```
/*1.Array operations        */

#include<stdio.h>
#define maxsize 10
int a[maxsize],n,i,ele,pos;
int create()
{
printf("enter the no of elements\n");
scanf("%d",&n);
printf("enter %d elements",n);
for(i=0;i<n;i++)
scanf("%d",&a[i]);
}
int display()
{
        if(n==0)
        {
        printf("array is empty");
        return;
        }
        else
        {
        printf("array elements are\n");
        for(i=0;i<n;i++)
        printf("%d \t",a[i]);
        }
}
int insert()
{
        if (n==maxsize)
        {
        printf("array is full");
        return;
        }
        else
        {
        printf("enter the element to be inserted");
        scanf("%d",&ele);
        printf("enter the valid positon");
        scanf("%d",&pos);
        for(i=n-1;i>=pos-i;i--)
        {
        a[i+1]=a[i];
        }
        a[pos-1]=ele;
        n++;
        }
}
int del()
{
        if(n==0)
        {
        printf("array is empty");
        return;
        }
        else
        {
        printf("enter the position of elements to be deleted\n");
        scanf("%d",&pos);
        ele=a[pos-1];
        for(i=pos-1;i<n-1;i++)
        {
        a[i]=a[i+1];
        }
        n--;
        printf("the deleted elementes is %d",ele);
        }
}

void main()
{
int ch;
while(1)
{
        printf("---------------menu--------------\n");
        printf("1--->create\n");
        printf("2-->display\n");
        printf("3-->insert\n");
        printf("4-->delete\n");
        printf("5--->exit\n");
        printf("enter choice");
        scanf("%d",&ch);
                switch(ch)
                {
                case 1: create();
                break;
                case 2: display();
                break;
                case 3:insert();
                break;
                case 4:del();
                break;
                case 5:exit(0);
                default:printf("enter the valid choice");
                break;
                }
        }
}
```

```
/* 2.String Matching    */
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
void stringmatch(char str[100],char pat[50],char rep[50])
{
int i=0,m=0,c=0,j=0,k=0,flag=0;
char ans[50];
    while(str[c]!='\0')
    {
        if(str[m]==pat[i])
        {
        i++;
        m++;
            if(pat[i]=='\0')
            {
            flag=1;
            for(k=0;rep[k]!='\0';k++,j++)
            ans[j]=rep[k];
            i=0;
            c=m;
            }
```

```
                }
                else
                {
                ans[j]=str[c];
                j++;
                c++;
                m=c;i=0;
                }
        }
ans[j]='\0';
if(flag==1)
printf("\n The resultant string is \n %s",ans);
else
printf("\n Pattern string NOT found \n");
}
void readstring(char str[],char pat[],char rep[])
{
printf("\n Enter a main string \n");
gets(str);
printf("\n Enter a pattern string \n");
gets(pat);
printf("\n Enter a replace string \n");
gets(rep);
}
void main()
{
char str[100],pat[50],rep[50];
readstring(str,pat,rep);
stringmatch(str,pat,rep);
}
```

```
/*3.Stack Operations   */

#include<stdio.h>
#include<stdlib.h>
#define maxsize 4
int push(int s[],int *top)
{
int ele;
        if(*top==(maxsize-1))
        {
```

```
	printf("\n\nstack is overflow");
	return;
	}
	else
	{
	printf("\n enter a element to be pushed :");
	scanf("%d",&ele);
	s[++(*top)]=ele;
	}
}
int pop(int s[],int *top)
{
int ele;
ele=s[(*top)--];
return ele;
}
void palindrome(int v[],int top)
{
int flag=0,i;
	for(i=0;i<(top+1);i++)
	{
	if(v[i]==pop(v,&top))
	flag=1;
		else
		{
		flag=0;
		break;
		}
	}
	if(flag)
printf("stack contents are palindrome");
else
printf("stack contents are not palindrome");
}
 void display(int s[],int top)
{
int i;
	if(top==-1)
	{
	printf("\n stack is empty");
	return;
	}
	else
	{
        printf("\n the stack contents are");
                for(i=top;i>=0;i--)
                {
                printf("\n[%d]",s[i]);
                printf("\n");
                }
        }
}
void main()
{
int s[maxsize],ele;
int ch,top=-1;
        while(1)
