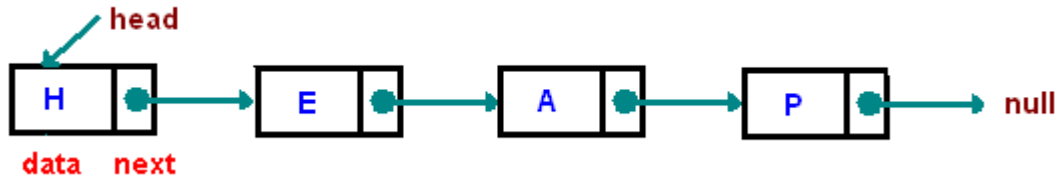
MODULE 3

LINKED LISTS

Linked list is a linear data structure that consists of a sequence of elements where each element (usually called a **node**) comprises of two items - the data and a reference (link) to the next node. The last node has a reference to null. The entry point into a linked list is called the **head (start)** of the list. It should be noted that head is not a separate node, but the reference to the first node. If the list is empty then the start is a null reference. The list with no nodes –empty list or null list.



Note: The head is a pointer which points to the first node in the list. The implementation in the class, it has been discussed with the pointer's name as start. Head or start refers to the starting node.

A linked list is a dynamic data structure. The number of nodes in a list is not fixed and can grow and shrink on demand. Any application which has to deal with an unknown number of objects will need to use a linked list.

Arrays –drawbacks

(1)The size of the arrays is fixed: So we must know the upper limit on the number of elements in advance. (static in nature)

(2) Inserting and a new element in an array of elements is expensive, because room has to be created for the new elements and to create room existing elements have to shifted. Same holds good for deletion also.

Advantages of linked list:

Efficient memory utilization: The memory of a linked list is not pre-allocated. Memory can be allocated whenever required. And it is de-allocated when it is no longer required.

Insertion and deletion operations are easier and efficient:
Linked list provide flexibility in inserting a data item at a specified position and deletion of a data item from the given position.

Extensive manipulation:
We can perform any number of complex manipulations without any prior idea of the memory space available. (i.e. in stacks and queues we sometimes get overflow conditions. Here no such problem arises.)

Arbitrary memory locations
Here the memory locations need not be consecutive. They may be any arbitrary values. But even then the accessing of these items is easier as each data item contains within itself the address to the next data item. Therefore, the elements in the linked list are ordered not by their physical locations but by their logical links stored as part of the data with the node itself.

As they are dynamic data structures, they can grow and shrink during the execution of the
program

Disadvantages of linked lists

- They have a tendency to use more memory due to pointers requiring extra storage space.
- Nodes in a linked list must be read in order from the beginning as linked lists are inherently sequentially accessed.(cannot be randomly accessed)
- Nodes are stored incontiguously, greatly increasing the time required to access individual elements within the list.
- Difficulties arise in linked lists when it comes to reverse traversing. For instance, singly linked lists are cumbersome to navigate backwards[1] and while doubly linked lists are somewhat easier to read, memory is wasted in allocating space for a back pointer.

Note : No particular data structure is the best. The choice of the data structure depends on the kind of application that needs to be implemented. While for some applications linked lists may be useful, for others, arrays may be useful.

Operations on linked lists

- Creation of a list

  Creation operation is used to create a linked list. Once a linked list is created with one node, insertion operation can be used to add more elements in a node.

- Insertion of an element into a linked list

  Insertion operation is used to insert a new node at any specified location in the linked list. A new node may be inserted.

(a) At the beginning of the linked list
(b) At the end of the linked list
(c) At any specified position in between in a linked list

- Deletion of a node from the linked list

  Deletion operation is used to delete an item (or node) from the linked list. A nodemay be deleted from the
  (a) Beginning of a linked list
  (b) End of a linked list
  (c) Specified location of the linked list

- Traversing and displaying the elements in the list

  Traversing is the process of going through all the nodes from one end to another end of a linked list. In a singly linked list we can visit from lptr to rptr, forward traversing,nodes only. But in doubly linked list forward and backward traversing is possible

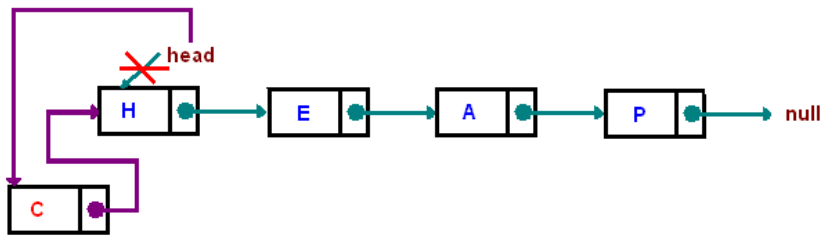
- Counting the number of elements in the list
- Searching for an element in the list
- Merging two lists(Concatenating lists)

