1. **SLL search and delete end:**
2. **SLL ordered insertion and alternate ele:**

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

typedef struct node
{
        int data;
        struct  node *next;
}node;

int find_key(int key,node **head)
{
        node *ptr;
        ptr=*head;
        if(ptr==NULL)
                return 0;
        else
        {
                while(ptr!=NULL)
                {
                        if(ptr->data==key)
                                return 1;
                        ptr=ptr->next;
                }
                return 0;
        }
}

void del_end(node **head)
{
        node *temp,*prev;
        int x;
        if((*head)==NULL)
        {
                printf("List empty\n");
                return;
        }
        else if((*head)->next==NULL)
        {
                temp=*head;
                *head=NULL;
                free(temp);
        }
```

```
        else
        {
                temp=*head;
                while(temp->next!=NULL)
                {
                        prev=temp;
                        temp=temp->next;
                }
                prev->next=NULL;
                free(temp);
        }
}

void in_front(int x,node **head)
{
        node *temp;
        temp=(node*)malloc(sizeof(node));
        temp->data=x;
        temp->next=NULL;
        if((*head)==NULL)
        {
                *head=temp;
        }
        else
        {
                temp->next=*head;
                *head=temp;
        }
}

void in_order(int x,node **head)
{
        node *temp,*ptr,*prev;
        ptr=(node*)malloc(sizeof(node));
        ptr->data=x;
        ptr->next=NULL;
        if((*head)==NULL)
                *head=ptr;
        else if(ptr->data<=(*head)->data)
        {
                ptr->next=*head;
                (*head)=ptr;
        }
        else
```

```
        {
                temp=*head;
                while((temp!=NULL)&&(temp->data>ptr->data))
                {
                        prev=temp;
                        temp=temp->next;
                }
                if(temp->data>ptr->data)
                {
                        prev->next=ptr;
                        ptr->next=temp;
                }
                else
                        temp->next=ptr;
        }
}

void alt(node *head)
{
        node *ptr;
        if(head==NULL)
        {
                printf("List empty\n");
                return;
        }
        else
        {
                ptr=head;
                while(ptr!=NULL)
                {
                        printf("%d->",ptr->data);
                        ptr=ptr->next->next;
                }
                printf("\n");
        }
}

void display(node* head)
{
        node *temp;
        if(head==NULL)
                printf("Empty List\n ");
        else {
                temp=head;
```

```
                while(temp!=NULL)
                {
                        printf("%d->", temp->data);
                        temp=temp->next;
                }
                printf("\n");
        }
}

void main()
{
        int choice,x;
        node *ptr,*head;
        head=NULL;
        while(1)
        {
                printf("1. Search node\n 2.Delete end\n3.Insert
beginning\n4.Ordered insertion\n5.Alternate nodes\n");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1:{
                        printf("Search a key element:");
                        scanf("%d", &x);
                        x=find_key(x, &head);
                        if (x) printf("successful search");
                        else  printf("unsuccessful search");
                        break;
                }

                        case 2:{

                                del_end(&head);
                                break;
                        }
                        case 3:{
                                printf("Enter the element");
                        scanf("%d", &x);
                        in_front(x, &head);
                        break;
                        }
                        case 4:{
                                printf("Ordered Insertion\n Enter the element");
                        scanf("%d", &x);
```