        {
        printf("\n----------->MAIN MENU<----------\n");
        printf("\n1---------->PUSH into the stack<------");
        printf("\n2---------->POP from the stack<------");
        printf("\n3---------->PALINDROME check using stack<----");
        printf("\n4----------> DISPLAY<-----------");
        printf("\n5---------->EXIT<------");
        printf("\n enter your choice:");
        scanf("%d",&ch);
                switch(ch)
                {
                case 1:push(s,&top);
                      display(s,top);
                      break;
                case 2:if(top==-1)
                        {
                          printf("\n stack under flow");
                        }
                        else
                        {
                          ele=pop(s,&top);
                          printf("\n popped element is %d",ele);
                        }
                        break;
                case 3:palindrome (s,top);
                        break;
                case 4:display(s,top);
                      break;
                case 5:exit(0);
                      break;
                default:printf("\n enter a valid choice");
```

```
            break;
        }
    }
}
```

```
/*4.Conversion of  infix to postfix expression  */
#include<stdio.h>
#include<string.h>
int f(char);
int g(char);
void infixtopostfix(char in[],char post[])
{
int i=0,j=0,top=-1;
char stk[20],sym;
stk[++ top]='#';
    while(in[i]!='\0')
    {
    sym=in[i++];
    while(f(stk[top])>g(sym))
    post[j++]=stk[top--];
    if(f(stk[top])!=g(sym))
    stk[++top]=sym;
    else
    stk[top--];
    }
    while(stk[top]!='#')
```

```
        {
        post[j++]=stk[top--];
        }
post[j++]='\0';
}
int f(char sym)
{
        switch(sym)
        {
        case '+':
        case '-':return 2;
        case '%':
        case '*':
        case '/':return 4;
        case '$':
        case '^':return 5;
        case '(':return 0;
        case '#':return-1;
        default:return 8;
        }
}
int g(char sym)
{
        switch(sym)
        {
        case '+':
        case '-':return 1;
        case '%':
        case '*':
        case '/':return 3;
        case '$':
        case '^':return 6;
        case '(':return 9;
        case ')':return 0;
        default:return 7;
        }
}
int main()
{
char infix[50],postfix[50];
printf("enter the valid infix expression \n");
scanf("%s",infix);
infixtopostfix(infix,postfix);
printf("the given infix expression is %s and the equivalent postfix espression is
%s",infix,postfix);
return 0;
}
```

```
/* 5.Evaluation of suffix Expression  */
#include<stdio.h>
```

```
#include<stdlib.h>
#include<math.h>
void tower(int n,int source,int temp,int destination)
{
  if(n==0)
   return;
  tower(n-1,source,destination,temp);
  printf("\n move disc %d from %c to %c",n,source,destination);
  tower(n-1,temp,source,destination);
}
void push(float stk[],int *top,float op)
{
  *top=*top+1;
  stk[*top]=op;
}
int pop(float stk[],int *top)
{
  float res;
          if(*top==-1)
            {
              printf("stack is underflow");
              return;
             }
            else
            {
               res=stk[*top];
               *top=*top-1;
               return res;
            }
}
float compute(float op1,float op2,char s)
{
           switch(s)
           {
           case '+' : return (op1+op2);
           case '-' : return (op1-op2);
           case '*' : return (op1*op2);
           case '/' : return (op1/op2);
           case '$' :
           case '^' : return pow(op2,op1);
           }
}
void evalpostfix()
{
  char sym,post[20];
  int i=0,j=0,top=-1;
  float stk[20],item,op1,op2,res;
  printf("enter the valid postfix expn:");
  scanf("%s",post);
            while(post[i]!='\0')
            {
              sym=post[i++];
                        if(isdigit(sym))
                        {
                          item=sym-'0';
                          push(stk,&top,item);
                        }
                        else