  Merging is the process of appending the second list to the end of the first list. Consider a list A having n nodes and B with m nodes. Then the operation concatenation will place the 1st node of B in the (n+1)th node in A. After concatenation A will contain (n+m) nodes

Linked List Basic Operations

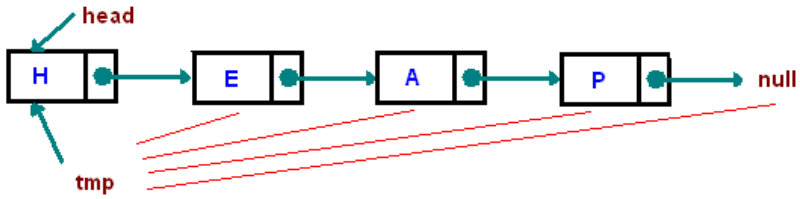
INsertBegin

The method creates a node and inserts it at the beginning of the list.



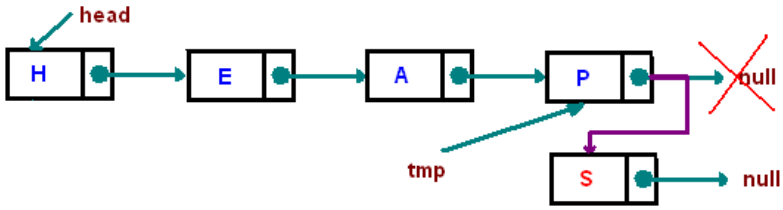
Traversing

Start with the head and access each node until you reach null. Do not change the head reference.



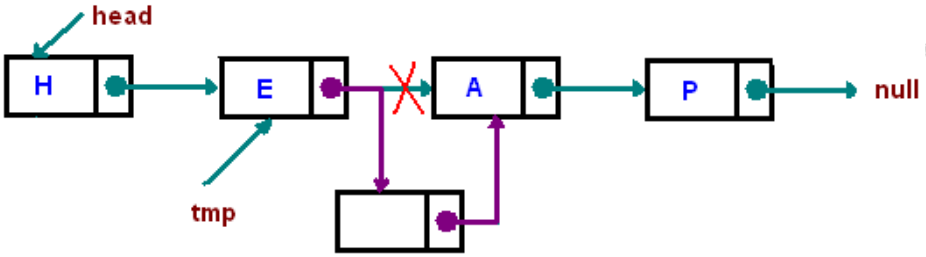
INsertend

The method appends the node to the end of the list. This requires traversing, but make sure you stop at the last node



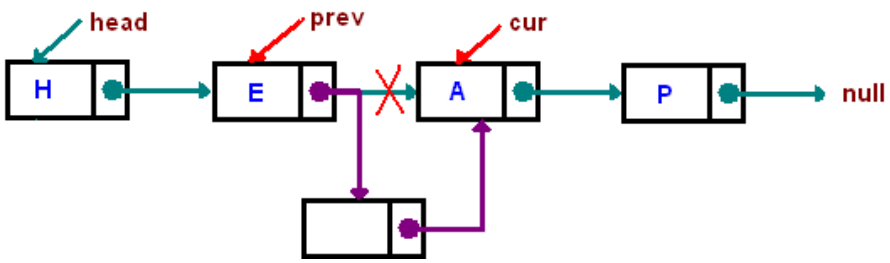
Inserting "after"

Find a node containing "key" and insert a new node after it. In the picture below, we insert a new node after "E":



Inserting "before"

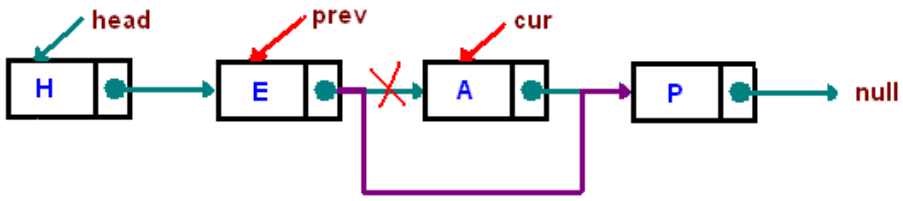
Find a node containing "key" and insert a new node before that node. In the picture below, we insert a new node before "A":



For the sake of convenience, we maintain two references prev and cur. When we move along the list we shift these two references, keeping prev one step before cur. We continue until cur reaches the node before which we need to make an insertion. If cur reaches null, we don't insert, otherwise we insert a new node between prev and cur.