```
                        in_order(x, &head);
                        break;
                        }
                        case 5:{
                                alt(head);
                                break;
                        }
                }
                display(head);
        }
}
```

3. **DLL pos insertion and pos deletion:**

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

typedef struct node
{
        int data;
        struct node *prev,*next;
}Node;

void insert_head(struct node **p, int x)
  {
    struct node *temp;
   //create a node

   temp=(struct node*)malloc(sizeof(struct node));
   temp->data=x;
   temp->prev=temp->next=NULL;

  //if this is the first node

   if(*p==NULL)
      *p=temp;//make first point to temp
    else
    {
      temp->next=*p;//link the new node to first node
      (*p)->prev=temp;
      *p=temp;//make  first point to new node
     }
  }
```

```
void insert_pos(Node **p,int x,int pos)
{
	Node *temp,*q;
	temp=(Node*)malloc(sizeof(Node));
	temp->data=x;
	temp->next=temp->prev=NULL;

	q=(*p);
	int i=1;

	while((q->next!=NULL)&&(i<pos))
	{
		i++;q=q->next;
	}

	if(q->next!=NULL)
	{
		if(q->prev==NULL)
		{
			q->prev=temp;
			temp->next=q;
			(*p)=temp;
		}
		else
		{
			q->prev->next=temp;
			temp->prev=q->prev;
			temp->next=q;
			q->prev=temp;
		}
	}
	else
	{
		if(i==pos)
		{
			q->prev->next=temp;
			temp->prev=q->prev;
			temp->next=q;
			q->prev=temp;
		}
		else if(i=pos-1)
		{
			q->next=temp;
			temp->prev=q;
```

```
                }
                else
                {
                        printf("Invalid pos\n");
                }
        }
}

void delete_pos(Node **p,int pos)
{
        Node *q;
        q=(*p);
        int i=1;
        while((q->next!=NULL)&&(i<pos))
        {
                i++;
                q=q->next;
        }
        if(q->next!=NULL)
        {
                if(q->prev==NULL && q->next==NULL)
                        (*p)=NULL;
                else if(q->prev==NULL)
                {
                        q->next->prev=NULL;
                        (*p)=q->next;
                }
                else if(q->next==NULL)
                {
                        q->prev->next==NULL;
                }
                else
                {
                        q->prev->next=q->next;
                        q->next->prev=q->prev;
                }
                free(q);
        }
        else
        {
                printf("Invalid position\n");
        }

}
```

```
void display(Node *p)
  {
    if(p==NULL)
      printf("\nEmpty List..\n");
    else
     {
       while(p!=NULL)
       {
         printf("%d<->",p->data);
         p=p->next;
       }
       printf("\n");
     }
  }


void main()
{
          Node *first;
          first=NULL;
          int x,ch,pos;
          while(1)
          {
                    display(first);
   printf("\n1..Insert Head..\n");
   printf("2..Insert Tail..\n");
   printf("3..Delete First..\n");
   printf("4..Delete Last..\n");
   printf("5..Delete node..\n");
   printf("6..Delete at position\n");
   printf("7..Insert at position\n");
   scanf("%d",&ch);

   switch(ch)
   {
         case 1: printf("Enter the value..\n");
           scanf("%d",&x);
           insert_head(&first,x);
           break;
     case 6: printf("Enter the position..\n");
           scanf("%d",&x);
           delete_pos(&first,x);
           break;
```

```
        case 7: printf("Enter the value and position..\n");
                scanf("%d%d",&x,&pos);
                insert_pos(&first,x,pos);
                break;

    }

            }
}
```

4. **Josephus:**

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

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

// To create a new node of circular
// linked list
Node *newNode(int data)
{
  Node *temp = (Node*)malloc(sizeof(Node));
  temp->next = temp;
  temp->data = data;
}

int jose(int n,int m)
{
        Node *head=newNode(1);
        Node *prev=head;
        int i;
        for(i=2;i<=n;i++)
        {
                prev->next=newNode(i);
                prev=prev->next;
        }
        prev->next=head;
        Node *ptr1,*ptr2;
        ptr1=ptr2=head;
        while(ptr1->next!=ptr1)
        {
```

```
                int count=1;
                while(count!=m)
                {
                        ptr2=ptr1;
                        ptr1=ptr1->next;
                        count++;
                }

                ptr2->next=ptr1->next;
                ptr1=ptr2->next;
        }
        return ptr1->data;
}

