lists

```
void main( )
{
        int choice, item;
        NODE first = NULL;
        clrscr( );
```

```
        for(;;)
        {
                printf("\n1:push\n2:pop\n3:display\n4:exit\n");
                printf("enter your choice");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1: printf("enter the item to be pushed\n");
                                scanf("%d",&item);
                                first = insert_front(item,first);
                                break;
                        case 2: first = delete_front(first);
                                break;
                        case 3: display(first);
                                break;
                        default : exit(0);
                }
                getch( );
        }
}
// Main function for queues using linked lists
void main( )
{
        int choice, item;
        NODE first = NULL;
        clrscr( );
        for(;;)
        {
                printf("\n1:insert\n2:delete\n3:display\n4:exit\n");
                printf("enter your choice");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1: printf("enter the item to be pushed\n");
                                scanf("%d",&item);
                                first = insert_rear(item, first);
                                break;
                        case 2: first = delete_front(first);
                                break;
                        case 3: display(first);
                                break;
                        default : exit(0);
                }
                getch( ):
        }
}
```

Dr.Mahesh G	Dr.Harish G
Assoc. Prof.	Assoc. Prof.
BMSIT & M	Dr. AIT

Lab program

Design, Develop and Implement a menu driven Program in C for the following operations on **Singly Linked List (SLL)** of Student Data with the fields: ***USN, Name, Branch, Sem, PhNo***

a. Create a **SLL** of **N** Students Data by using ***front insertion***.
b. Display the status of **SLL** and count the number of nodes in it
c. Perform Insertion / Deletion at End of **SLL**
d. Perform Insertion / Deletion at Front of **SLL(Demonstration of stack)**
e. Exit

```
#include<stdio.h>

struct node
{
        char usn[10];
        char name[20];
        char branch[20];
        int sem;
        char phno[20];

        struct node *link;
};
typedef struct node * NODE;
```


Function to insert at the front end

```
NODE insert_front(char usn[], char name[], char branch[], int sem, char phno[], NODE first)
{
        NODE temp;
        temp = (NODE)malloc(sizeof(struct node));
        strcpy(temp->usn, usn);
        strcpy(temp->name, name);
        strcpy(temp->branch, branch);
        temp->sem = sem;
        strcpy(temp->phno, phno);

        temp->link = first;
        return temp;
}
```

Function to delete at the front end

```
NODE delete_front(NODE first)
{
        NODE temp;

        if(first = = NULL)    // no node
        {
                printf("list is empty\n");
                return first;
        }
```

```
        temp = first;
        first = first->link;

        printf("Student with following details is deleted\n");
        printf("usn : %s\n", temp->usn);
        printf("name : %s\n", temp->name);
        printf("branch : %s\n", temp->branch);
        printf("sem : %d\n", temp->sem);
        printf("phone no : %s\n", temp->phno);

        free(temp);
        return(first);
}
```

Function to insert at the rear end

```
NODE insert_rear(char usn[], char name[], char branch[], int sem, char phno[], NODE first)
{
        NODE temp, cur;
        temp=(NODE)malloc(sizeof(struct node));

        strcpy(temp->usn, usn);
        strcpy(temp->name, name);
        strcpy(temp->branch, branch);
        temp->sem = sem;
        strcpy(temp->phno, phno);

        temp->link = NULL;

        if(first = = NULL)    //no node
        {
                return temp;
        }
        cur = first;                // node exists
        while(cur->link != NULL)
        {
                cur = cur->link;
        }
        cur->link = temp;
        return first;
}
```

Function to delete at the rear end

```
NODE delete_rear(NODE first)
{
        NODE prev, cur;
        if(first = = NULL)    // no node
        {
                printf("list is empty\n");
                return first;
        }
        if(first ->link = = NULL)    // only one node
        {

                printf("Student with following details is deleted\n");
                printf("usn : %s\n", first->usn);
                printf("name : %s\n", first ->name);
                printf("branch : %s\n", first ->branch);
                printf("sem : %d\n", first ->sem);
                printf("phone no : %s\n", first ->phno);

                free(first);
                return( NULL );
        }
        prev = NULL;          // more than one node
        cur = first;
        while(cur->link != NULL)
        {
                prev = cur;
                cur = cur->link;
        }

        printf("Student with following details is deleted\n");
        printf("usn : %s\n", cur->usn);
        printf("name : %s\n", cur ->name);
        printf("branch : %s\n", cur ->branch);
        printf("sem : %d\n", cur ->sem);
        printf("phone no : %s\n", cur ->phno);

        free(cur);
        prev->link = NULL;
        return(first);
}
```

Function to display the contents of the list and count the number of nodes

```
void display(NODE first)
{
        NODE cur;
        int count = 0;
        if(first = = NULL)
        {
                printf("the list is empty\n");
                printf("the number of nodes in the list is %d\n", count);
                return;
        }
        cur = first;
        printf("the contents of the list are:\n");
        printf("usn\t name\t branch\t sem\t phoneno\n");

        while(cur != NULL)
        {
                count = count + 1;

                printf("usn : %s\t", cur->usn);
                printf("name : %s\t", cur ->name);
                printf("branch : %s\t", cur ->branch);
                printf("sem : %d\t", cur ->sem);
                printf("phone no : %s\n", cur ->phno);

                cur = cur->link;
        }
        printf("the number of nodes in the list is %d\n", count);
}
```

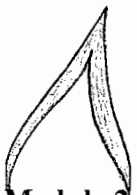


```
void main( )
{
        int choice, sem, n,i;
        char usn[20], name[20], branch[20], phno[20];
        NODE first = NULL;
        clrscr( );

        // Creating a list of 'n' students using front insertion

        printf("Enter the number of students");
        scanf("%d", &n);
        for(i=0; i<n; i++)
        {
                printf("Enter usn\n");
                gets(usn);
```

```
            printf("Enter name\n");
            gets(name);
            printf("Enter branch\n");
            gets(branch);
            printf("Enter sem\n");
            scanf("%d", &sem);
            printf("Enter phone no\n");
            gets(phno);

             first = insert_front( usn, name, branch, sem, phno, first);
    }

    for(;;)
    {
            printf("1:Insert front\n2:Insert rear\n");
            printf("3:Delete front\n4:Delete rear\n");
            printf("5:display\n6:exit\n");

            printf("enter your choice");
            scanf("%d",&choice);

            switch(choice)
            {
                    case 1: printf("Enter usn\n");
                            gets(usn);
                            printf("Enter name\n");
                            gets(name);
                            printf("Enter branch\n");
                            gets(branch);
                            printf("Enter sem\n");
                            scanf("%d", &sem);
                            printf("Enter phone no\n");
                            gets(phno);

                             first = insert_front( usn, name, branch, sem, phno, first);
                             break;

                    case 2: printf("Enter usn\n");
                            gets(usn);
                            printf("Enter name\n");
                            gets(name);
                            printf("Enter branch\n");
                            gets(branch);
                            printf("Enter sem\n");
                            scanf("%d", &sem);
                            printf("Enter phone no\n");
```

Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

```
                    gets(phno);
                    first = insert_rear( usn, name, branch, sem, phno, first);
                    break;

                case 3: first = delete_front(first);
                    break;

                case 4: first = delete_rear(first);
                    break;

                case 5: display(first);
                    break;

                default : exit(0);
            }
            getch( );
        }
}
```

22/10/18

Linked Stacks and Queues

Implementation of multiple stacks using linked lists

```
#include<stdio.h>

struct node
{
        int info;
        struct node *link;
};
typedef struct node * NODE;

// Function to insert an element at the front end

// Function to delete an element from the front end

// Function to display the contents of the list

// Main function for Multiple Stacks using Linked Lists
void main( )
{
        int choice, item, i;
        NODE first[100];

        int n;                  // number of stacks;

        printf("enter the number of stacks\n");
        scanf("%d", &n);

        for(i=0; i<n; i++)
        {
                first[i] = NULL;
        }

        for(;;)
        {
                printf("\n1:push\n2:pop\n3:display\n4:exit\n");
                printf("enter your choice");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1: printf("enter the stack number to push\n");
                                scanf("%d",&i);
                                printf("enter the item to be pushed\n");
                                scanf("%d",&item);
                                first[i] = insert_front(item,first[i]);
                                break;
```

```
                case 2: printf("enter the stack number to pop\n");
                        scanf("%d",&i);
                        first[i] = delete_front(first[i]);
                        break;

                case 3: printf("enter the stack number to display\n");
                        scanf("%d",&i);
                        display(first[i]);
                        break;

                default : exit(0);
            }
        }
}
```

Implementation of multiple queues using linked lists

```
#include<stdio.h>

struct node
{
        int info;
        struct node *link;
};
typedef struct node * NODE;
```



```
// Function to insert an element at the rear end

// Function to delete an element from the front end

// Function to display the contents of the list

// Main function for Multiple Queues using Linked Lists
void main( )
{
        int choice, item, i;
        NODE first[100];

        int n;                  // number of queues;
        printf("enter the number of queues\n");
        scanf("%d", &n);

        for(i=0; i<n; i++)
        {
                first[i] = NULL;
        }
```

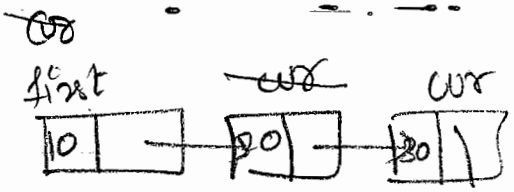
```
for(;;)
{
    printf("\n1:insert \n2:delete \n3:display\n4:exit\n");
    printf("enter your choice");
    scanf("%d",&choice);
    switch(choice)
    {
        case 1: printf("enter the queue number to insert\n");
                scanf("%d",&i);
                printf("enter the item to be inserted\n");
                scanf("%d",&item);
                first[i] = insert_rear(item,first[i]);
                break;

        case 2: printf("enter the queue number to delete\n");
                scanf("%d",&i);
                first[i] = delete_front(first[i]);
                break;

        case 3: printf("enter the queue number to display\n");
                scanf("%d",&i);
                display(first[i]);
                break;

        default : exit(0);
    }
  }
}
```

Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

Other operations of a linked list

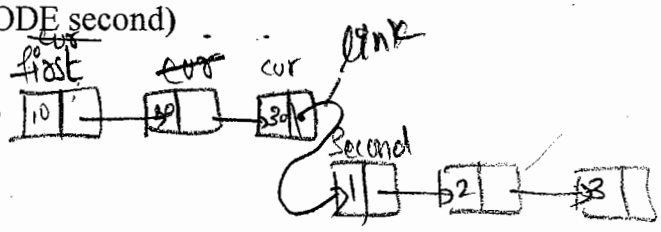
Function to count the number of nodes in a list

```
int count_nodes(NODE first)
{
        int count = 0;
        NODE cur;
        cur = first;
        while(cur != NULL)
        {
                count = count + 1;
                cur = cur->link;
        }
        return count;
}
```



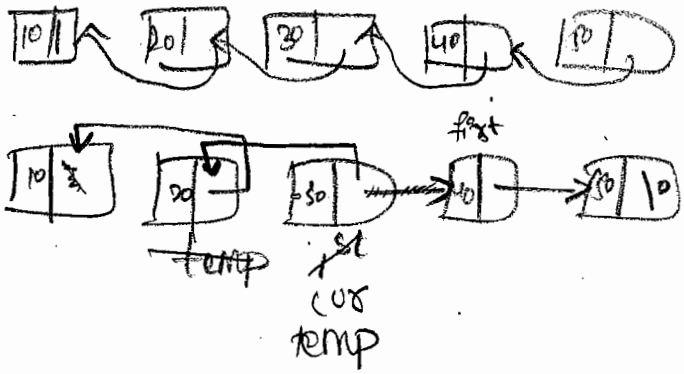
Function to concatenate 2 lists

```
NODE concatenate(NODE first, NODE second)
{
        NODE cur;
        if(first = = NULL)
                return second;
        if(second = = NULL)
                return first;
        cur = first;
        while(cur ->link!= NULL)
        {
                cur = cur->link;
        }
        cur->link = second;
        return first;
}
```



Function to reverse a singly linked list without creating a new node

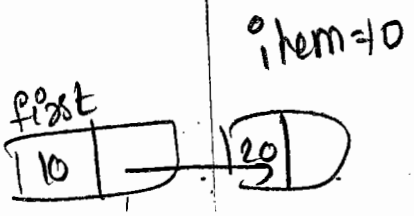
```
NODE reverse(NODE first)
{
        NODE temp, cur;
        if(first = = NULL)
                return first;
        temp = NULL;
        while(first!= NULL)
        {
                cur = first;
                first = first->link;
                cur->link = temp;
                temp = cur;
        }
        return temp;
}
```

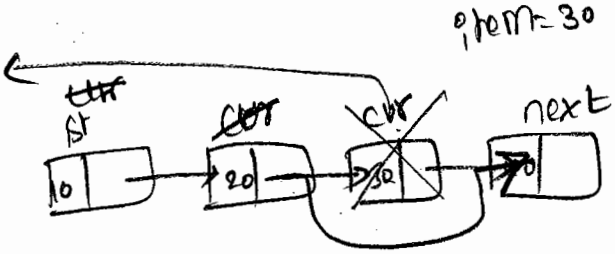


Note: At first or beginning, first contains address of the list to be reversed, later temp contains address of the reversed list.

Function to delete a node whose information field is given

```
NODE delete_info(int item, NODE first)
{
        NODE cur, temp, prev, next;
        if(first = = NULL)              // empty list
        {
                printf("the list is empty\n");
                return first;
        }

        if(item = = first->info)        // if it is the first node
        {
                temp=first;
                first = first->link;
                printf("item deleted  is %d",temp->info);
                free(temp);
                return first;
        }

        prev = NULL;            // other than first node
        cur = first;
        while(cur != NULL && item != cur->info)
        {
                prev = cur;
                cur = cur->link;
        }

        if(cur = = NULL)
        {
                printf("item not found\n");
                return first;
        }

        next = cur->link
        prev->link = next;
        printf("item deleted  is %d",cur->info);
        free(cur);
        return first;
}
```





It possible only if cur==NULL

Function to insert an item at a specified position

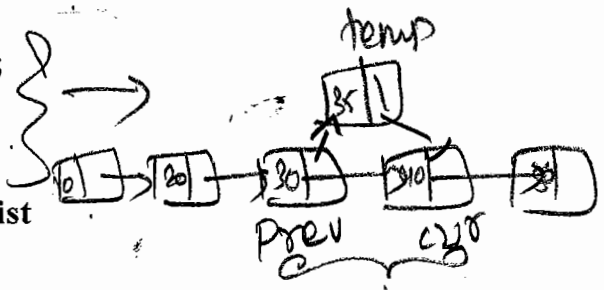
```
NODE insert_pos(int item, int pos, NODE first)
{
        NODE temp, cur, prev;
        int count;

        temp=(NODE)malloc(sizeof(struct node));
        temp->info = item;
        temp->link = NULL;

        if(first = = NULL && pos = = 1)  // no elements in the list
                return temp;

        if(pos = = 1)           // insert at the beginning
        {
                temp->link = first;
                return temp;
        }

        prev = NULL;            // find appropriate position
        cur = first;
        count = 1;
        while(cur != NULL && count != pos)
        {
                prev = cur;
                cur = cur->link;
                count++;
        }

        if(count != pos)
        {
                printf("invalid position\n");
                free(temp);
                return first;
        }

        prev->link = temp;
        temp->link = cur;
        return first;
}
```

Function to insert an element into the list such that after insertion the list remains sorted.

Here pos is not given

```
NODE insert_order(int item, NODE first)
{
        NODE temp, cur, prev;

        temp=(NODE)malloc(sizeof(struct node));
        temp->info = item;
        temp->link = NULL;

        if(first = = NULL)    // no elements in the list
                return temp;

        if(item <= first->info)              // insert at the beginning
        {
                temp->link = first;
                return temp;
        }

        prev = NULL;          // insert at middle or end
        cur = first;

        while(cur != NULL && item > cur->info)
        {
                prev = cur;
                cur = cur->link;
        }

        prev->link = temp;
        temp->link = cur;
        return first;
}
```

item = 35



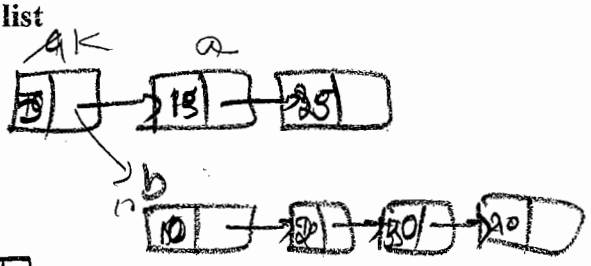
item = 5

5 < 10

(35 > 30)

Function to merge 2 sorted lists into a single sorted list

```
NODE merge(NODE a, NODE b)
{
        NODE k, c, temp;
        c = (NODE) malloc(sizeof(struct node));
        k = c;
        while(a!= NULL && b!= NULL)
        {
                if(a->info < b->info)
                {
                        k->link = a;
                        a = a->link;
                        k = k->link;
```



```
                }
                else
                {
                        k->link = b;
                        b = b->link;
                        k = k->link;

                }
        }
        if(b!=NULL)
                k->link = b;
        else
                k->link = a;

        temp = c;
        c = c->link;
        free(temp);
        return c;
}
```

Function to search for an item in the list and display the position if found

```
void search_item(int item, NODE first)
{
        int pos;

        if(first = = NULL)
        {
                printf("list is empty\n");
                return;
        }

        cur = first;
        pos = 1;

        while(cur != NULL && item != cur->info)
        {
                cur = cur->link;
                pos = pos + 1;
        }

        if(cur = = NULL)
        {
                printf("Unsuccessful search\n");
                return;
        }

        printf("Successful search and item found at position %d\n", pos);
}
```

Function to delete an element at the specified position

```
NODE delete_pos(int pos, NODE first)
{
        NODE cur, prev, next, temp;

        if(first = = NULL)              // empty list
        {
                printf("the list is empty\n");
                return first;
        }

        if(pos = = 1)  // if it is the first position
        {
                temp=first;
                first = first->link;
                printf("item deleted is %d",temp->info);
                free(temp);
                return first;
        }

        prev = NULL;          // other than first position
        cur = first;
        count = 1;

        while(cur != NULL && count != pos)
        {
                prev = cur;
                cur = cur->link;
                count = count +1;
        }

        if(cur = = NULL)
        {
                printf("invalid position\n");
                return first;
        }

        next = cur->link
        prev->link = next;

        printf("item deleted is %d",cur->info);

        free(cur);
        return first;
}
```

Function to delete all the nodes whose info field is same as the item specified

```
NODE delete_all_specified_items(int item, NODE first)
{
        int flag;
        NODE prev, cur;

        flag = 0;
        prev = NULL;
        cur = first;

        while( cur != NULL )
        {
                if(item = = cur->info)
                {
                        flag = flag + 1;
                        if(prev != NULL)    // other than first
                        {
                                prev->link = cur->link;
                                free(cur);
                                cur = prev->link;            // update cur
                        }
                        else    // if it is the first node
                        {
                                first = first->link;
                                free(cur);
                                cur = first;
                        }
                }
                else
                {
                        prev = cur;
                        cur = cur->link;
                }
        }

        if(flag = = 0)
                printf("item not found\n");
        else
                printf("%d number of nodes are deleted\n", flag);

        return first;
}
```

Write a 'C' function to remove DUPLICATE elements in a singly linked list.

```
NODE remove_duplicate(NODE first)
{

        if(first = = NULL)          // empty list
        {
                printf("the list is empty\n");
                return first;
        }

        temp = first;
        while(temp != NULL)
        {
                item = temp->info;

                prev = temp;
                cur = temp->link;

                while( cur != NULL )
                {
                        if(item = = cur->info)
                        {
                                prev->link = cur->link;
                                free(cur);
                                cur = prev->link;          // update cur
                        }
                        else
                        {
                                prev = cur;
                                cur = cur->link;
                        }
                }
                temp = temp->link;
        }
        return first;
}
```