Deletion

Find a node containing "key" and delete it. In the picture below we delete a node containing "A"



The algorithm is similar to insert "before" algorithm. It is convinient to use two references prev and cur. When we move along the list we shift these two references, keeping prev one step before cur. We continue until cur reaches the node which we need to delete. There are three exceptional cases, we need to take care of:

1. list is empty
2. delete the head node
3. node is not in the list

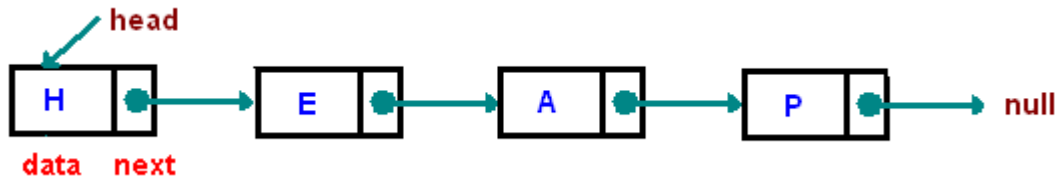
Type of linked lists

Depending on the way in which the links are used to maintain adjacency, several different types of linked lists are possible

1. Singly linked list
2. Doubly linked list
3. Circular linked list

Singly linked list

In a singly linked list, each node except the last one contains a single pointer to the next element.The last node has a null pointer to indicate the termination of the list.

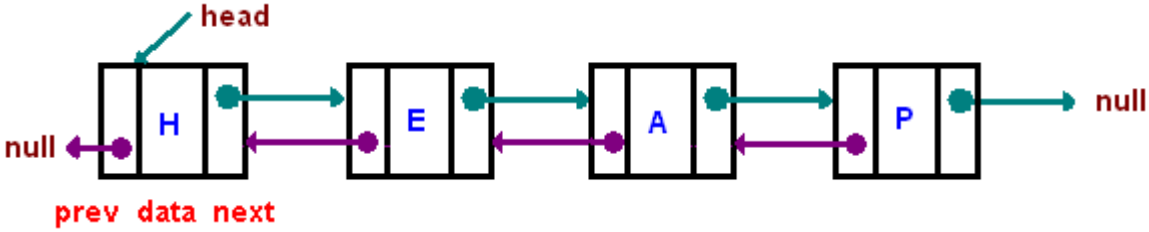


Doubly Linked List

A **doubly linked list** is a linked data structure that consists of a set of sequentially linked nodes. Each node contains two fields, called *links*, that are references to the previous and to the next node in the sequence of nodes. The beginning and ending nodes' **previous** and **next** links, respectively, point to NULL, to facilitate traversal of the list.

The two node links allow traversal of the list in either direction. While adding or removing a node in a doubly linked list requires changing more links than the same operations on a singly linked list, but the operations are simpler and more efficient **(for nodes other than first nodes)** because there is no need to keep track of the previous node during traversal or no need to traverse the list to find the previous node, so that its link can be modified.

Double-linked lists require more space per node , and their elementary operations are more expensive; but they are often easier to manipulate because they allow sequential access to the list in both directions. In particular, one can insert or delete a node in a constant number of operations given only that node's address. (Compared with singly- , which require the previous node's address in order to correctly insert or delete.) Some algorithms require access in both directions.



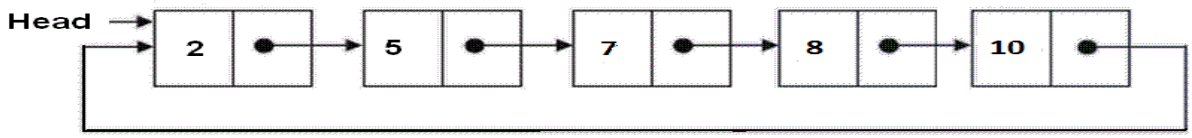
CIRCULAR LINKED LIST

In a circularly-linked list, the first and final nodes are linked together. This can be done for both singly and doubly linked lists. To traverse a circular linked list, you begin at any node and follow the list in either direction until you return to the original node. Viewed another way, circularly-linked lists can be seen as having no beginning or end. This type of list is most useful for managing buffers for data ingest, and in cases where you have one object in a list and wish to iterate through

all other objects in the list in no particular order.Note that a circular lisr does not have a natural "first" or a "last" node. *There is no NULL at the end. A circular linked list can be a singly circular linked list or doubly circular linked list.*

Convention(usual practice)

Let a **pointer(last)** point to last node of circular list and allow the following node to be the first node. If **last** is pointer to the last node of the circular list, the last node can be referenced by the last pointer and the first node by referencing last->next. If last ==NULL, it indicates an empty list