void main()
{
        int m,n;
        printf("Number of people and number of skipped:\n");
        scanf("%d%d",&n,&m);
        int j=jose(n,m);
        printf("Last person,%d",j);
}
```

5. A) Infix to postfix (Stacks)

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

int f(char sym)
{
        switch(sym)
        {
                case '+':
                case '-':return 2;
                case '*':
                case '/':return 4;
                case '(':return 0;
                case '#':return -1;
                default:return 8;
        }
}

int g(char sym)
{
```

```
	switch(sym)
	{
		case '+':
		case '-':return 1;
		case '*':
		case '/':return 3;
		case '(':return 9;
		case ')':return 0;
		default:return 7;
	}
}

void intopost(char infi[],char post[])
{
	int top=-1,i,j=0;
	char symbol;
	char s[50];
	s[++top]='#';
	for(i=0;i<strlen(infi);i++)
	{
		symbol=infi[i];
		while(f(s[top])>g(symbol))
		{
			post[j]=s[top--];
			j++;
		}
		if(f(s[top])!=g(symbol))
		{
			s[++top]=symbol;
		}
		else
			top--;
	}
	while(s[top]!='#')
	{
		post[j]=s[top--];
		j++;
	}
	post[j]='\0';
}


int main()
{
```

```
        char infix[20];
        char postfix[20];
        printf("enter the string\n");
        scanf("%s", infix);
        intopost(infix, postfix);
        printf("Postfix expression is \n");
        printf("%s", postfix);
    }
```

b) Fibbo series:

```
#include<stdio.h>


int fibbo(int n)
{
    if(n==0)
        return 0;
    else if(n==1)
        return 1;
    else
        return fibbo(n-1)+fibbo(n-2);
}


int main()
{
    int n;
    printf("Enter number n:");
    scanf("%d",&n);
    printf("%dth fibo is:%d",n,fibbo(n));
}
```

6. **A)Postfix eval:**

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

```
void push(int *s,int *t,int x)
{
	++(*t);
	s[*t]=x;
}

int pop(int *s,int *t)
{
	return s[(*t)--];
}

int isoperator(char ch)
{
	switch(ch)
	{
		case '+':
		case '-':
		case '*':
		case '/': return 1;
	}
	return 0;
}

int postfixeval(char *postfix)
{
	int s[20];
	int top=-1;
	int i=0;
	int r;
	while(postfix[i]!='\0')
	{
		char ch=postfix[i];
		if(isoperator(ch))
		{
			int op1=pop(s,&top);
			int op2=pop(s,&top);
			switch(ch)
			{
				case '+':r=op1+op2;
					push(s,&top,r);
					break;
				case '-':r=op1-op2;
					push(s,&top,r);
					break;
```

```
                                case '*':r=op1*op2;
                                        push(s,&top,r);
                                        break;
                                case '/':r=op1/op2;
                                        push(s,&top,r);
                                        break;
                        }
                }
                else push(s,&top,ch-48);
                i++;
        }
        return pop(s,&top);
}

void main()
{
        char postfix[20];
        printf("Enter postfix:");
        scanf("%s",postfix);
        int r=postfixeval(postfix);
        printf("Result is %d",r);
}
```

B) TOH:

```
#include<stdio.h>

int count;

void toh(int n,int A,int B,int C)
{
        if(n==1)
        {
                count=count+1;
                printf("Move disc from %d to %d..\n",A,C);
        }
        else
        {
                toh(n-1,A,C,B);
                toh(1,A,B,C);
                toh(n-1,B,A,C);
        }
}

int main()
```

```
{
        int n;
        printf("Enter number of discs:");
        scanf("%d",&n);
        int A=1,B=2,C=3;
        toh(n,A,B,C);
        printf("Total Count = %d\n", count);
   return 0;
}
```

7. Enqueue and dequeue:

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

typedef struct node
{
        int data;
        struct node *link;
}Node;

void display(Node *f,Node *r)
{
        if(f==NULL)
                printf("EMPTY LIST\n");
        else
        {
                while(f!=r)
                {
                        printf("%d->",f->data);
                        f=f->link;
                }
                printf("%d\n",f->data);
        }
}

void enq(int x,Node **f,Node**r)
{
        Node *temp;
        temp=(Node*)malloc(sizeof(Node));
        temp->data=x;
        temp->link=NULL;

        if(*f==NULL)
                *f=*r=temp;
```

```
        else
        {
                (*r)->link=temp;
                *r=temp;
        }
}

void deq(Node **f,Node **r)
{
        Node *q=*f;
        if(*f==NULL)
                printf("Empty queue\n");
        else
        {
                if(*f==*r)
                        *f=*r=NULL;
                else
                {
                        *f=q->link;
                }
                free(q);
        }
}

int main()
{
        int x, ch, k;
        Node *front, *rear;
        front=rear=NULL;
        while(1)
        {
                display(front,rear);
                printf("1:insert from rear\n 2: delete from front\n 3: exit\n");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1: printf("enter the element\n");
                                scanf("%d",&x);
                                enq(x,&front,&rear);
                                break;