Write a 'C' function to find UNION of two singly linked lists.
Assumption: No duplicate elements in the list

```
int search(int item, NODE first)  // return 1 if item found else return 0
{
        if(first = = NULL)
        {
                return 0;
        }

        cur = first;

        while(cur != NULL && item != cur->info)
        {
                cur = cur->link;
        }

        if(cur = = NULL)
        {
                return 0;
        }
        return 1;
}
```



```
NODE union(NODE a , NODE b)
{
        NODE c=NULL, cur;
        int flag;
        if(a = = NULL)
                return b;
        if(b = = NULL)
                return a;

        cur = a;
        while(cur != NULL)
        {
                c = insert_rear(cur->info, c);
                cur = cur->link;
        }

        cur = b;
        while(cur != NULL)
        {
                flag = search(cur->info, a);
                if(flag == 0)
                        c = insert_rear(cur->info, c);
                cur = cur->link;
        }
        return c;
}
```

Write a 'C' function to find INTERSECTION of two singly linked lists.
Assumption: No duplicate elements in the list

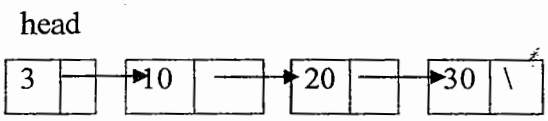
```
int search(int item, NODE first)  // return 1 if item found else return 0
{
        if(first = = NULL)
        {
                return 0;
        }

        cur = first;

        while(cur != NULL && item != cur->info)
        {
                cur = cur->link;
        }

        if(cur = = NULL)
        {
                return 0;
        }

        return 1;

}

NODE intersection(NODE a , NODE b)
{
        NODE c=NULL, cur;
        int flag;
        if(a = = NULL) .
                return NULL;
        if(b = = NULL) '
                return NULL;

        cur = a;
        while(cur != NULL)
        {
                flag = search(cur->info, b);
                if(flag == 1)
                        c = insert_rear(cur->info, c);
                cur = cur->link;
        }
        return c;
}
```

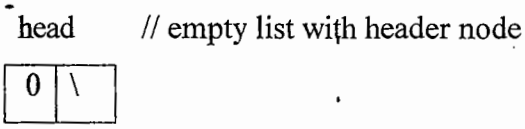
Header Nodes

Some times it is desirable to keep an extra node at the front of a list which simplifies the deign process of some list operations. Such a node does not represent an item in the list and is called a header node or a list header.

The info field of such a header node generally contains the global information of the entire list, like the number of nodes in the list.

Ex:





In both the cases the info field of head contains the total number of nodes in the list. Each time a node is added or deleted, the count in this field (i.e info field of the header) must be readjusted so as to contain actual number of nodes currently present which is certainly a overhead.

Note:

- ✓ If the list is empty, link field of the header node points to NULL. Otherwise, link field of the header node contains the address of the first node of the list and the link field of the last node contains NULL.

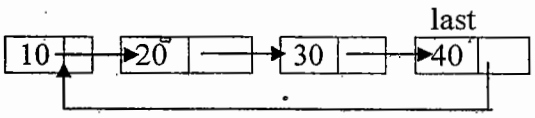
Given a list with header node, any node in the list can be accessed without the need for a pointer variable (first, which points to the first node as in before examples).

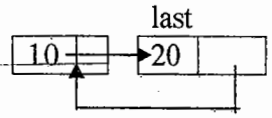
Circular Singly Linked List

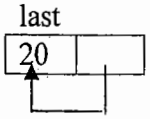
In normal singly linked list,

- ✓ The link part of the last node has NULL value, specifying that it is the last node.
- ✓ Given the address of any node 'x', it is only possible to reach the nodes which follow 'x' and it is not possible to reach the nodes that precedes 'x' (previous nodes). To reach the nodes that precedes 'x', it is required to preserve a pointer variable first, which contains the address of the first node in the list and use this for traversing.
- ✓ Also, given the address of the first node to get the address of the last node, the list has to be traversed from the beginning till the last node is reached.

These disadvantages can be overcome using circular singly linked list. In a circular singly linked list the link field of the last node points to the first node in the list, which makes the list a circular one. The pictorial representation is as shown below.







Note:

- ✓ Here the link field of the last node contains address of the first node.
- ✓ In the following functions we use the structure definition as shown below.
- ✓

```
struct node
{
        int info;
        struct node *link;
};
typedef struct node * NODE;
```

Function to insert an element at the rear end

```
NODE insert_rear(int item, NODE last)
{
        NODE temp;
        temp=(NODE)malloc(sizeof(struct node));
        temp->info = item;
        temp->link = temp;
        if(last = = NULL)    //no node
        {
                return temp;
        }
        temp->link = last->link;
        last->link = temp;
        return temp;
}
```


Function to insert an element at the front end

```
NODE insert_front(int item, NODE last)
{
        NODE temp;
        temp=(NODE)malloc(sizeof(struct node));
        temp->info = item;
        temp->link = temp;
        if(last = = NULL)    //no node
        {
                return temp;
        }
        temp->link = last->link;
        last->link = temp;
        return last;
}
```

Function to delete an element at the front end

```
NODE delete_front(NODE last)
{
        NODE first;
        if(last = = NULL)    // no node
        {
                printf("list is empty\n");
                return last;
        }
        if(last ->link = = last)        // only one node
        {
                printf("the deleted item is %d", last->info);
                free(last);
                return( NULL );
        }
```

```
        first = last->link;      // more than one node
        last->link = first->link;
        printf("the deleted item is %d", first->info);
        free(first);
        return(last);
}
```

Function to delete an element at the rear end

```
NODE delete_rear(NODE last)
{
        NODE cur;
        if(last = = NULL)    // no node
        {
                printf("list is empty\n");
                return last;
        }
        if(last ->link = = last)          // only one node
        {
                printf("the deleted item is %d", last->info);
                free(last);
                return( NULL );
        }
        cur = last->link;       // more than one node
        while(cur->link != last)
        {
                cur = cur->link;
        }
        cur->link = last->link;
        printf("the deleted item is %d", last->info);
        free(last);
        return(cur);
}
```


Function to display the contents of the list

```
void display(NODE last)
{
        NODE cur;
        if(last = = NULL)
        {
                printf("the list is empty\n");
                return;
        }
        cur = last->link;
```

```
        printf("the contents of the list are:\n");

        while(cur != last)
        {
                printf("%d\n",cur->info);
                cur = cur->link;
        }
        printf("%d\n",last->info);
}

// Main function for circular singly linked lists
void main( )
{
        int choice, item;
        NODE last = NULL;
        clrscr( );
        for(;;)
        {
                printf("\n1:insert-front\n 2:insert-rear\n ");
                printf("\n3:delete-front\n 4:delete-rear\n ");
                printf("\n5:display\n 6:exit\n ");
                printf("enter your choice");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1: printf("enter the item to be inserted\n");
                                scanf("%d",&item);
                                last = insert_front(item, last);
                                break;
                        case 2: printf("enter the item to be inserted\n");
                                scanf("%d",&item);
                                last = insert_rear(item, last);
                                break;
                        case 3: last = delete_front(last);
                                break;
                        case 4: last = delete_rear(last);
                                break;
                        case 5: display(last);
                                break;
                        default : exit(0);
                }
                getch( ):
        }
}
```

Function to find the length of a circular list

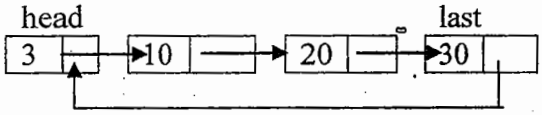
```
void length(NODE last)
{
        NODE cur;
        int count;
        if(last = = NULL)
        {
                printf("the length of the list is 0 \n");
                return;
        }
        cur = last->link;
        count = 1;
        while(cur != last)
        {
                count = count + 1;
                cur = cur->link;
        }
        printf("the length of the list is %d \n", count);
}
```

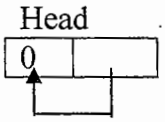
Function to search for an item in the list. The function should return a pointer to the node containing the item if found and NULL otherwise

```
NODE search_item(int item, NODE last)
{
        NODE cur;
        if(last = = NULL)
        {
                printf("list is empty\n");
                return NULL;
        }
        if(item = = last->info)
        {
                printf("Successful search\n");
                return last;
        }
        cur = last->link;
        while(cur != last && item != cur->info)
        {
                cur = cur->link;
        }
        if(cur = = last)
        {
                printf("Unsuccessful search\n");
                return NULL;
        }
        printf("Successful search \n");
        return cur;
}
```

Circular Singly Linked List with header node

Here if the list is empty, link of head contains head, otherwise link field of header node contains address of the first node. The link field of last node contains address of header node. The pictorial representation of this is as shown below.





Note: In the following functions we use the structure definition as shown below.