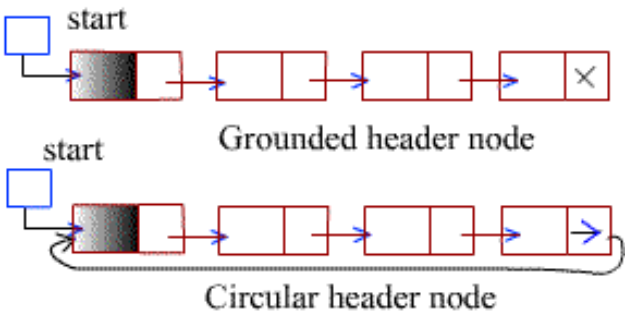
Advantages of Circular Linked Lists:

1) Any node can be a starting point. We can traverse the whole list by starting from any point. We just need to stop when the first visited node is visited again.

2) Circular lists are useful in applications to repeatedly go around the list. For example, when multiple applications are running on a PC, it is common for the operating system to put the running applications on a list and then to cycle through them, giving each of them a slice of time to execute, and then making them wait while the CPU is given to another application. It is convenient for the operating system to use a circular list so that when it reaches the end of the list it can cycle around to the front of the list.

3) Multiplayer board games can be implemented using circular lists where each player waits for his turn to play in a circular fashion.

Header Linked List.



A header linked list is a linked list which always contains a special node called the

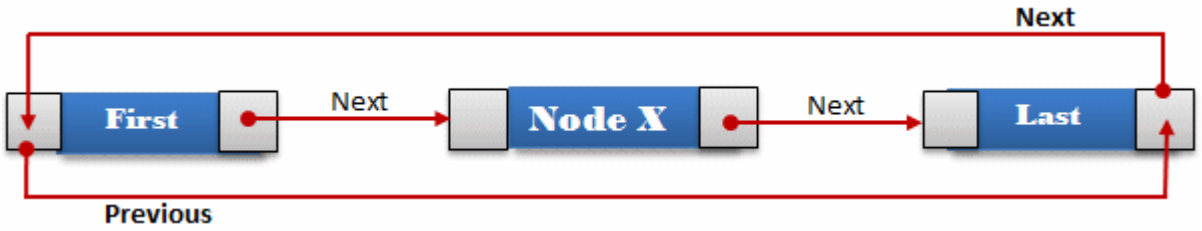
header node at the beginning of the list. It is an extra node kept at the front of a list. **Such a node does not represent an item in the list**. The information portion might be unused. There are two types of header list

1. ***Grounded header list*: is a header list where the last node contains the null pointer.**
2. ***Circular header list*: is a header list where the last node points back to the header node.**

More often , the information portion of such a node could be used to keep **global information** about the entire list such as:

- number of nodes (not including the header) in the list count in the header node must be adjusted after adding or deleting the item from the list
- pointer to the last node in the list it simplifies the representation of a queue
- pointer to the current node in the list eliminates the need of a external pointer during traversal

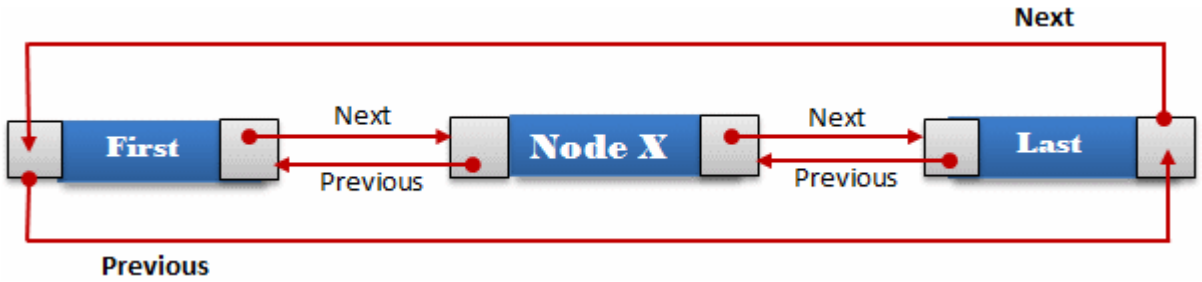
Circular Singly Linked Lists



IN a singly linked circular list, the pointer field of the last node stores the address of the starting node
In the list.Hence it is easy to traverse the list given the address of any node in the list.

Circular Doubly Linked List

A *circular, doubly-linked list* is shown in Figure . The last element of the list is made the predecessor of the first node,and the first element is the successor of the last. We no longer need both a head and tail variable to keep track of the list. Even if only a single variable is used, both the first and the last list elements can be found.