                        case 2: deq(&front,&rear);
                                break;
```

```
                        case 3: exit(24);
                }

        }
}
```

8. BT array traversal:

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

void create(int *t,int x)
{
        int num;
        printf("Enter the value:");
        scanf("%d",&num);
        if(num==-1) return;
        t[x]=num;
        printf("LST of %d",t[x]);
        create(t,2*x);
        printf("RST of %d",t[x]);
        create(t,2*x+1);

        return;
}

void preorder(int *t,int x)
{
        if(t[x]!=-1)
        {
                printf("%d ",t[x]);
                preorder(t,2*x);
                preorder(t,2*x+1);
        }
}

void inorder(int *t,int x)
{
        if(t[x]!=-1)
        {
                inorder(t,2*x);
                printf("%d ",t[x]);
                inorder(t,2*x+1);
        }
}
```

```
void postorder(int *t,int x)
{
	if(t[x]!=-1)
	{
		postorder(t,2*x);
		postorder(t,2*x+1);
		printf("%d ",t[x]);
	}
}


int main()
{
	int i,t[100];
	for(i=0;i<100;i++)
		t[i]=-1;
	create(t,1);
	printf("Preorder\n");
	preorder(t,1);
	printf("Inorder\n");
	inorder(t,1);
	printf("Postorder\n");
	postorder(t,1);
}
```

9. BT LL traversal, leafcounts, height:

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

typedef struct node
{
	int data;
	struct node *left,*right;
}node;

node *create()
{
	node *p;
	int x;
	printf("Enter value:");
	scanf("%d",&x);
	if(x==-1)
		return NULL;
```

```
        p=(node*)malloc(sizeof(node));
        p->data=x;
        printf("LST of %d\n",x);
        p->left=create();
        printf("RST of %d\n",x);
        p->right=create();
        return p;
}

void preorder(node *t)
{
        if(t!=NULL)
        {
                printf("%d ",t->data);
                preorder(t->left);
                preorder(t->right);
        }
}

void inorder(node *t)
{
        if(t!=NULL)
        {
                inorder(t->left);
                printf("%d ",t->data);
                inorder(t->right);
        }
}

void postorder(node *t)
{
        if(t!=NULL)
        {
                postorder(t->left);
                postorder(t->right);
                printf("%d ",t->data);
        }
}

int leafcount(node *t)
{
        int l,r;
        if(t!=NULL)
        {
```

```
        if(t->left==NULL && t->right==NULL)
                return 1;
        l=leafcount(t->left);
        r=leafcount(t->right);
        return l+r;
        }
        return 0;
}

int height(node *t)
{
        int l,r;
        if(t!=NULL)
        {
                if(t->left==NULL && t->right==NULL)
                        return 0;
                l=height(t->left);
                r=height(t->right);
                if(l>r)
                        return l+1;
                return r+1;
        }
        return -1;
}