                        {
                          op2=pop(stk,&top);
                          op1=pop(stk,&top);
                          res=compute(op1,op2,sym);
                          push(stk,&top,res);
                        }
            }
  res=pop(stk,&top);

 printf("the result of evaluation of postfix expn is %f",res);
}
void main()
{
  int n,ch;
  printf("\n main menu");
           while(1)
          {
          printf("\n 1.evalution of postfix expn");
          printf("\n 2.tower of honoi");
          printf("\n 3.exit");
          printf("\nenter your choice:");
          scanf("%d",&ch);
                    switch(ch)
                    {
                    case 2: printf("\n enter the no. of discs :");
                              scanf("%d",&n);
                              tower(n,'A','B','C');
                              printf("\n total no. of moves are");
                              break;
                    case 1: evalpostfix();
```

```
                        break;
                case 3: exit(0);
                }
        }
}
```

```
/*6.Circular Queue operations   */
#include<stdio.h>
#include<stdlib.h>
#define MAXSIZE 4
void insert(char q[],int *r,int *c)
{
   char item;
    if(*c==MAXSIZE)
        {
        printf("\n queue is full");
        return;
        }
  else
        {
         printf("enter the character to be inserted:");
        scanf(" %c",&item);
        *r=(*r+1)% MAXSIZE;
        q[*r]=item;
        (*c)++;
        }
}
void deleteq(char q[],int *f,int *c)
{
  char item;
  if(*c==0)
        {
        printf("\n queue is empty");
        return;
        }
        item=q[*f];
        printf("\n deleted item is :%c",item);
        *f=(*f+1)%MAXSIZE;
        (*c)--;
}
void display(char q[],int f,int *c)
{
  int i;
            if(*c==0)
        printf("\nqueue is empty");
        else
            {
        printf("\n contents of queue is \n");
        for(i=1;i<=*c;i++)
                  {
                  printf("%c \t",q[f]);
                  f=(f+1)%MAXSIZE;
                  }
           }
}
void main()
{
  int ch,f=0,r=-1,c=0;
  char q[MAXSIZE];
  while(1)
            {
                  printf("\n main menu");
                   printf("\n1.insert");
                  printf("\n2.delete");
                  printf("\n3.display");
                  printf("\n4.exit");
                  printf("\n enter choice:");
                  scanf("%d",&ch);
                  switch(ch)
                            {
                            case 1 :insert(q,&r,&c);
                                                break;
                            case 2 : deleteq(q,&f,&c);
                                      break;
            case 3 : display(q,f,&c);
                break;
            case 4 : exit(0);
            }
    }
}
```

```
/* 7.Silngly Linked List  */
#include<stdio.h>
#include<stdlib.h>
struct studentnode
{
   char  usn[11];
   char  name[30];
   char  branch[5];
   int   sem;
   char  phno[11];
   struct studentnode *link;
};
typedef struct studentnode *NODE;
NODE getnode()
{
  NODE newnode;
  newnode=(NODE)malloc(sizeof(struct studentnode));
  if(newnode==NULL)
    return NULL;
  printf("\nenter usn,name,branch,sem,ph.no\n");
  scanf("%s%s%s",newnode->usn,newnode->name,newnode->branch);
  scanf("%d%s",&newnode->sem,newnode->phno);
  newnode->link=NULL;
  return newnode;
}
void display(NODE first)
{
  NODE cur;
  int count=0;
          if(first==NULL)
           printf("\nempty list-no student data\n");
          else
          {
          cur=first;
          printf("\n \t\t student data\t\n");
          printf("\n USN \t NAME \t BRANCH \t SEM \t PH.NO");
                    while(cur!=NULL)
                    {
                    printf("\n%s \t %s\t %s\t %d\t %s\t",cur->usn,cur->name,cur-
>branch,cur->sem,cur->phno);
                    cur=cur->link;
                    count=count+1;
                    }
          printf("the no. of nodes in the list is %d",count);
          }
}
NODE insertfront(NODE first)
{
 NODE newnode;