1. Implementation of Singly linked list

```
#include<stdio.h>
#include<stdlib.h>
struct SLL
{
        char info;
        struct SLL *next;
};
typedef struct SLL node;
node *start;

node *getnodeSLL()
{
        node *newnode;
        new1=(node*)malloc(sizeof(node));
        printf("\n Enter the data");
        scanf("%d", &new1->info);
        new1->next=NULL;
        return new1;
}

void insert_front()
{
        node *n1;
        n1=getnodeSLL();
        if(start==NULL)
        {
                start=n1;
                return;
        }
                n1->next=start;
                start=n1;

}

void delete_front()
{
```

```
        node *temp = start;
        if(start==NULL)
        {
                printf("\n List is empty");
                return;
        }
        printf("\nt%d\t is deleted ",temp->info);
        start=temp->next;
        free(temp);
        }
}

void insert_end()
{
        node *n1,*temp = atsrt;
        n1=getnodeSLL();
        if(start==NULL)
        {
                start=n1;
                return;
        }
        while(temp-> next!=NULL)
                temp=temp->next;

        temp->next=n1;
}

void delete_end()
{
        node *temp=start, *prev;
        if(start==NULL)
        {
                printf("\n Empty list");
                return;
        }
        if(start->next==NULL)
        {
                printf("\nt%d\t is deleted",start->info);
                free(start);
                start=NULL; return;
        }
        while(temp->next != NULL)
        {
                prev = temp;
```

```
            temp = temp->next;
        }
        prev->next = NULL;
        printf("\nThe deleted node is %d\t", temp->info);
        free(temp);
}



void display()
{
        node *temp=start;
        if(start==NULL)
        {
                printf("\n Empty list");
                return;
        }
        printf("\n The details are");
        while(temp!=NULL)
        {
                printf("\n %d\t",temp->info);
                temp=temp->next;
        }

}

int main()
{
        int n,m,i;
        while(1)
        {
                printf("\n Enter 1:insert_front\n 2:insert_end\n 3:delete_front\n
                4:delete_end\n 5:display");
                scanf("%d",&m);
                switch(m)
                {
                        case1: insert_front();  break;
                        case 2: insert_end();break;
                        case 3: delete_front();break;
                        case 4 : delete_end();break;
                        case 5:display();break;
                        case 6: exit(0);
                }
```

```
        }
        return 0;
}
```

2. Implementation of Doubly Linked list

```
#include<stdio.h>
#include<stdlib.h>
struct DLL
{
        int info;
        struct DLL *lptr,*rptr;
};
typedef struct DLL node;
node *start=NULL;

node *getnodeDLL()
{
        node *new1;
        new1=(node*)malloc(sizeof(node));
        printf("\n Enter the data");
        scanf("%d", &new1->info);
        new1->lptr=NULL;
        new1->rptr=NULL;
        return new1;
}

void insert_front()
{
        node *n;
        n=getnodeDLL();
        if(start == NULL)
        {
                start = n;
                return;
        }
        n->rptr=start;
        start->lptr=n;
        start=n;
}

void delete_front()
```

```
{
        node *temp = start;
        if(start==NULL)
        {
                printf("\n List is empty");
                return;
        }
        printf("\nt%d\t is deleted ",temp->info);
        start=temp->rptr;
        start->lptr=NULL;
        free(temp);
}


void insert_end()
{
        node *n1,*temp = start;
        n1=getnodeDLL();
        if(start==NULL)
        {
                start=n1;
                return;
        }
        while(temp->rptr!=NULL)
                temp=temp->rptr;

        temp->rptr=n1;
        n1->lptr=temp;
}

void delete_end()
{
        node *temp =start;
        if(start==NULL)
        {
                printf("\n Empty list");
                return;
        }
        if(start->next==NULL)
        {
                printf("\nt%d\t is deleted",start->info);
                free(start);
                start=NULL; return;
        }
```

```
        while(temp->rptr!=NULL)
                temp=temp->rptr;
        (temp->lptr)->rptr=NULL;
        printf("\nThe deleted node is %d\t", temp->info);
        free(temp);

}


void traverse()
{
        node *temp = start;
        if(start==NULL)
        {
                printf("\n Empty list");
                return;
        }
        printf("\n The details are");
        while(temp!=NULL)
        {
                printf("\n %d\t",temp->info);
                temp=temp->rptr;
        }

}

int main()
{
        int n,m,i;
        while(1)
        {
                printf("\n Enter 1:insert_front\n 2:insert_end\n 3:delete_front\n
                4:delete_end\n 5:display");
                scanf("%d",&m);
                switch(m)
                {
                        case1: insert_front();  break;
                        case 2: insert_end();break;
                        case 3: delete_front();break;
                        case 4 : delete_end();break;
                        case 5:traverse();break;
                        case 6: exit(0);
                }
        }
```

```
        return 0;
}
```