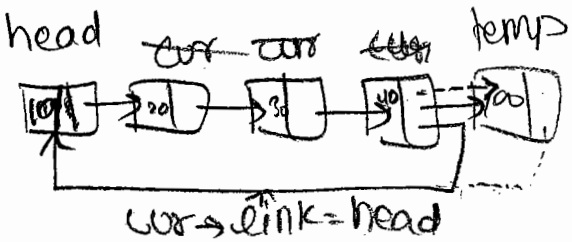
```
struct node
{
        int info;
        struct node *link;
};
typedef struct node * NODE;
```


Function to insert an element at the front end

```
NODE insert_front(int item, NODE head)
{
        NODE temp;
        temp=(NODE)malloc(sizeof(struct node));
        temp->info = item;
        temp->link = head->link;
        head->link = temp;
        head->info = head->info +1;
        return head;
}
```

Function to insert an element at the rear end

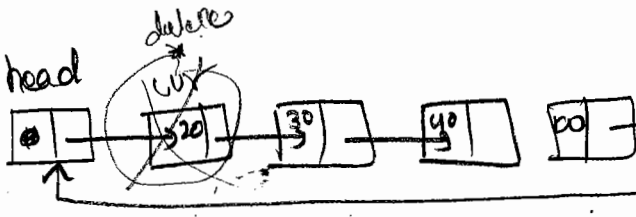
```
NODE insert_rear(int item, NODE head)
{
        NODE temp, cur;
        temp=(NODE)malloc(sizeof(struct node));
        temp->info = item;
        cur = head->link;
        while(cur->link != head)
        {
                cur = cur->link;
        }
        cur->link = temp;
        temp->link = head;
        head->info = head->info + 1;
        return head;
}
```



In insert = head→info + 1
In delete = " " " − 1

Function to delete an element at the front end

```
NODE delete_front(NODE head)
{
        NODE cur;
        if(head->link = = head)      // no node
        {
                printf("list is empty\n");
                return head;
        }
        cur = head->link;
        head->link = cur->link;
        printf("the deleted item is %d", cur->info);
        free(cur);
        head->info = head->info - 1;
        return(head);
}
```

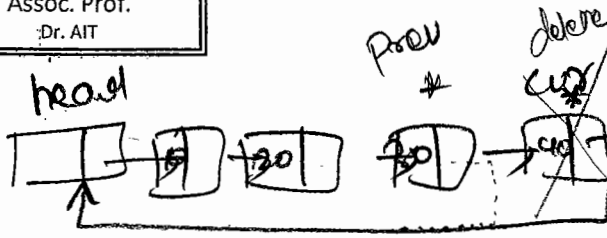


link b/w head & 30 node
cur (20) deleted
physically deleted

Function to delete an element at the rear end

```
NODE delete_rear(NODE head)
{
        NODE prev, cur;
        if(head->link = = head)      // no node
        {
                printf("list is empty\n");
                return head;
        }
        prev = head;
        cur = head->link;
        while(cur->link != head)
        {
                prev = cur;
                cur = cur->link;
        }
        prev->link = head;
        printf("the deleted item is %d", cur->info);
        free(cur);
        head->info = head->info - 1;
        return(head);
}
```

same



Function to display the contents of the list

```
void display(NODE head)
{
        NODE cur;
        if(head->link = = head)
        {
                printf("the list is empty\n");
```

```
            return;
        }
        cur = head->link;
        printf("the contents of the list are:\n");
        while(cur != head)
        {
                printf("%d\n",cur->info);
                cur = cur->link;
        }
}

// Main function for circular singly linked lists with header node
void main( )
{
        int choice, item;    NODE head;
        head = (NODE) malloc(sizeof(struct node));
        head->info = 0;
        head->link = head;
        clrscr( );
        for(;;)
        {
                printf("\n1:insert-front\n  2:insert-rear\n ");
                printf("\n3:delete-front\n  4:delete-rear\n ");
                printf("\n5:display\n  6:exit\n ");
                printf("enter your choice");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1: printf("enter the item to be inserted\n");
                                scanf("%d",&item);
                                head = insert_front(item, head);
                                break;
                        case 2: printf("enter the item to be inserted\n");
                                scanf("%d",&item);
                                head = insert_rear(item, head);
                                break;
                        case 3: head = delete_front(head);
                                break;
                        case 4: head = delete_rear(head);
                                break;
                        case 5: display(head);
                                break;
                        default : exit(0);
                }
        }
}
```

Design, Develop and Implement a Program in C for the following operations on **Singly Circular Linked List (SCLL)** with header nodes

a. Represent and Evaluate a Polynomial **P(x,y,z) = 6x2y2z-4yz5+3x3yz+2xy5z-2xyz3**

b. Find the sum of two polynomials **POLY1(x,y,z)** and **POLY2(x,y,z)** and store the result in **POLYSUM(x,y,z)**

Support the program with appropriate functions for each of the above operations

```
#include<stdio.h>

struct node
{
        int cf;
        int px;
        int py;
        int pz;
        struct node *link;
};
typedef struct node* NODE;

// Function to display the polynomial
void display(NODE head)
{
        NODE cur;
        if(head->link = = head)
        {
                printf("Polynomial does not exist\n");
                return;
        }
        cur = head->link;
        while(cur != head)
        {
                if(cur->cf > 0)
                                printf("+");
                printf("%d ",cur->cf);
                printf("x^ %d", cur->px);
                printf("y^ %d", cur->py);
                printf("z^ %d", cur->pz);
                cur = cur->link;
        }
}


// Function to insert an element at the rear end
NODE insert_rear(int cf, int px, int py, int pz, NODE head)
{
        NODE temp, cur;

        temp=(NODE)malloc(sizeof(struct node));
```

Dr.Mahesh G	Dr.Harish G
Assoc. Prof. BMSIT & M	Assoc. Prof. Dr. AIT

```
        temp->cf = cf;
        temp->px = px;
        temp->py = py;
        temp->pz = pz;
        temp->link = NULL;

        cur = head->link;
        while(cur->link != head)
        {
                cur = cur->link;
        }
        cur->link = temp;
        temp->link = head;
        return head;

}
```

Dr.Mahesh G Dr.Harish G
Assoc. Prof. BMSIT & M
Assoc. Prof. Dr. AIT

// Function to read a polynomial

Assumption: The terms in P(x,y,z) will be lexicographically ordered i.e. first we order the terms according to decreasing degrees in x; those with the same degree in x we order according to decreasing degrees in y; those with the same degrees in x and y we order according to decreasing degrees in z.

Example: The polynomial P(x,y,z) = $8x^2y^2z - 6yz^8 + 3x^3yz + 2xy^7z - 5x^2y^3 - 4xy^7z^3$ needs to be read in the order P(x,y,z) = $3x^3yz - 5x^2y^3 + 8x^2y^2z - 4xy^7z^3 + 2xy^7z - 6yz^8$

```
NODE readpoly(NODE head)
{
        int n, i, cf, px;
        printf("Enter the number of terms \n");
        scanf("%d", &n);
        for(i = 1; i<=n; i++)
        {
                printf("Enter the coefficient of %d term\n",i);
                scanf("%d",&cf);

                printf("Enter the power of x of %d term\n",i);
                scanf("%d", &px);
                printf("Enter the power of y of %d term\n",i);
                scanf("%d", &py);
                printf("Enter the power of z of %d term\n",i);
                scanf("%d", &pz);

                head = insert_rear(cf, px, py, pz, head);
        }
        return head;
}
```

```
// Function to evaluate a polynomial
void evaluate(NODE head)
{
        NODE cur;
        int x,y,z,result=0;
        if(head->link = = head)
        {
                printf("Polynomial does not exist\n");
                return;
        }

        printf("Enter the value of x\n");
        scanf("%d",&x);
        printf("Enter the value of y\n");
        scanf("%d",&y);
        printf("Enter the value of z\n");
        scanf("%d",&z);

        cur = head->link;
        while(cur != head)
        {
                result = result + cur->cf * pow(x, cur->px)* pow(y, cur->py)* pow(z, cur->pz);
                cur = cur->link;
        }

        printf("The result of evaluation is %d\n", result);
}
```



```
// Function to add 2 polynomials
NODE addpoly(NODE h1, NODE h2, NODE h3)
{
        NODE p1, p2;
        p1 = h1->link;
        p2 = h2->link;
        int sumcf;
        while(p1 != h1 && p2 != h2)
        {

                if(p1->px > p2->px) // Compare power of x
                {
                        h3 = insert_rear(p1->cf, p1->px, p1->py, p1->pz, h3);
                        p1 = p1->link;
                }
                else if(p1->px < p2->px)
                {
                        h3 = insert_rear(p2->cf, p2->px, p2->py, p2->pz, h3);
                        p2 = p2->link;
                }
```

```
        else
        {
                if(p1->py > p2->py)             // Compare power of y
                {
                        h3 = insert_rear(p1->cf, p1->px, p1->py, p1->pz, h3);
                        p1 = p1->link;
                }
                else if(p1->py < p2->py)
                {
                        h3 = insert_rear(p2->cf, p2->px, p2->py, p2->pz, h3);
                        p2 = p2->link;
                }
                else
                {
                        if(p1->pz > p2->pz)             // Compare power of z
                        {
                                h3 = insert_rear(p1->cf, p1->px, p1->py, p1->pz, h3);
                                p1 = p1->link;
                        }
                        else if(p1->pz < p2->pz)
                        {
                                h3 = insert_rear(p2->cf, p2->px, p2->py, p2->pz, h3);
                                p2 = p2->link;
                        }
                        else    // All powers are equal
                        {
                                sumcf = p1->cf + p2->cf;
                                if(sumcf != 0)
                                        h3 = insert_rear(sumcf, p1->px, p1->py, p1->pz, h3);
                                p1 = p1->link;
                                p2 = p2->link;

                        }
                }
        }
}

// Add remaining terms of Polynomial 1
while(p1 != h1)
{
        h3 = insert_rear(p1->cf, p1->px, h3);
        p1 = p1->link;
}

// Add remaining terms of Polynomial 2
while(p2 != h2)
{
        h3 = insert_rear(p2->cf, p2->px, h3);
        p2 = p2->link;
}
return h3;
}
```

```
void main( )
{
        NODE h1, h2, h3;

        h1 = (NODE)malloc(sizeof(struct node));
        h2 = (NODE)malloc(sizeof(struct node));
        h3 = (NODE)malloc(sizeof(struct node));

        h1->link = h1;
        h2->link = h2;
        h3->link = h3;

        printf("enter the first polynomial\n");
        h1 = readpoly(h1);

        printf("enter the second polynomial\n");
        h2 = readpoly(h2);

        h3 = addpoly(h1, h2, h3)

        printf("The first polynomial is\n");
        display(h1);
        evaluate(h1);
        printf("The second polynomial is\n");
        display(h2);
        evaluate(h2);
        printf("Their sum  is\n");
        display(h3);
}       evaluate(h3);
```

Doubly Linked List

Disadvantages of Singly Linked List

- ✓ Given the address of a node in the list, it is difficult to find the address of previous node.
- ✓ Traversing of list is possible only in the forward direction and hence singly linked list is also termed as one-way list.

Doubly Linked List Representation

To increase the performance and efficiency of algorithms, it is required to traverse the list in either forward or backward direction. Therefore a two-way list called as doubly linked list can be made use of, so that traversing from left to right (forward) or right to left (backward) is possible.