 newnode=getnode();
 if(newnode==NULL)
  printf("memory not available");
 else
  newnode->link=first;
  return newnode;
}
NODE insertrear(NODE first)
{
  NODE newnode,cur=first;
  newnode=getnode();
  if(newnode==NULL)
   return newnode;
  while(cur->link!=NULL)
   cur=cur->link;
   cur->link=newnode;
  return first;
}
NODE deletefront(NODE first)
{
  NODE temp;
          if(first==NULL)
          printf("\n list is empty");
          else
          {
          temp=first;
          first=first->link;
          free(temp);
          }
 return first;
}
NODE deleterear(NODE first)
{
 NODE cur=first,prev=first;
          if(first==NULL)
          {
          printf("\nlist is empty");
          return;
          }
          if(first->link==NULL)
          {
          free(first);
          first=NULL;
          }
          else
          {
                    while(cur->link!=NULL)
                    {
                    prev=cur;
                    cur=cur->link;
                    }
          free(cur);
          prev->link=NULL;
          }
 return first;
}
NODE stacksimulation(NODE first)
{
 int ch;
          while(ch!=3)
          {
           printf("\n SLL used as stack");
           printf("\n 1.push(insert at front)\t 2.pop(delete at front)\t 3.exit");
           printf("enter your choice:");
           scanf("%d",&ch);
                    switch(ch)
                    {
                    case 1 : first=insertfront(first);
                              break;
                    case 2 : first=deletefront(first);
                              break;
                    case 3 : return first;
                    }//while(ch!=3);
          display(first);
          return first;
          }
}
NODE createlist(NODE first)
{
  int i,n;
  printf("enter the no. of student we need to add to list:");
  scanf("%d",&n);
  for(i=0;i<n;i++)
   first=insertfront(first);
  return first;
}
int main()
{
  int ch;
  NODE first;first=NULL;
  printf("\n-------student database--------\n");
          while(1)
          {
          printf("\n1.create \t 2.insert front \t 3.insert rear \t 4.delete front \n 5.delete
rear \t 6.stack simulation \t 7.display\t 8.exit");
          printf("\nenter choice:");
          scanf("%d",&ch);
                    switch(ch)
                    {
                    case 1 : first=createlist(first);
                              break;
                    case 2 : first=insertfront(first);
                              break;
                    case 3 : first=insertrear(first);
                              break;
                    case 4 : first=deletefront(first);
                              break;
                    case 5 : first=deleterear(first);
                              break;
                    case 6 : first=stacksimulation(first);
                              break;
                    case 7 : display(first);
                              break;
                    case 8 : exit(0);
                    }
          }
}


/* 8.Doubly Linked Lists  */
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct node
{
  int ssn;
  char name[20];
  char desi[20];
  char dept[20];
  int sal;
  char ph[20];
  struct node *llink;
  struct node *rlink;
```

```
};
typedef struct node *NODE;
NODE insertfront(NODE first)
{
  NODE temp;
  temp=(NODE)malloc(sizeof(struct node));
  printf("enter employee details");
  printf("\nenter ssn,name,dept,desig,salary,phone no.:\n");
  scanf("%d",&temp->ssn);
  scanf("%s",temp->name);
  scanf("%s",temp->dept);
  scanf("%s",temp->desi);
  scanf("%d",&temp->sal);
  scanf("%s",temp->ph);
  if(first==NULL)
   return temp;
  temp->rlink=first;
  first->llink=temp;
  temp->llink=NULL;
  return temp;
}
NODE insertrear(NODE first)
{
  NODE temp,cur;
  temp=(NODE)malloc(sizeof(struct node));
  printf("\nenter employee details\n");
  printf("\nenter ssn,name,dept,desig,salary,phone no.:\n");
  scanf("%d",&temp->ssn);
  scanf("%s",temp->name);
  scanf("%s",temp->dept);
  scanf("%s",temp->desi);
  scanf("%d",&temp->sal);