3. Implementation of Circular Singly Linked list using last pointer

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

struct CSLL
{
  int info;
  struct CSLL *next;
};
typedef struct CSLL node;
node *last = NULL;

void insert_begin()
{
 node *new1;

 new1 = getnodeSLL();
 if (last == NULL)
 {
   last =new1;
   last->next = last;
    return;
 }
 new1 -> next = last->next;
 last -> next = new1;

}

void insert_end()
{
 node *new1;
 new1 = getnodeSLL();
 if (last == NULL)
 {
    last =new1;
    last->next = last;
```

```
    return;
  }
   new1 -> next = last->next;
   last->next = new1;
   last = new1;
}


void del_end()
{
        node *prev,*temp=last->next;
        if (last == NULL)
        {
                printf("\n empty");
                return;
        }

         if (last->next == temp->next)
        {
                 printf("deleted item is %d  ", last->info);
                free(last);
                 last = NULL;
                return;
         }
        printf("deleted item is %d  ", last->info);
        while(temp->next != last)
             temp=temp->next;

        temp->next = last->next;
        free(last);
        last = temp;
}

void display()
{
        node *temp = last->next;
         if(last == NULL)
        {
                printf("list empty\n");
                return;
        }
         do
        {
                printf("%d   ",temp->info);
```

```
                temp=temp->next;
        }while(temp != last->next);
}

main()
{
   int choice; clrscr();
   while (1)
   {
        printf("\n1.Insert_beg\n 2.Isert_end\n 3.del end \n 4. Display\n 5.Quit\n");
        printf("Enter your choice : ");
        scanf("%d", &choice);
        switch (choice)
        {
           case 1: insert_begin();
           break;
           case 2: insert_end();
           break;
           case 3: del_end();
           break;
           case 4: display();
           break;
           case 5: exit(1);
           default:
                   printf("Wrong choice\n");
        }
    }
  getch();
}
```


4. Program to implement Addition of two polynomials with 3 variables in each term. Use Circular Singly Linked List with header node.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
struct poly
{
  int coef, expo1,expo2,expo3,flag;
  struct poly *next;
};
typedef struct poly node;

void insert_end(node *h, int a, int x, int y, int z)
```

```
{
 node *temp = h->next, *new1;
 new1 = (node*) malloc(sizeof(node));
 new1->coef = a;
 new1->expo1 = x;
 new1->expo2 = y;
 new1->expo3 = z;
 new1->flag=0;
 while(temp->next != h)
  temp = temp -> next;

 temp -> next = new1;
 new1 -> next = h;
}

void read_poly(node *head)
{
  int a,x,y,z;
  char ch;
  do
   {
    printf("\nenter coef & expo1, expo2, expo3\n");
    scanf("%d%d%d%d",&a,&x,&y,&z);
    insert_end(head,a,x,y,z);
    printf("do u want to continue(Y/N) ?");
    ch=getche();
   }while(ch == 'Y' || ch == 'y');
}

void add_poly(node *h1,node *h2,node *h3)
{
  node *p1=h1->next, *p2;
  int x;

  while( p1 != h1)
  {
    p2 = h2 -> next;
    while(p2 != h2)
    {
       if( p1->expo1 == p2->expo1 && p1->expo2 == p2->expo2 && p1->expo3
       == p2->expo3)
        {
              x = p1->coef + p2->coef;
              insert_end(h3,x,p1->expo1,p1->expo2,p1->expo3);
```

```
                p1->flag=1;
                p2->flag=1;
            }
            p2 = p2->next;
        }
        p1 = p1 -> next;

    }
    p1=h1->next;
    p2=h2->next;

    while(p1 != h1)
    {
      if(p1 -> flag==0)
            insert_end(h3, p1->coef, p1->expo1, p1->expo2, p1->expo3);
      p1 = p1 -> next;
    }

    while(p2 != h2)
    {
       if(p2 -> flag == 0)
             insert_end(h3,p2->coef,p2->expo1, p2->expo2, p2->expo3);
       p2 = p2->next;
    }
}

void display(node *h)
{
  node *temp = h->next;
  if(temp == h)
  {
    printf("list empty\n");
    return;
  }
  while(temp != h)
  {
       printf(" %+ dx^%dy^%dz^%d ",temp->coef,temp->expo1,temp->expo2,temp-
>expo3);
     temp=temp->next;
  }
}

void evaluate(node *h)
{
```

```
        int x,y,z,sum=0;
        node *temp = h->next;
        printf("\nEvauate the resultant polynomial by giving values for X, Y and Z");
        scanf("%d%d%d",&x,&y,&z);
        while(temp != h)
        {
                sum = sum + temp->coef * pow(x,temp->expo1) * pow(y,temp->expo2) * pow(z,temp->expo3);
                temp = temp->next;
        }
        printf("\nSum = %d",sum);
}

main()
{
 node *h1,*h2,*h3;
 clrscr();
 h1 = (node*) malloc(sizeof(node));
 h1->next = h1;
 h2 = (node*) malloc(sizeof(node));
 h2->next = h2;
 h3 = (node*) malloc(sizeof(node));
 h3->next = h3;
 printf("\nenter the first poly");
 read_poly(h1);
 printf("\nenter the second poly");
 read_poly(h2);
 add_poly(h1,h2,h3);
 printf("\nTHE FIRST POLY IS\n");
 display(h1);
 printf("\nTHE SEC POLY IS\n");
 display(h2);
 printf("\nADDition of TWO poly are\n");
 display(h3);
 evaluate(h3);
 getch();
}
```