int main()
{
        node *root;
        int leaves,h;
        root=create();
        printf("Preorder\n");
        preorder(root);
        printf("\n");
        printf("Inorder\n");
        inorder(root);
        printf("\n");
        printf("Postorder\n");
        postorder(root);
        printf("\n");
        leaves=leafcount(root);
        printf("Leaves:%d\n",leaves);
        h=height(root);
        printf("Height:%d\n",h);
}
```

10. BFS ,DFS and number of components:

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

int n;
int visitb[100];
int visitd[100];
int a[100][100];
int q[100];
int f,r;

void creategraph()
{
        int i,j;
        printf("Enter source and destination...\n");
        scanf("%d%d",&i,&j);
        if((i==0)&&(j==0))break;
        a[i][j]=a[j][i]=1;
}

void insert(int v)
{
        r++;
        q[r]=v;
        if(f==-1)
                f=0;
}


int deletee()
{
        int w;
        w=q[f];
        if(f==r)
                f=r=-1;
        else
                f++;
        return w;
}

int empty()
{
        if(f==-1)
```

```
                return 1;
        return 0;
}

void bfs(int v)
{
        int w;
        visitb[v]=1;
        printf("%d",v);
        insert(v);
        while(!empty())
        {
                v=deletee();
                for(w=1;w<=n;w++)
        {
                if((a[v][w]==1)&&(visitb[w]==0))
                {
                        visitb[w]=1;
                        printf("%d",w);
                        insert(w);
                }
        }
        }
}

void dfs(int v)
{
   int w;
   visitb[v]=1;
   printf("%d ",v);
   for(w=1;w<=n;w++)
   {
      if((a[v][w]==1)&&(visitb[w]==0))
                dfs(w);
   }
}

int component()
 {
  int i;
  label=0;
  for(i=1;i<=n;i++)
   visit[i]=0;
```

```
  for(i=1;i<=n;i++)
  {
    if(visit[i]==0)
    {
      ++label;
       dfs(i);
    }
  }
    return label;
 }

int main()
{
        int i,v;
        f=r=-1;
        printf("enter the no of vertex\n");
        scanf("%d",&n);
        creategraph();
        printf("enter the src vertex\n");
        scanf("%d",&v);
        printf("vertex reachable from %d using BFS are...\n", v);
        bfs(v);
        printf("The vertices reachable from %d using DFS are..\n",v);
   dfs(v);
   int result=component();
   printf("The no of components = %d\n",result);
}
```

11.

12. Max heap bottom up:

```
#include<stdio.h>

#define MAX 100


void read_Array(int* array, int n)

{

        int i;

        for(i = 1; i <= n ; ++i)

        {
```

```
                scanf("%d", &array[i]);
        }
}

void display_Array(int* array, int n)
{
        int i;
        for(i = 1; i <= n ; ++i)
        {
                printf("%d ", array[i]);
        }
        printf("\n");
}

void swap(int* a, int* b)
{
        int temp = *a;
        *a = *b;
        *b = temp;
}

void heapify(int* array, int n, int position)
{
        int i;
        int j;
        i = position;
        j = 2 * i;
        int flag = 0;
```

```
        while(j <= n && !flag)
        {
                if(j < n && array[j + 1] > array[j])
                {
                        ++j;
                }
                if(array[i] < array[j])
                {
                        swap(&array[i], &array[j]);
                        i = j;
                        j = 2 * i;
                }
                else
                {
                        flag = 1;
                }
        }
}

void insert(int* array, int n)
{
        int i;
        for(i = n/2; i >= 1; --i)
        {
                heapify(array, n, i);
        }
}
```

```
int main()
{
	int array[MAX];
	int n;
	printf("Enter the number of elements\n");
	scanf("%d", &n);

	read_Array(array, n);
	display_Array(array, n);
	insert(array, n);
	printf("After heapify\n");
	display_Array(array, n);

	return 0;
}
```

13. Priority q using heap(Max heap):

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

typedef struct element{
	int data,pty;
}ele;

void pqinsert(ele *h,ele key,int *count)
{
	int i,j;
	i=*count;
	h[i]=key;
```

```
        ++(*count);
        j=(i-1)/2;
        while((i>0)&&(key.pty>h[j].pty))
        {
                h[i]=h[j];
                i=j;
                j=(i-1)/2;
        }
        h[i]=key;
}

void heapify(ele *h,int n)
{
        ele key;
        int i,j;
        j=0;
        key=h[j];
        i=2*j+1;
        while(i<=n)
        {
                if(i+1<=n)
                {
                        if(h[i+1].pty>h[i].pty)
                                i++;
                }
                if(key.pty<h[i].pty)
                {
                        h[j]=h[i];
                        j=i;
```

```
                        i=2*j+1;
                }
                else
                        break;
        }
        h[j]=key;
}

void display(ele *h,int count)
{
        int i;
        printf("Elements are:\n");
        for(i=0;i<count;i++)
        {
                printf("ELE:%d  PRIORITY:%d\n",h[i].data,h[i].pty);
        }
        printf("\n");
}

ele pqdelete(ele *h,int *count)
{
        ele temp;
        temp=h[0];
        h[0]=h[*count-1];
        --(*count);
        heapify(h,*count-1);
        return temp;
}
```