  scanf("%s",temp->ph);
  if(first==NULL)
   return temp;
  cur=first;
  while(cur->rlink!=NULL)
   cur=cur->rlink;
  cur->rlink=temp;
  temp->llink=cur;
  temp->rlink=NULL;
  return first;
}
NODE deletefront(NODE first)
```

```
{
 NODE temp;
        if(first==NULL)
        {
        printf("\nlist is empty");
        return;
        }
        if(first->rlink==NULL)
        {
        printf("\nemployee details deleted ssn:%d\n",first->ssn);
        free(first);
        return NULL;
        }
 temp=first->rlink;
 temp->llink=NULL;
 printf("\nemp details ssn:%d\n",first->ssn);
 free(first);
 return temp;
}
NODE deleterear(NODE first)
{
  NODE temp,cur;
        if(first==NULL)
        {
        printf("\nempty list\n");
        return;
        }
        if(first->rlink==NULL)
        {
        printf("\nemp details ssn:%d\n",first->ssn);
        free(first);
        return NULL;
        }
        cur=first;
        while(cur->rlink!=NULL)
         cur=cur->rlink;
        temp=cur->llink;
        printf("\nemp details ssn:%d\n",cur->ssn);
        temp->rlink=NULL;
        free(cur);
        return first;
}
void display(NODE first)
{
  NODE cur; int c=0;
        if(first==NULL)
        {
        printf("\nlist is empty\n");
        return;
        }
  cur=first;
        while(cur!=NULL)
        {
         printf("\n%d\n%d\n%s\n%s\n%s\n%s\n",cur->ssn,cur->sal,cur->name,cur-
>dept,cur->desi,cur->ph);
         cur=cur->rlink; c++;
        }
  printf("\nno. of emp=%d\n",c);
}
void main()
{
  NODE first;
  int ch;
  first=NULL;
        while(1)
        {
        printf("\n1.insert front\t 2.insert rear\t 3.delete front\n4.delete
rear\t5.display\t 6.exit");
        printf("\nenter choice:");
        scanf("%d",&ch);
                switch(ch)
                {
                case 1 : first=insertfront(first);
                        break;
                case 2 : first=insertrear(first);
                        break;
                case 3 : first=deletefront(first);
                        break;
                case 4 : first=deleterear(first);
                break;
                case 5 : display(first);
                        break;
                case 6 : exit(0);
                }
        }
}
```

```
/* 9.Polynomial operations using Circular singly linked lists */
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
struct poly
{
  int cf,px,py,pz;
  int flag;
  struct poly *link;
};
typedef struct poly *NODE;

NODE insertrear(NODE h,int cf,int px,int py,int pz)
{
   NODE  temp,cur;
   temp=(NODE)malloc(sizeof(struct poly));
   temp->cf=cf;
   temp->px=px;
   temp->py=py;
   temp->pz=pz;
          if(h->link==h)
          {
          h->link=temp;
          temp->link=h;
          return h;
          }
   cur=h->link;
   while(cur->link!=h)
        cur=cur->link;
   cur->link=temp;
   temp->link=h;
   return h;
}
NODE readpoly(NODE h)
{
  int cf,px,py,pz,ch;
          do
          {
          printf("enter co-eff,px,py,pz:");
          scanf("%d%d%d%d",&cf,&px,&py,&pz);
          h=insertrear(h,cf,px,py,pz);
          printf("\n1 to continue 0 to exit:");
          scanf("%d",&ch);
          }while(ch!=0);
  return h;
}
void evalpoly(NODE h1)
{
  int x,y,z;
  float result=0.0;
  NODE temp=h1->link;
  printf("\nenter the value of x,y,z:");
  scanf("%d%d%d",&x,&y,&z);
          while(temp!=h1)
          {
          result=result+temp->cf*pow(x,temp->px)*pow(x,temp->py)*pow(z,temp-
>pz);
          temp=temp->link;
          }
  printf("\nthe result=%f\n",result);
}
void display(NODE h1)
{
  NODE temp;
          if(h1->link==h1)
          {
           printf("\npolynomial is empty");
          return;
          }
  temp=h1->link;
         while(temp!=h1)
         {
          if(temp->cf>0)
          printf("+%dx%dy%dz%d",temp->cf,temp->px,temp->py,temp->pz);
         else
          printf("%dx%dy%dz%d",temp->cf,temp->px,temp->py,temp->pz);
         temp=temp->link;
         }
}
NODE addpoly(NODE h1,NODE h2,NODE h3)
{
  NODE p1,p2;
  int cf1,px1,py1,pz1,cf2,px2,py2,pz2,cf;
  p1=h1->link;