5. Write function for inserting a new node before the key element in DLL*/

```
void ins_node_bef_Key()
{
        node *temp=start,*new1;
        int key;
```

```
        if(start == NULL)
        {
                printf("Insertion is not possible\n");
                return;
        }
        printf("Enter the key\n");
        scanf("%d",&key);
        if(start->info==key)
        {
                insert_front();
                return;
        }
        while(temp!=NULL && temp->info!=key)
                temp=temp->rptr;

        if(temp==NULL)
        {
                printf("Key not found\n");
                return;
        }
        new1=getnode();
        (temp->lptr)->rptr=new1;
        new1->lptr=temp->lptr;
        new1->rptr=temp;
        temp->lptr=new1;
}
```

6. Write function to delete the key element in DLL*/

```
void del_key()
{
        node *temp=start;
        int key;
        if(start == NULL)
        {
                printf("Deletion not possible\n");
                return;
        }
        printf("Enter the key to be deleted\n");
        scanf("%d",&key);
        if(start->info == key)
        {
                printf("\n deleted element is %d ", start->info);
                start=start->rptr;
```

```
            free(temp);
            return;
        }

        while(temp!=NULL&&temp->info!=key)
            temp=temp->rptr;
        if(temp ==NULL)
        {
            printf("Key not found");
            return;
        }
        (temp->lptr)->rptr=temp->rptr;
        (temp->rptr)->lptr=temp->lptr;
        printf("The deleted element is %d",temp->info);
        free(temp);
        return;
}
```

7. **Write function for concatenation of two lists**

Assuming two linked lists are created with start1 pointing to 1st list and start2 pointing to 2nd list

```
node * concat(node *start1, node *start2)
{
     node  *temp = start1;
      if( start1 == NULL)
        return (start2);
     if(start2 == NULL)
        return(start1);
     if(start1 == NULL  && start2 == NULL)
     {
        Printf("list is empty");
        return;
      }
      While (temp -> next != NULL)
        temp = temp -> next;

    temp -> next = start2;
    return (start1);
}
```

8. Function to reverse SLL

```
void reverse()
{
        node *temp, *prev = NULL;
        while(start != NULL)
        {
                temp = start;
                start = start -> next'
                temp -> next = prev;
                prev = temp;
        }
        start = temp;
}
```

9. Function to display nodes in reverse order in DLL

```
void display()
{
        node *temp = start;
        while(temp -> rptr != NULL)
                temp = temp -> rptr;
         while(temp != NULL)
        {
                printf("%d",temp->info);
                temp = temp -> lptr;
        }
}
```

10. Write function to delete all nodes in SLL & DLL

```
void del_all()
{
        node *temp = start;
        if(start == NULL)
        {
                printf("list is empty");
```

```
            return;
        }
        while(temp != NULL)
        {
            start = start -> next;
            printf(" %d deleted from the list ",temp -> info);
            free(temp);
            temp = start;
        }
}
```

11. Write a program to implement queue using Singly linked list:

Method I

You should make use of Insert_front(), Delete_End() and Display() functions

Method II

Insert_End(), Delete_front and Display()

12. Write a program to implement stack using Singly linked list:

Method I

You should make use of Insert_front(), Delete_Front() and Display() functions

Method II

Insert_End(), Delete_End() and Display()

13. To find the length of the list

```
void length()
{
   node  *temp = start;
   while(temp!=NULL)
    {
     temp=temp->next;
       cnt++;
    }
    printf(" length of list are %d", cnt);
}
```