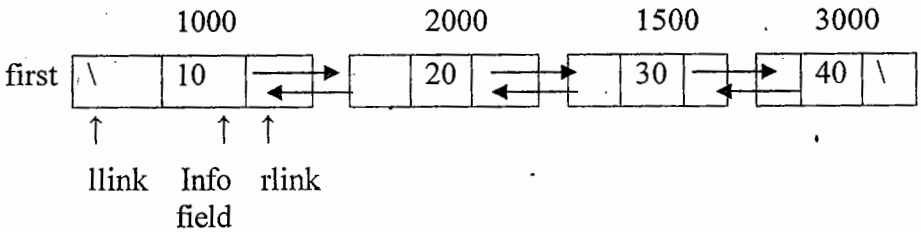
Dr.Mahesh G Dr.Harish G
Assoc. Prof. BMSIT & M
Assoc. Prof. Dr. AIT

Definition

A list where each node has 2 links

- ✓ Left link (llink) – Contains the address of the left node
- ✓ Right link (rlink) – Contains the address of the right node

So that both forward and backward traversal is possible is called doubly linked list.



Each node in the list consists of

- ✓ llink – Contains the address of the left node or previous node
- ✓ info – It is the data to be processed.
- ✓ rlink – Contains the address of the right node or next node

The structure declaration for each node is as shown below

```
struct node
{
        int info;
        struct node *llink;
        struct node *rlink;
};
typedef struct node * NODE;
```

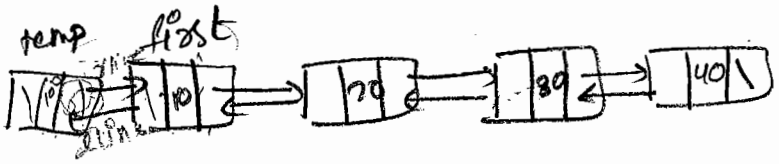
Disadvantages of Doubly Linked List

- ✓ Each node in the list requires an extra link and hence more memory is consumed.
- ✓ If a node is inserted or deleted both llink and rlink should be manipulated

Fundamental Operations of a Doubly Linked List

Function to insert an element at the front end

```
NODE insertfront(int item, NODE first)
{
        NODE temp;
        temp = (NODE) malloc(sizeof(struct node));
        temp->info = item;
        temp->llink = NULL;
        temp->rlink = NULL;
        if(first == NULL)
                return temp;
        temp->rlink = first;
        first->llink = temp;
        return temp;
}
```



Function to insert an element at the rear end

```
NODE insertrear(int item, NODE first)
{
        NODE temp, cur;
        temp = (NODE) malloc(sizeof(struct node));
        temp->info = item;
        temp->llink = NULL;
        temp->rlink = NULL;
        if(first == NULL)
                return temp;
        cur = first;
        while(cur->rlink != NULL)
                cur = cur->rlink;
        cur->rlink = temp;
        temp->llink = cur;
        return first;
}
```

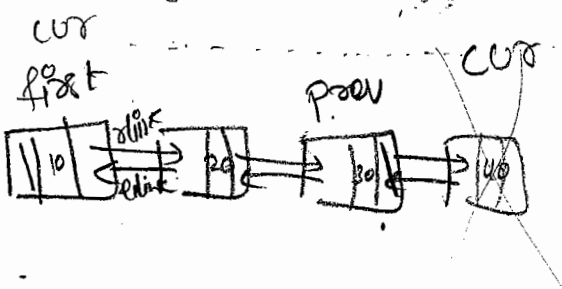
Function to delete an element at the front end

```
NODE deletefront( NODE first)
{
        NODE cur;
        if(first == NULL)
        {
                printf("list is empty\n");
                return first;
        }
        if(first->rlink == NULL)
        {
                printf("item deleted is %d\n", first->info);
                free(first)
                return NULL;
        }
        cur = first;
```

```
        first =first->rlink;
        printf("item deleted is %d\n", cur->info);
        free(cur);
        frist->llink = NULL;
        return first;
}
```

Function to delete an element at the rear end

```
NODE deleterear( NODE first)
{
        NODE prev, cur;
        if(first = = NULL)
        {
                printf("list is empty\n");
                return first;
        }
        if(first->rlink = = NULL)
        {
                printf("item deleted is %d\n", first->info);
                free(first)
                return NULL;
        }
        cur = first;
        while(cur->rlink != NULL)
                cur = cur->rlink;
        prev =cur->llink;
        printf("item deleted is %d\n", cur->info);
        free(cur);
        prev->rlink = NULL;
        return first;
}
```



Function to display the contents of the list

```
void display(NODE first)
{
        NODE cur;
        if(first = = NULL)
        {
                printf("the list is empty\n");
                return;
        }
        printf("the contents of the list are:\n");
        cur = first;
        while(cur != NULL)
        {
                printf("%d\n",cur->info);
                cur = cur->rlink;
        }
}
```

Other Operations of a Doubly Linked List

Function to insert a new node to the left of a node whose key value is read as input

```
NODE insert_left(int key, int item, NODE first)
{
        NODE cur, prev, temp;

        temp=(NODE)malloc(sizeof(struct node));
        temp->info = item;
        temp->llink = temp->rlink = NULL;

        if(first = = NULL)
        {
                printf("the list is empty!cannot insert");
                free(temp);
                return first;
        }

        if(key = = first->info)
        {
                temp->rlink = first;
                first->llink = temp;
                return temp;
        }

        cur = first;
        while(cur != NULL && key != cur->info)
                cur = cur->rlink;

        if(cur = = NULL)
        {
                printf("the node with key value not present!cannot insert\n");
                free(temp);
                return first;
        }

        prev = cur->llink;
        prev->rlink = temp;
        temp->llink = prev;
        temp->rlink = cur;
        cur->llink = temp;

        return first;
}
```

Function to delete a node whose info field is specified

```
NODE delete_info(int item,NODE first)
{
        NODE cur,prev,next;

        if(first = = NULL)
        {
                printf("list is empty\n");
                return first;
        }

        if(item = = first->info)
        {
                cur = first;
                first = first->rlink;

                if(first != NULL)
                        first->llink = NULL;

                printf("item deleted is %d",cur->info);
                free(cur);
                return first;
        }

        cur = first;
        while(cur != NULL && item != cur->info)
                cur = cur->rlink;

        if(cur = = NULL)
        {
                printf("item not found\n");
                return first;
        }
        prev = cur->llink;
        next = cur->rlink;
        prev->rlink = next;

        if(next != NULL)
                next->llink = prev;

        printf("deleted item is %d",cur->info);
        free (cur);
        return first;
}
```

lab prog

Design, Develop and Implement a menu driven Program in C for the following operations on **Doubly Linked List (DLL)** of Employee Data with the fields: ***SSN, Name, Dept, Designation, Sal, PhNo***

a. Create a **DLL** of N Employees Data by using ***end insertion***.
b. Display the status of **DLL** and count the number of nodes in it
c. Perform Insertion and Deletion at End of **DLL**
d. Perform Insertion and Deletion at Front of **DLL**
e. Demonstrate how this **DLL** can be used as **Double Ended Queue**
f. Exit

```
#include<stdio.h>

struct node
{
        char ssn[10];
        char name[20];
        char dept[20];
        char desg[20];
        float sal;
        char phno[20];

        struct node *llink;
};      struct node *llink
typedef struct node * NODE;
```


Function to insert at the front end

```
NODE insert_front(char ssn[], char name[], char dept[], char desg[], float sal, char phno[], NODE firs
{
        NODE temp;
        temp = (NODE) malloc(sizeof(struct node));

        strcpy(temp->ssn, ssn);
        strcpy(temp->name, name);
        strcpy(temp->dept, dept);
        strcpy(temp->desg, desg);
        temp->sal = sal;
        strcpy(temp->phno, phno);

        temp->llink = NULL;
        temp->rlink = NULL;

        if(first == NULL)
                return temp;
        temp->rlink = first;
        first->llink = temp;
        return temp;
}
```

temp -> info = item

Function to insert at the rear end

```
NODE insert_rear(char ssn[], char name[], char dept[], char desg[], float sal, char phno[], NODE first)
{
    NODE temp, cur;
    temp = (NODE) malloc(sizeof(struct node));
    strcpy(temp->ssn, ssn);
    strcpy(temp->name, name);
    strcpy(temp->dept, dept);
    strcpy(temp->desg, desg);
    temp->sal = sal;
    strcpy(temp->phno, phno);

    temp->llink = NULL;
    temp->rlink = NULL;

    if(first == NULL)
        return temp;
    cur = first;
    while(cur->rlink != NULL)
        cur = cur->rlink;
    cur->rlink = temp;
    temp->llink = cur;
    return first;
}
```


Function to delete at the front end

```
NODE delete_front( NODE first)
{
    NODE cur;
    if(first == NULL)
    {
        printf("list is empty\n");
        return first;
    }
    if(first->rlink == NULL)
    {
        printf("Employee with following details is deleted\n");
        printf("ssn : %s\n", first->ssn);
        printf("name : %s\n", first ->name);
        printf("dept : %s\n", first ->dept);
        printf("desg : %s\n", first ->desg);
        printf("sal : %f\n", first ->sal);
        printf("phone no : %s\n", first->phno);

        free(first)
        return NULL;
    }
    cur = first;
```

```
        first =first->rlink;

        printf("Employee with following details is deleted\n");
        printf("ssn : %s\n", cur->ssn);
        printf("name : %s\n", cur->name);
        printf("dept : %s\n", cur->dept);
        printf("desg : %s\n", cur->desg);
        printf("sal : %f\n", cur->sal);
        printf("phone no : %s\n", cur->phno);

        free(cur);
        frist->llink = NULL;
        return first;
}
```