```
int main()
{
        ele hpq[100],temp;
        int ch,count=0;
        while(1)
        {
                display(hpq,count);
                printf("\n1.Insertion\n2.Deletion\n3.Exit\n");
                scanf("%d",&ch);
                switch(ch)
  {
    case 1:printf("Enter the value and the priority..");
          scanf("%d %d",&temp.data,&temp.pty);
          pqinsert(hpq,temp,&count);
           break;
   case 2:temp=pqdelete(hpq,&count);
          printf("Element = %d Priority= %d\n",temp.data, temp.pty);
          break;
    case 3:exit(0);
  }
        }
}
```

14. Threaded BST preorder traversal:

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

struct Node
{
  struct Node *left, *right;
```

```
  int info;
  int lthread;
  int rthread;
};
typedef struct Node Node;

Node *insert(Node *root,int key)
{
        Node *cur;
        Node *prev;
        cur=root;
        prev=NULL;
        while(cur!=NULL)
        {
                if(key==(cur->info))
                        return root;
                prev=cur;
                if(key < cur->info)
                {
                        if(!cur->lthread)
                                cur=cur->left;
                        else
                                break;
                }
                else
                {
                        if(!cur->rthread)
                                cur=cur->right;
                        else
```

```
                                        break;
                        }
                }
                Node *tmp = (Node *)malloc(sizeof(struct Node));
    tmp -> info = key;
    tmp -> lthread = 1;
    tmp -> rthread = 1;
    if(prev==NULL)
    {
                root=tmp;
                tmp->left=NULL;
                tmp->right=NULL;
                }
                else if(key<(prev->info))
                {
                        tmp->left=prev->left;
                        tmp->right=prev;
                        prev->lthread=0;
                        prev->left=tmp;
                }
                else
                {
                        tmp->right=prev->right;
                        tmp->left=prev;
                        prev->rthread=0;
                        prev->right=tmp;
                }
                return root;
}
```

```
void preorder(Node* root)
{
   Node* curr = root;

   while(curr!=NULL)
   {
      printf("%d ",curr->info);

      if(curr->left!=NULL)
         curr=curr->left;

      else if(curr->rthread==1)
         curr=curr->right;

      else
      {
         while(curr->right!=NULL && curr->rthread==0)     //right thread exists
            curr=curr->right;

         if(curr->right == NULL)                    //last node
            break;
         else
            curr=curr->right;

      }
   }
}
```

15. BT using infix. Evaluate expression and traverse using postorder:

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

struct tnode
{
  char data;
  struct tnode *left;
  struct tnode *right;
};

typedef struct tnode tnode;

int evaluate(tnode *t)
 {
   int x;

   switch(t->data)
   {
     case '+': return(evaluate(t->left)+evaluate(t->right));
     case '-': return(evaluate(t->left)-evaluate(t->right));
     case '*': return(evaluate(t->left)*evaluate(t->right));
     case '/': return(evaluate(t->left)/evaluate(t->right));
     default:printf("%c=",t->data);
     scanf("%d",&x);
     return x;
   }
}

int isoper(char ch)
```

```
 {
   switch(ch)
   {
     case '+':
     case '-':
     case '*':
     case '/':return 1;
     default: return 0;
   }
 }

void push(tnode **stk, int *t, tnode* temp)
 {
   ++*t;
   stk[*t]=temp;
 }

tnode *pop(tnode **stk, int *t)
 {
   struct tnode *temp;
    temp=stk[*t];
    --*t;
   return temp;
 }

void postorder(tnode *t)
 {
   if(t!=NULL)
   {
```

```
    postorder(t->left);
    postorder(t->right);
    printf("%c ",t->data);
  }
 }

void inorder(tnode *t)
 {
   if(t!=NULL)
   {
    inorder(t->left);
    printf("%c ",t->data);
    inorder(t->right);
  }
 }



tnode* create_exptree(char *exp)
{
        tnode *temp;
        tnode *stk[100];
        int top=-1,i=0;
        while(exp[i]!='\0')
        {
                char ch=exp[i];
                temp=(tnode*)malloc(sizeof(tnode));
                temp->data=ch;
                temp->left=temp->right=NULL;
```