         while(p1!=h1)
         {
       cf1=p1->cf;
       px1=p1->px;
       py1=p1->py;
       pz1=p1->pz;
       p2=h2->link;
         while(p2!=h2)
         {
         cf2=p2->cf;
         px2=p2->px;
         py2=p2->py;
         pz2=p2->pz;
         if(px1==px2&&py1==py2&&pz1==pz2)
                    break;
         p2=p2->link;
         }
        if(p2!=h2)
        {
         cf=cf1+cf2;
         p2->flag=1;
         if(cf!=0)
          h3=insertrear(h3,cf,px1,py1,pz1);
         p1=p1->link;
         p2=p2->link;
        }
       else
         {
       h3=insertrear(h3,cf1,px1,py1,pz1);
```

```
        p1=p1->link;
        }
    }
          p2=h2->link;
                   while(p2!=h2)
                   {
                   if(p2->flag==0)
                    h3=insertrear(h3,p2->cf,p2->px,p2->py,p2->pz);
                   p2=p2->link;
                   }
                   return h3;
}

void main()
{
  int ch;
  NODE h1,h2,h3;
  h1=(NODE)malloc(sizeof(struct poly));
  h2=(NODE)malloc(sizeof(struct poly));
  h3=(NODE)malloc(sizeof(struct poly));
  h1->link=h1;
  h2->link=h2;
  h3->link=h3;
          while(1)
          {
           printf("\n1.evaluate a polynomial\n2.add polynomial\n3.exit\n");
           printf("enter choice:");
           scanf("%d",&ch);
                    switch(ch)
                    {
                      case 1 : printf("enter polynomial:");
                               h1= readpoly(h1);
                               evalpoly(h1);
                               display(h1);
                               h1->link=h1;
                               break;
                      case 2 : printf("enter 1st polynomial:");
                                h1=readpoly(h1);
                               printf("enter 2nd polynomial:");
                                h2=readpoly(h2);
                               h3=addpoly(h1,h2,h3);
                               printf("\n1st polynomial is\n");
                               display(h1);printf("\n");
                               printf("\n2nd polynomial is\n");
                    display(h2);printf("\n");
                    printf("\nresultant polynomial is \n");
                    display(h3);printf("\n");
                            break;
                  case 3 : exit(0);
                }
        }
}
```

```
/* 10.Binary Search tree */

#include<stdio.h>
#include<stdlib.h>
struct node
{
  int info;
  struct node *llink;
  struct node *rlink;
};
typedef struct node *NODE;
NODE insert(NODE root,int item)
{
  NODE temp,cur,prev;
  temp=(NODE)malloc(sizeof(NODE));
  temp->info=item;
  temp->llink=temp->rlink=NULL;
  if(root==NULL)
    return temp;
  cur=root;
        while(cur!=NULL)
        {
        prev=cur;
                if(cur->info==item)
                {
                printf("insertion not possible");
                free(temp);
                return;
                }
        if(cur->info > item)
        cur=cur->llink;
        else
```

```
          cur=cur->rlink;
          }
  if(prev->info>item)
   prev->llink=temp;
  else
   prev->rlink=temp;
  return root;
}
void search(NODE root,int key)
{
  NODE cur;
  cur=root;
          if(root==NULL)
          {
          printf("tree is empty");
          return;
          }
          while(cur!=NULL)
          {
                    if(cur->info==key)
                    {
                    printf("key element is found");
                    return;
                    }
          if((cur->info)>key)
          cur=cur->llink;
          else
          cur=cur->rlink;
          }
  printf("key not found");
}
void preorder(NODE root)
{
 if(root==NULL) return;
 printf("%d\t",root->info);
 preorder(root->llink);
 preorder(root->rlink);
}
void postorder(NODE root)
{
  if(root==NULL)return;
  postorder(root->llink);
  postorder(root->rlink);
  printf("%d\t",root->info);
```

```
}
void inorder(NODE root)
{
  if(root==NULL)return;
  inorder(root->llink);
  printf("%d\t",root->info);
  inorder(root->rlink);
}
void main()
{
    int ch,x,item,key;