14. To display odd and even nodes in the list along with its count

```
void odd_even_inlist()
```

```
{
     node  *temp = start;
    printf(“ even nodes are \n”);
     while(temp!=NULL)
     {
       if( temp -> info % 2 == 0)
       {
            printf(“   %d   “, temp -> info);
          even++;
        }
       temp = temp -> next;
      }
   temp = start;
    printf(“ odd nodes are \n”);
     while(temp!=NULL)
     {
       if( temp -> info % 2 != 0)
       {
            printf(“   %d   “, temp -> info);
          odd++;
        }
       temp = temp -> next;
    }
     printf(“ Even and odd num are  %d & %d  “, even,odd);
 }
```


15. To search for a given node in the SLL and display the status

```
void search_key()
{
        int key;
        node *new1,*prev,*temp = start;
        if(start == NULL)
         {
           printf("empty ");
           return;
         }
```

```
        printf("enter the key");
        scanf("%d",&key);
        while(temp != NULL && temp->info != key)
         {
          prev = temp;
          temp = temp->next;
         }

         if(temp == NULL)
         {
          printf("key not found");
          return;
         }
         printf(" key found");
}
```

16. write a function to insert a node before key element in SLL

```
void  ins_node_before_key()
{
        int key;
        node *new1,*prev,*temp = start;
        if(start == NULL)
         {
           printf("insertion not possible ");
           return;
         }

        printf("enter the key");
        scanf("%d",&key);
        while(temp != NULL && temp->info != key)
         {
          prev = temp;
          temp = temp->next;
         }

         if(temp == NULL)
```

```
        {
          printf("key not found");
          return;
        }

        new1 = getnode();
        if(start->info == key)
        {
           new1->next = start;
           start = new1;
           return;
        }

        new1->next = temp;
        prev->next = new1;

}
```

17. write a function to delete a key node in SLL

```
void del_key()
{
        int key;
        node *new1,*prev,*temp = start;

        if(start == NULL)
        {
           printf("list empty ");
           return;
        }
        printf("enter the key to be deleted");
        scanf("%d",&key);
        while(temp != NULL && temp->info != key)
        {
          prev = temp;
          temp = temp->next;
        }
```

```
        if(temp == NULL)
        {
          printf("key not found");
          return;
        }
        if(start->info == key )
        {
           start=start->next;
           free(temp);
           return;
        }

        if(temp->info == key && temp->next == NULL)
        {
           free(temp);
           prev->next=NULL;
           return;
        }

        prev->next = temp->next;
        free(temp);

}
```

18. Create SLL of integers and write C functions to perform the following

a. Create a node list with data 10,20 and 30
b. Insert a node with value 15 in between 10 and 20
c. Delete the node whose data is 20
d. Display the resulting SLL

```
struct sll
{
        int info;
        struct sll *next;
};
typedef struct sll node;
node *start = NULL;
```

```
void ins_front()
{
  node *new1;
  new1=getnode();
  new1->next = start;
  start=new1;
}

void ins_key_bef20()
{
        int key;
        node *new1,*prev,*temp = start;
        if(start == NULL)
         {
           printf("insertion not possible ");
           return;
         }
        while(temp != NULL && temp->info != 20)
         {
          prev = temp;
          temp = temp->next;
         }
         new1 = getnode(); /* 15 is stored in new1->info*/
         new1->next = temp;
         prev->next = new1;

}

void del_key20()
{
        int key;
        node *new1,*prev,*temp = start;

        if(start == NULL)
         {
           printf("list empty ");
           return;
```

```
        }
        while(temp != NULL && temp->info != 20)
        {
          prev = temp;
          temp = temp->next;
        }
        prev->next = temp->next;
        free(temp);
}

void display()
{
        node *temp=start;
        if(start == NULL)
        {
                printf("The sll is empty");
                return;
        }
        printf("The contents of sll are \n");
        while(temp!=NULL)
        {
         printf("%d  \n",temp->info);
         temp=temp->next;
        }
}

int main()
{
        int choice,n,i;
        clrscr();
           while(1)
           {
                printf("1. insert begin\t 2.insert key\t 3. Del key\t 4. Display \t 5.
                Exit\n");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1: for(i=0;i<3;i++)
```

```
                        ins_front(); /* this function is called 3 times with values
                                        entered first entered as 30 then  20 and
                                        last 10*/
                        break;
                case 2:ins_key_bef20();break;
                case 3:del_key20();break;
                case 4:display();break;
                case 5:printf("Exiting ... \n");exit(1);
                                break;
                default : printf("Invalid choice\n");
            }
        }
}
```