```
                    if(isoper(ch))//if operator
    {
      temp->right=pop(stk,&top);
      temp->left=pop(stk,&top);
      push(stk,&top,temp);
     }
   else
     push(stk,&top,temp);//operand
    i++;
         }
         return pop(stk,&top);
}

void main()
{
 struct tnode *root;
 int ch,num,k;
 char exp[100];
 root=NULL;

 printf("Enter the postfix form of the expression...");
 scanf("%s",exp);

 root=create_exptree(exp);
 printf("The infix expression = \n");
 inorder(root);
 printf("\nThe postfix expression = \n");
 postorder(root);
```

```
  printf("\nEvaluating the expression\n");
  int result = evaluate(root);
  printf("\nThe evaluation of the expression = %d\n",result);

}
```

16. Tries: a) Search word b)delete:

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

#define ARRAY_SIZE(a) sizeof(a)/sizeof(a[0])

// Alphabet size (# of symbols)
#define ALPHABET_SIZE (26)

// Converts key current character into index
// use only 'a' through 'z' and lower case
#define CHAR_TO_INDEX(c) ((int)c - (int)'a')

// trie node
struct TrieNode
{
struct TrieNode *children[ALPHABET_SIZE];

// isEndOfWord is true if the node represents
// end of a word
bool isEndOfWord;
```

```
};

bool isLeafNode(struct TrieNode* root)
{
   return root->isEndOfWord != false;
}


// Returns new trie node (initialized to NULLs)
struct TrieNode *getNode(void)
{
struct TrieNode *pNode = NULL;

pNode = (struct TrieNode *)malloc(sizeof(struct TrieNode));

if (pNode)
{
int i;

pNode->isEndOfWord = false;

for (i = 0; i < ALPHABET_SIZE; i++)
pNode->children[i] = NULL;
}

return pNode;
}

// If not present, inserts key into trie
```

```
// If the key is prefix of trie node, just marks leaf node
void insert(struct TrieNode *root, const char *key)
{
int level;
int length = strlen(key);
int index;

struct TrieNode *pCrawl = root;

for (level = 0; level < length; level++)
{
index = CHAR_TO_INDEX(key[level]);
if (!pCrawl->children[index])
pCrawl->children[index] = getNode();

pCrawl = pCrawl->children[index];
}

// mark last node as leaf
pCrawl->isEndOfWord = true;
}

// Returns true if key presents in trie, else false
bool search(struct TrieNode *root, const char *key)
{
int level;
int length = strlen(key);
int index;
struct TrieNode *pCrawl = root;
```

```
for (level = 0; level < length; level++)
{
index = CHAR_TO_INDEX(key[level]);

if (!pCrawl->children[index])
return false;

pCrawl = pCrawl->children[index];
}

return (pCrawl != NULL && pCrawl->isEndOfWord);
}



bool isEmpty(struct TrieNode* root)
{
           int i;
   for (i = 0; i < ALPHABET_SIZE; i++)
      if (root->children[i])
         return false;
   return true;
}

void display(struct TrieNode* root, char str[], int level)
{

   if (isLeafNode(root))
   {
```

```
        str[level] = '\0';
        printf("%s\n",str);
    }

    int i;
    for (i = 0; i < ALPHABET_SIZE; i++)
    {
        if (root->children[i])
        {
            str[level] = i + 'a';
            display(root->children[i], str, level + 1);
        }
    }
}

struct TrieNode* delete(struct TrieNode* root, char* key, int depth)
{


    if (!root)
        return NULL;


    if (depth == strlen(key)) {


        if (root->isEndOfWord)
            root->isEndOfWord = false;
```

```
        if (isEmpty(root)) {
            free(root);
            root = NULL;
        }

        return root;
    }


    int index = key[depth] - 'a';
    root->children[index] =
         delete(root->children[index], key, depth + 1);


    if (isEmpty(root) && root->isEndOfWord == false) {
        free(root);
        root = NULL;
    }

    return root;
}


int main()
{


char output[][32] = {"Not present in trie", "Present in trie"};
```

```
struct TrieNode *root = getNode();
char a[20], s[20], d[20];
int f=1;


while(f)
{
printf("Enter the option\n");
printf("1. Insert\n");
printf("2. Search\n");
printf("3. Delete\n");
printf("4. Display\n");
int c;
scanf("%d", &c);
switch(c)
{
case 1:scanf("%s",a);insert(root,a);break;
case 2:scanf("%s",s);printf("%s --- %s\n", s, output[search(root, s)] ); break;
case 3:scanf("%s",d);delete(root, d, 0);break;
case 4:display(root,a,0);break;
default: f=0;
}
}

return 0;
}
```