Function to delete at the rear end

```
NODE delete_rear( NODE first)
{
        NODE prev, cur;
        if(first == NULL)
        {
                printf("list is empty\n");
                return first;
        }
        if(first->rlink == NULL)
        {
                printf("Employee with following details is deleted\n");
                printf("ssn : %s\n", first->ssn);
                printf("name : %s\n", first ->name);
                printf("dept : %s\n", first ->dept);
                printf("desg : %s\n", first ->desg);
                printf("sal : %f\n", first ->sal);
                printf("phone no : %s\n", first->phno);

                free(first)
                return NULL;
        }
        cur = first;
        while(cur->rlink != NULL)
                cur = cur->rlink;
        prev =cur->llink;

        printf("Employee with following details is deleted\n");
        printf("ssn : %s\n", cur->ssn);
        printf("name : %s\n", cur->name);
        printf("dept : %s\n", cur->dept);
        printf("desg : %s\n", cur->desg);
        printf("sal : %f\n", cur->sal);
```

```
        printf("phone no : %s\n", cur->phno);

        free(cur);
        prev->rlink = NULL;
        return first;
}
```

Function to display the contents of the list and count the number of nodes

```
void display(NODE first)
{
        NODE cur;
        int count = 0;
        if(first = = NULL)
        {
                printf("the list is empty\n");
                printf("the number of nodes in the list is %d\n", count);
                return;
        }
        cur = first;
        printf("the contents of the list are:\n");
        printf("ssn\t name\t dept\t desg\t sal\t phoneno\n");

        while(cur != NULL)
        {
                count = count + 1;

                printf("ssn : %s\n", cur->ssn);
                printf("name : %s\n", cur->name);
                printf("dept : %s\n", cur->dept);
                printf("desg : %s\n", cur->desg);
                printf("sal : %f\n", cur->sal);
                printf("phone no : %s\n", cur->phno);

                cur = cur->rlink;
        }
        printf("the number of nodes in the list is %d\n", count);
}
```


printf(cur -> info)

```
void main( )
{
        int choice, n, i;
        char ssn[20], name[20], dept[20], desg[20], phno[20];
        float sal;
        NODE first = NULL;
        clrscr( );
```

```
// Creating a list of 'n' employees using end (rear) insertion

printf("Enter the number of employees");
scanf("%d", &n);
for(i=0; i<n; i++)
{
        printf("Enter ssn\n");
        gets(ssn);
        printf("Enter name\n");
        gets(name);
        printf("Enter dept\n");
        gets(dept);
        printf("Enter desg\n");
        gets(desg);
        printf("Enter sal\n");
        scanf("%f", &sal);
        printf("Enter phone no\n");
        gets(phno);

        first = insert_rear( ssn, name, dept, desg, sal, phno, first);
}
```

Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

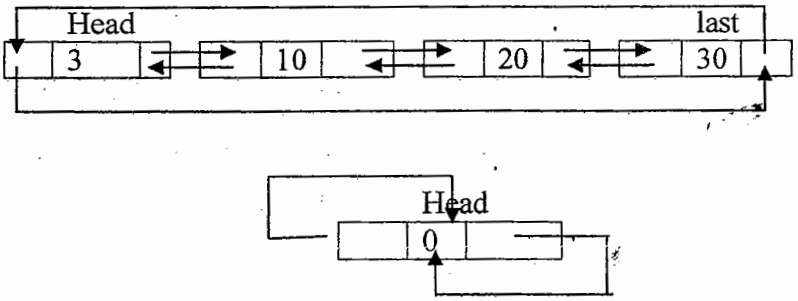
In singly LL we have insert_front

```
for(;;)
{
        printf("1:Insert front\n2:Insert rear\n");
        printf("3:Delete front\n4:Delete rear\n");
        printf("5:display\n6:exit\n");

        printf("enter your choice");
        scanf("%d",&choice);

        switch(choice)
        {
            case 1: printf("Enter ssn\n");
                    gets(ssn);
                    printf("Enter name\n");
                    gets(name);
                    printf("Enter dept\n");
                    gets(dept);
                    printf("Enter desg\n");
                    gets(desg);
                    printf("Enter sal\n");
                    scanf("%f", &sal);
                    printf("Enter phone no\n");
                    gets(phno);
```

```
                first = insert_front( ssn, name, dept, desg, sal, phno, first);
                break;

        case 2: printf("Enter ssn\n");
                gets(ssn);
                printf("Enter name\n");
                gets(name);
                printf("Enter dept\n");
                gets(dept);
                printf("Enter desg\n");
                gets(desg);
                printf("Enter sal\n");
                scanf("%f", &sal);
                printf("Enter phone no\n");
                gets(phno);

                first = insert_rear( ssn, name, dept, desg, sal, phno, first);
                break;

        case 3: first = delete_front(first);
                break;

        case 4: first = delete_rear(first);
                break;

        case 5: display(first);
                break;

        default : exit(0);
    }
    getch( );
  }
}
```

Circular Doubly Linked List with header

Here if the list is empty, llink of head contains head and rlink of head also contains head, otherwise llink of header node contains address of the last node and rlink of header node contains address of the first node. The link field of last node contains address of header node. The pictorial representation of this is as shown below.



Note: In the following functions we use the structure definition as shown below.

```
struct node
{
        int info;
        struct node *llink;
        struct node *rlink;
};
typedef struct node * NODE;
```

Function to insert an element at the front end

```
NODE insertfront(int item, NODE head)
{
        NODE temp, cur;
        temp = (NODE) malloc(sizeof(struct node));
        temp->info = item;
        cur = head->rlink;
        head->rlink = temp;
        temp->llink = head;
        temp->rlink = cur;
        cur->llink = temp;
        return head;
}
```


Function to insert an element at the rear end

```
NODE insertrear(int item, NODE head)
{
        NODE temp, last;
        temp = (NODE) malloc(sizeof(struct node));
        temp->info = item;
        last = head->llink;
        temp->llink = last;
        last->rlink = temp;
        temp->rlink = head;
        head->llink = temp;

        return head;
}
```

Function to delete an element at the front end

```
NODE deletefront(NODE head)
{
        NODE cur, next;
        if(head->rlink = = head)
        {
                printf("list is empty\n");
                return head;
        }
        cur = head->rlink;
        next = cur->rlink;
        head->rlink = next;
        next->llink = head;

        printf("element deleted is %d\n", cur->info);
        free(cur);
        return head;
}
```

Function to delete an element at the rear end

```
NODE deleterear(NODE head)
{
        NODE cur, last;
        if(head->rlink = = head)
        {
                printf("list is empty\n");
                return head;
        }
        cur = head->llink;
        last = cur->llink;
        last->rlink = head;
        head->llink = last;

        printf("element deleted is %d\n", cur->info);
        free(cur);
        return head;
}
```


Function to display the contents of the list

```
void display(NODE head)
{
        NODE cur;
        if(head->rlink = = head)
        {
                printf("the list is empty\n");
                return;
        }
        printf("the contents of the list are:\n");
        cur = head->rlink;
```

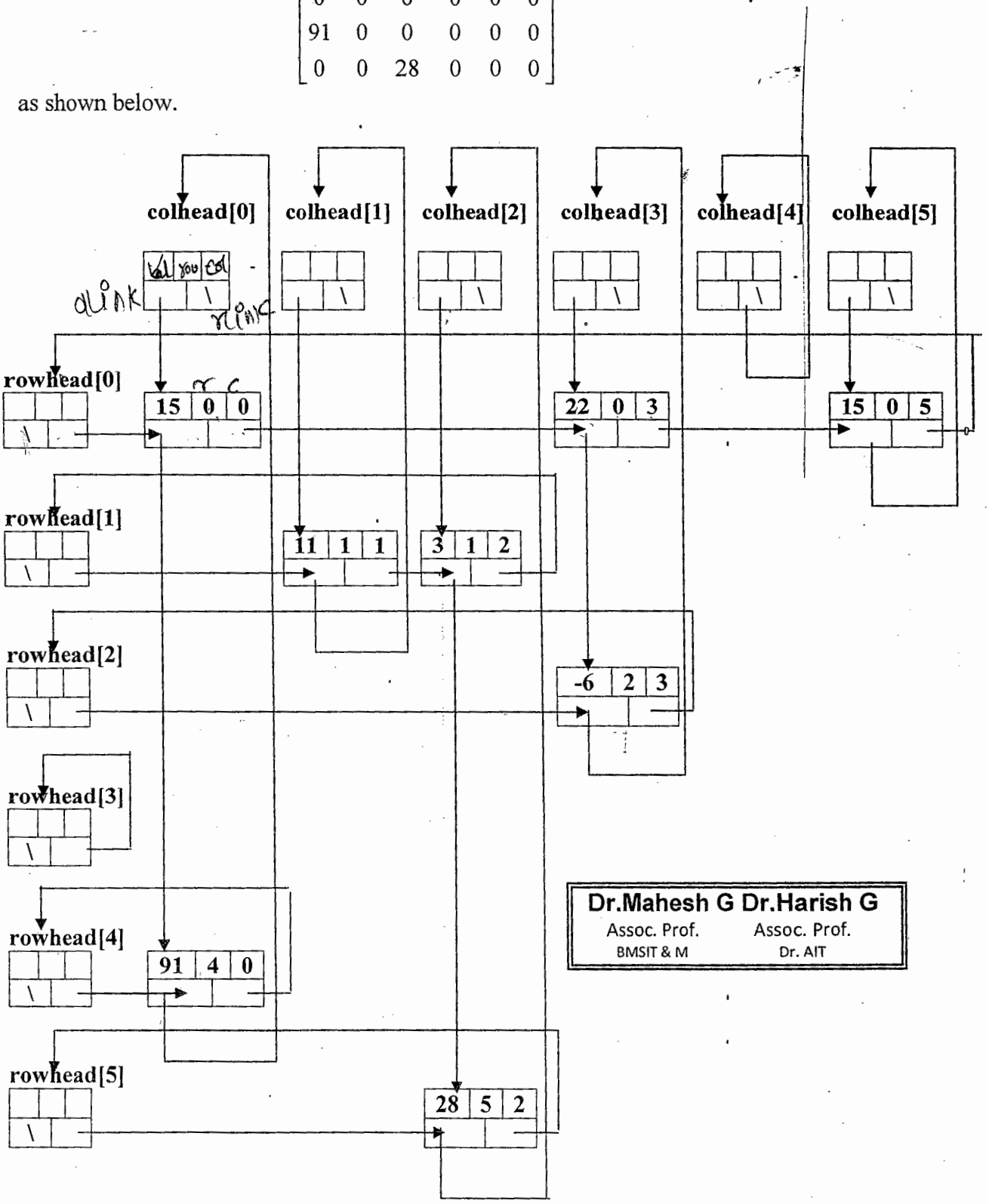
```
        while(cur != head)
        {
                printf("%d\n",cur->info);
                cur = cur->rlink;
        }
}
```

Note: Updating head->info is optional.

```
// Main function for circular doubly linked lists with header node
void main( )
{
        int choice, item;
        NODE head;
        head = (NODE) malloc(sizeof(struct node));
        head->llink = head->rlink=head;
        clrscr( );
        for(;;)
        {
                printf("\n1:insert-front\n 2:insert-rear\n ");
                printf("\n3:delete-front\n 4:delete-rear\n ");
                printf("\n5:display\n 6:exit\n ");
                printf("enter your choice");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1: printf("enter the item to be inserted\n");
                                scanf("%d",&item);
                                head = insert_front(item, head);
                                break;
                        case 2: printf("enter the item to be inserted\n");
                                scanf("%d",&item);
                                head = insert_rear(item, head);
                                break;

                        case 3: head = delete_front(head);
                                break;
                        case 4: head = delete_rear(head);
                                break;
                        case 5: display(head);
                                break;
                        default : exit(0);
                }
                getch( ):
        }
}
```

Sparse Matrix Representation using linked lists

Consider the sparse matrix, $\begin{bmatrix} 15 & 0 & 0 & 22 & 0 & 15 \\ 0 & 11 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & -6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 91 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 28 & 0 & 0 & 0 \end{bmatrix}$, the link list representation of this is as shown below.



Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

Sparse Matrix Using Linked Lists

```
#include<stdio.h>

struct node
{
        int value;
        int row;
        int col;
        struct node *rlink;
        struct node *dlink;
};
typedef struct node* NODE;

void insert_matrix(NODE rowhead[ ], NODE colhead[ ], int value, int row, int col)
{
        NODE temp, cur, prev, head;

        temp = (NODE)malloc(sizeof(struct node));
        temp->value = value;
        temp->row = row;
        temp->col = col;

        // Insert into appropriate column
        head = rowhead[row];
        prev = head;
        cur = head->rlink;
        while(cur!=head && cur->col < col)
        {
                prev = cur;
                cur = cur->rlink;
        }
        prev->rlink = temp;
        temp->rlink = cur;

        // Insert into appropriate row
        head = colhead[col];
        prev = head;
        cur = head->dlink;
        while(cur!=head && cur->row < row)
        {
                prev = cur;
                cur = cur->dlink;
        }
        prev->dlink = temp;
        temp->dlink = cur;
}
```

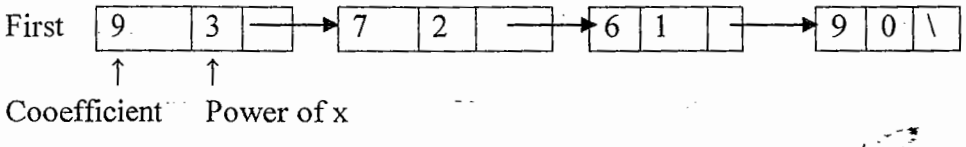
```
void read_matrix(NODE rowhead[ ], NODE colhead[ ], int m, int n)
{
        int value, i, j;

        printf("\n\nEnter the elements of the matrix");
        for(i=0;i<m;i++)
        {
                for(j=0;j<n;j++)
                {
                        scanf("%d",&value);
                        if(value != 0)
                        {
                                insert_matrix(rowhead, colhead, value, i, j);
                        }
                }
}
void display(NODE rowhead[ ],int m)
{
        int i;
        NODE head, cur;
        for(i=0; i<m; i++)
        {
                head = rowhead[i];
                cur = head->rlink;
                while(cur != head)
                {
                        printf("(%d, %d) = %d\n", cur->row, cur->col, cur->value);
                        cur = cur->rlink;
                }
        }
}
void search(int key, NODE rowhead[ ],int m)
{
        int i;
        NODE head, cur;
        for(i=0; i<m; i++)
        {
                head = rowhead[i];
                cur = head->rlink;
                while(cur != head)
                {
                        if(key == cur->value)
                        {
                                printf("Successful Search\n");
                                printf(" Element found at row %d and column %d\n"), cur->row, cur->col);
                                return;
                        }
                        cur = cur->rlink;
                }
        }
```

```
        printf(" Unsuccessful Search Element not Found\n");
}

void main( )
{
        int m, n
        NODE rowhead[20], colhead[20], temp,

        printf("Enter the number of rows");
        scanf("%d",&m);

        printf("Enter the number of columns");
        scanf("%d",&n);

        // Initialize row headers
        for(i=0; i<m; i++)
        {
                temp = (NODE)malloc(sizeof(struct node));
                temp->rlink = temp;
                temp->dlink = NULL;
                rowhead[i] = temp;
        }

        // Initialize column headers
        for(i=0; i<n; i++)
        {
                temp = (NODE)malloc(sizeof(struct node));
                temp->rlink = NULL;
                temp->dlink = temp;
                colhead[i] = temp;
        }

        // Read the matrix elements
        read_matrix(rowhead, colhead, m, n);

        // Print the matrix
        display(rowhead, m);

        // Read the key element to be searched
        printf("Enter the element to be searched \n");
        scanf("%d",&key);

        // Search for the key in the matrix
        search(key, rowhead, m);

        getch( );
}
```

Polynomial – Using Singly Linked List

Representation

Consider the polynomial $9x^3 + 7x^2 + 6x + 9$. This polynomial can be represented using a singly linked list as shown below.



Every node has 3 fields

- ✓ cf – coefficient field
- ✓ px – power of x
- ✓ link – contains the address of next node.

Therefore, the structure declaration to represent this node will be

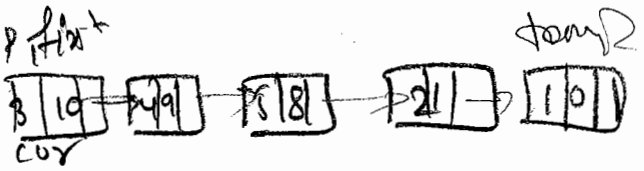
```
struct node
{
        int cf;
        int px;
        struct node *link;
};
typedef struct node* NODE;
```


Program to add two Polynomials using Singly Linked List

```
#include<stdio.h>

// Function to display the polynomial
void display(NODE first)
{
        NODE cur;
        if(first == NULL)
        {
                printf("Polynomial does not exist\n");
                return;
        }
        cur = first;
        while(cur != NULL)
        {
                if(cur->cf > 0)
                        printf("+");
                printf("%d x^ %d",cur->cf, cur->px);
                cur = cur->link;
        }
}
```

$3x^{10} + 4x^9 + (x^8 + 2x + 1$
$5x^{10} \longrightarrow 4x^9 + 3x^6$

```
// Function to insert an element at the rear end
NODE insert_rear(int cf, int px, NODE first)
{
        NODE temp, cur;

        temp=(NODE)malloc(sizeof(struct node));
        temp->cf = cf;
        temp->px = px;
        temp->link = NULL;

        if(first = = NULL)   //no node
        {
                return temp;
        }

        cur = first;              // node exists
        while(cur->link != NULL)
        {
                cur = cur->link;
        }

        cur->link = temp;
        return first;
}
```



P_3 NULL

Dr.Mahesh G Dr.Harish G
Assoc. Prof. Assoc. Prof.
BMSIT & M Dr. AIT

```
// Function to read a polynomial
NODE readpoly(NODE first)
{
        int n, i, cf, px;
        printf("Enter the number of terms \n");
        scanf("%d", &n);
        for(i = 1; i<=n; i++)
        {
                printf("Enter the coefficient and power of x of %d term\n",i);
                scanf("%d%d",&cf, &px);
                first = insert_rear(cf, px, first);
        }
        return first;
}
// Function to compare 2 numbers
int compare(int x, int y)
{
        if(x<y)
                return -1;
        else if(x = =y)
                return 0;
             else
```

```
            return 1;
}
// Function to add 2 polynomials
NODE addpoly(NODE p1, NODE p2, NODE p3)
{
        int sumcf;
        while(p1 != NULL && p2 != NULL)
        {
                switch( compare(p1->px, p2->px ))
                {
                        case 0 : // p1's exponent  = p2's exponent
                                sumcf = p1->cf + p2->cf;
                                if(sumcf != 0)
                                p3 = insert_rear(sumcf, p1->px, p3);
                                p1 = p1->link;
                                p2 = p2->link;
                                break;

                        case  1: // p1's exponent > p2's exponent
                                p3 = insert_rear(p1->cf, p1->px, p3);
                                p1 = p1->link;
                                break;

                        case  -1: // p1's exponent < p2's exponent
                                p3 = insert_rear(p2->cf, p2->px, p3);
                                p2 = p2->link;
                                break;
                }
        }

        // Add remaining terms of Polynomial 1
        while(p1 != NULL)
        {
                p3 = insert_rear(p1->cf, p1->px, p3);
                p1 = p1->link;
        }

        // Add remaining terms of Polynomial 2
        while(p2 != NULL)
        {
                p3 = insert_rear(p2->cf, p2->px, p3);
                p2 = p2->link;
        }
        return p3;
}

void main( )
{
        NODE p1 = NULL, p2 = NULL, p3 = NULL;

        printf("enter the first polynomial\n");
        p1 = readpoly(p1);
```

P_3 is sum of P_1 & P_2

$4x^9 - 4x^9$

P_1 P_2

$5x^8 > 3x^6$

P_1 P_2

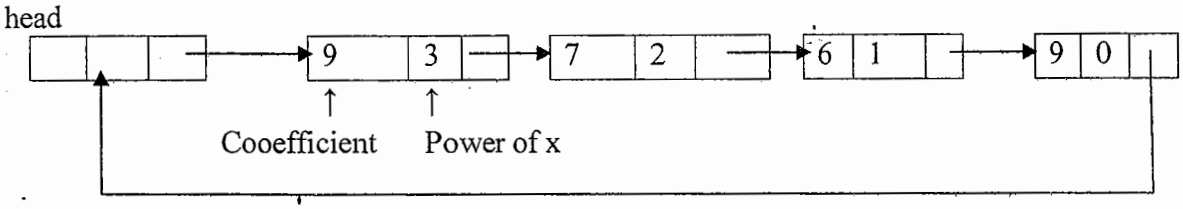
$2x < 3x^6$

```
        printf("enter the second polynomial\n");
        p2 = readpoly(p2);

        p3 = addpoly(p1, p2, p3)

        printf("The first polynomial is\n");
        display(p1);

        printf("The second polynomial is\n");
        display(p2);

        printf("Their sum is\n");
        display(p3);
}
```


Polynomial – Using Circular Singly Linked List with header node

Representation

Consider the polynomial $9x^3 + 7x^2 + 6x + 9$. This polynomial can be represented using a circular singly linked list with header node as shown below.