17. Hashing to avoid collision separate chaining

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

typedef struct
{
   int *table;
   int size;
} HashTable;



HashTable *create_table(int size);
int search(HashTable *htable, int element);
void insert(HashTable *htable, int element);
void delete (HashTable *htable, int element);
void display_table(HashTable *htable);
void destroy_table(HashTable *htable);

int main()
{
   int size, choice, loop = 1;
   int element, find;
   scanf("%d", &size);
   HashTable *htable = create_table(size);
   while (loop)
   {
      scanf("%d", &choice);
      switch (choice)
```

```
{
case 1:
   // Insert element
   scanf("%d", &element);
   insert(htable, element);
   break;

case 2:
   // Delete element
   scanf("%d", &element);
   delete (htable, element);
   break;

case 3:
   // Search element
   scanf("%d", &element);
   find = search(htable, element);
   if (find)
      printf("YES\n");
   else
      printf("NO\n");
   break;

case 4:
   // Print all elements in the hash table
   display_table(htable);
   break;

default:
```

```
            // Destroy tree and exit the loop
            destroy_table(htable);
            loop = 0;
            break;
        }
    }
}

HashTable *create_table(int size)
{
    HashTable *a = (HashTable*)malloc(sizeof(HashTable));
    a->table = (int*)malloc(sizeof(int)*size);
    a->size = size;
    int i;
    for(i=0;i<size;i++)
    {
        a->table[i] = -1;
    }
    return a;
}

void insert(HashTable *htable, int element)
{
    int index = element%(htable->size);
    if(htable->table[index]==-1)
    {
        htable->table[index] = element;
    }
    else
```

```
    {
        int flag = 0;
        int i;
        for(i = index+1;i<htable->size;i++)
        {
            if(htable->table[i]==-1)
            {
                htable->table[i] = element;
                flag = 1;
                break;
            }

        }

        if(flag==0)
        {
             int j;
            for(j = 0;j<index;j++)
            {
                if(htable->table[j]==-1)
                {
                    htable->table[j] = element;
                    break;
                }
            }
        }
    }
}
```

```
int search(HashTable *htable, int element)
{
   int index = element%(htable->size);
   if(htable->table[index]==element)
   {
      return 1;
   }
   else
   {
      int i = index+1;
      while(htable->table[i]!=-1 && i<htable->size)
      {
         if(htable->table[i]==element)
         {
            return 1;
         }
         i++;
      }
      if(htable->table[i]==-1)
      {
         return 0;
      }
      if(i==htable->size)
      {
         int j = 0;
         while(htable->table[j]!=-1 && j<index)
         {
            if(htable->table[j]==element)
            {
```

```
                return 1;
            }
            j++;
        }
        return 0;
    }



  }
}

void delete (HashTable *htable, int element)
{
  int pres = search(htable,element);
  if(pres)
  {
    int index = element%(htable->size);
  if(htable->table[index]==element)
  {
    htable->table[index] = -1;
    return;
  }
  else
  {
    int i = index+1;
    while(htable->table[i]!=-1 && i<htable->size)
    {
      if(htable->table[i]==element)
      {
```

```
            htable->table[i] = -1;
            return;
          }
          i++;
        }

        if(i==htable->size)
        {
          int j = 0;
          while(htable->table[j]!=-1 && j<index)
          {
            if(htable->table[j]==element)
            {
              htable->table[j] = -1;
              return;
            }
            j++;
          }
        }
    }
    }
}

void display_table(HashTable *htable)
{
           int i;
    for(i = 0;i<htable->size;i++)
    {
        printf("%d ",htable->table[i]);
```

```
    }
    printf("\n");
}

void destroy_table(HashTable *htable)
{
    free(htable->table);
}
```