Note: link field of last node contains the address of head and link field of header node points to the first node of the list.

Every node has 3 fields

- ✓ cf – coefficient field
- ✓ px – power of x
- ✓ link – contains the address of next node.

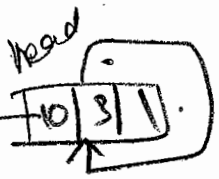
Therefore, the structure declaration to represent this node will be

```
struct node
{
        int cf;
        int px;
        struct node *link;
};
typedef struct node* NODE;
```

Program to add two Polynomials using Circular Singly Linked List header node

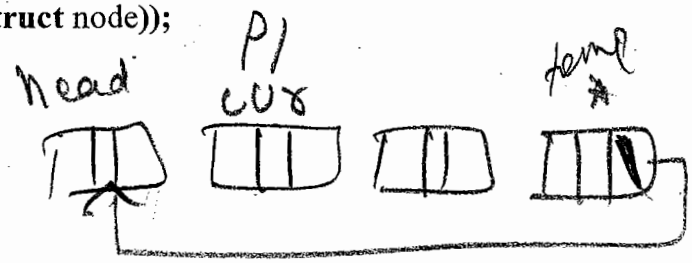
```
#include<stdio.h>

// Function to display the polynomial
```

```
void display(NODE head)
{
        NODE cur;
        if(head->link = = head)
        {
                printf("Polynomial does not exist\n");
                return;
        }
        cur = head->link;
        while(cur != head)
        {
                if(cur->cf > 0)
                        printf("+");
                printf("%d x^ %d",cur->cf, cur->px);
                cur = cur->link;
        }
}
```



first = head->link


```
// Function to insert an element at the rear end
NODE insert_rear(int cf, int px, NODE head)
{
        NODE temp, cur;

        temp=(NODE)malloc(sizeof(struct node));
        temp->cf = cf;
        temp->px = px;
        temp->link = NULL;

        cur = head->link;
        while(cur->link != head)
        {
                cur = cur->link;
        }
        cur->link = temp;
        temp->link = head;
        return head;

}

// Function to read a polynomial
NODE readpoly(NODE head)
{
        int n, i, cf, px;
        printf("Enter the number of terms \n");
        scanf("%d", &n);
        for(i = 1; i<=n; i++)
```



```
        {
                printf("Enter the coefficient and power of x of %d term\n",i);
                scanf("%d%d",&cf, &px);
                head = insert_rear(cf, px, head);
        }
        return head;
}
// Function to compare 2 numbers
int compare(int x, int y)
{
        if(x<y)
                return -1;
        else if(x = =y)
                return 0;
             else
                return 1;
}
```



```
// Function to add 2 polynomials
NODE addpoly(NODE h1, NODE h2, NODE h3)
{
        NODE p1, p2;
        p1 = h1->link;
        p2 = h2->link;
        int sumcf;
        while(p1 != h1 && p2 != h2)
        {
                switch( compare(p1->px, p2->px ))
                {
                        case 0 : // p1's exponent  = p2's exponent
                                sumcf = p1->cf + p2->cf;
                                if(sumcf != 0)
                                h3 = insert_rear(sumcf, p1->px, h3);
                                p1 = p1->link;
                                p2 = p2->link;
                                break;

                        case  1: // p1's exponent > p2's exponent
                                h3 = insert_rear(p1->cf, p1->px, h3);
                                p1 = p1->link;
                                break;

                        case  -1: // p1's exponent < p2's exponent
                                h3 = insert_rear(p2->cf, p2->px, h3);
                                p2 = p2->link;
                                break;
                }
        }

        // Add remaining terms of Polynomial 1
        while(p1 != h1)
```

```
        {
                h3 = insert_rear(p1->cf, p1->px, h3);
                p1 = p1->link;
        }

        //Add remaining terms of Polynomial 2
        while(p2 != h2)
        {
                h3 = insert_rear(p2->cf, p2->px, h3);
                p2 = p2->link;
        }
        return h3;
}

void main( )
{
        NODE h1, h2, h3;

        h1 = (NODE)malloc(sizeof(struct node));
        h2 = (NODE)malloc(sizeof(struct node));
        h3 = (NODE)malloc(sizeof(struct node));

        h1->link = h1;
        h2->link = h2;
        h3->link = h3;

        printf("enter the first polynomial\n");
        h1 = readpoly(h1);

        printf("enter the second polynomial\n");
        h2 = readpoly(h2);

        h3 = addpoly(h1, h2, h3)

        printf("The first polynomial is\n");
        display(h1);

        printf("The second polynomial is\n");
        display(h2);

        printf("Their sum  is\n");
        display(h3);
}
```

Miscellaneous Functions

Write a 'C' function called strcmpl(L1, L2) to compare 2 character strings represented as lists L1 and L2. This function should return '0' if L1==L2, -1 if L1 < L2 and 1 if L1 >L2.

```
int strcmpl(NODE L1, NODE L2)
{
        NODE C1, C2;
        C1 = L1;
        C2 = L2;
        while( C1 != NULL && C2 != NULL)
        {
                if(C1->info > C2->info)
                        return 1;
                else if(C1->info < C2->info)
                        return -1;
                else
                {
                        C1 = C1->link;
                        C2 = C2->link;
                }
        }
        if(C1 == NULL && C2 != NULL)
                return -1;

        if(C2 == NULL && C1 != NULL)
                return 1;
        return 0;
}
```


Write a 'C' function to check whether a given string is a palindrome assuming that the string is stored in a circular doubly linked list with a header.

```
void palindrome(NODE head)
{
        NODE cur, last;
        cur = head->rlink;
        last=head->llink;
        while(cur!=head)
        {
                if(cur->info != last->info)
                {
                        printf("string is not a palindrome\n");
                        return;
                }
                cur = cur->rlink;
                last = last->llink;
        }
        printf("string is a palindrome\n");
}
```

Write a 'C' function search(P, x) that accepts a pointer 'P' to the list of integers and an integer 'x' and returns a pointer to a node containing 'x' if it exists and NULL otherwise.

```
NODE search(NODE P, int x)
{
        NODE cur;
        cur = P;
        while(cur != NULL && x != cur->info)
                cur=cur-link;
        return cur;
}
```

Write a 'C' function srchinst(P, x) that adds 'x' to P if it is not found and always returns a pointer containing 'x'.

```
NODE srchinst(NODE P, int x)
{
        NODE cur, temp, prev;
        cur = P;
        prev = NULL;
        while(cur != NULL && x != cur->info)
        {
                prev =cur;
                cur=cur->link;
        }
        if(cur!= NULL)
                return cur;
        else
        {
                temp = (NODE)malloc(sizeof(struct node));
                temp->info = x;
                temp->link = NULL;
                if(prev==NULL)
                {
                        P=temp;
                        return temp;
                }
                prev->link=temp;
                return temp;
        }
}
```


Note: Instead of

```
temp->link = NULL;
if(prev==NULL)
{
        P=temp;
        return temp;
}
prev->link=temp;
```

we can write only **temp->link = P;**

Miscellaneous Functions

Write a 'C' function called strcmpl(L1, L2) to compare 2 character strings represented as lists L1 and L2. This function should return '0' if L1==L2, -1 if L1 < L2 and 1 if L1 >L2.

```
int strcmpl(NODE L1, NODE L2)
{
        NODE C1, C2;
        C1 = L1;
        C2 = L2;
        while( C1 != NULL && C2 != NULL)
        {
                if(C1->info > C2->info)
                        return 1;
                else if(C1->info < C2->info)
                        return -1;
                else
                {
                        C1 = C1->link;
                        C2 = C2->link;
                }
        }
        if(C1 == NULL && C2 != NULL)
                return -1;

        if(C2 == NULL && C1 != NULL)
                return 1;
        return 0;
}
```


Write a 'C' function to check whether a given string is a palindrome assuming that the string is stored in a circular doubly linked list with a header.

```
void palindrome(NODE head)
{
        NODE cur, last;
        cur = head->rlink;
        last=head->llink;
        while(cur!=head)
        {
                if(cur->info != last->info)
                {
                        printf("string is not a palindrome\n");
                        return;
                }
                cur = cur->rlink;
                last = last->llink;
        }
        printf("string is a palindrome\n");
}
```

Write a 'C' function rotate which accepts the header pointer to a circular doubly linked list and 'n' a integer value to rotate the elements to the left 'n' times.

```
NODE rotate_left(int n, NODE head)
{
        NODE cur, prev;
        int i;

        for(i=0; i<n; i++)
        {
                prev = head;
                cur = head->rlink;

                while(cur!=head)
                {
                        prev->info = cur->info;
                        prev = cur;
                        cur = cur->rlink;
                }
                prev->info = cur->info;
        }
        return head;
}
```

Write a 'C' function to find the minimum node in a singly linked list.

```
NODE minimum(NODE first)
{
        NODE min, cur;
        if(first==NULL)
        {
                printf("list is empty\n");
                return NULL;
        }
        min=first;
        cur=first->link;
        while(cur!=NULL)
        {
                if(min->info > prev->info)
                {
                        min=cur;
                        cur = cur->link;
                }
                else
                        cur=cur->link;
        }
        return min;
}
```

NOV - 4

FPL - 10 - 12 slides

AS size design

code, report, ppt, poster

ADE
DMS
DS
} Assign

372

SPC " %o[^\n],s")

scanf (" %[^\n]s", str);