    NODE root=NULL;
          while(1)
          {
          printf("\n1.create a BST\n2.traverse inpreorder\n3.traverse in
postorder\n4.inorder");
          printf("\n5.search in a BST\n6.exit");
          printf("\nenter choice:");
          scanf("%d",&ch);
                    switch(ch)
                    {
                    case 1 :          do
                                      {
                              printf("enter the element to be inserted:");
                              scanf("%d",&item);
                              root=insert(root,item);
                              printf("enter 1 to continue 0 to exit");
                              scanf("%d",&x);
                              }while(x!=0);
                              break;
                    case 2 : preorder(root);
                              break;
                    case 3 : postorder(root);
                              break;
                    case 4 : inorder(root);
                              break;
                    case 5 : printf("enter the element to search:");
                               scanf("%d",&key);
                              search(root,key);
                              break;
                    case 6 : exit(0);
                    }
          }
}
```

```
/* 11.Graph traversal Using BFS and DFS  */
#include<stdio.h>
#include<stdlib.h>
```

```
int a[10][10],n,s[10]={0};
void bfs(int u)
  {
         int f,r,q[10],v,i,s[10]={0};
        printf("The Nodes visited from %d:",u);
        f=0;r=-1;
        q[++r]=u;
        s[u]=1;
        printf(" %d",u);
                while(f<=r)
                        {
                          u=q[f++];
                        for(v=0;v<n;v++)
                        if(a[u][v]==1)
                                 if(s[v]==0)
                                {
                                        printf(" %d",v);
                                        s[v]=1;
                                        q[++r]=v;
                                }

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

int main()
{
  int u,i,j,ch;
   while(1)
        {
        printf("\n1.Create a graph");
         printf("\n2.Traversal using BFS");
         printf("\n3.Traversal using DFS");
        printf("\n4.Exit");
         printf("\nEnter your choice :");
```

```
    scanf("%d",&ch);
        switch(ch)
            {
            case 1:printf("Enter the number of nodes");
                scanf("%d",&n);
                for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                    {
                    printf("\n Enter A[%d][%d]=",i,j);
                    scanf("%d",&a[i][j]);
                    }
                break;
             case 2: printf("\n Enter the starting/source  verstex ");
                scanf("%d",&u);
                 bfs(u);
                break;
            case 3: printf("\n Enter the starting/source  verstex ");
                scanf("%d",&u);
                for(i=0;i<10;i++)
                s[i]=0;
                printf("The nodes visted from %d:",u);
                dfs(u);
                 break;
            case 4: _Exit(0);
            }
    }
}
```

```
/*12. Hashing  using Linear Probing   */
#include <stdio.h>
#include <stdlib.h>
#define MAX 10
int count=0;
struct employee
        {
        int id;
        char name[15];
        };
typedef struct employee EMP;
EMP emp[MAX];
int a[MAX];

int create(int num)
        {
        int key;
        key = num % 10;
        return key;
        }

int getemp(EMP emp[],int key)
        {
        printf("\nEnter emp id: ");
        scanf("%d",&emp[key].id);
        printf("\nEnter emp name: ");
        scanf("%s",emp[key].name);
        return key;
        }
```

```
void display()
{
int i, ch;
printf("\n1.Display ALL\n2.Filtered Display");
printf("\nEnter the choice: ");
scanf("%d",&ch);
if(ch == 1)
    {
    printf("\nThe hash table is:\n");
    printf("\nHTKey\tEmpID\tEmpName");
    for(i=0; i<MAX; i++)
    printf("\n%d\t%d\t%s", i, emp[i].id, emp[i].name);
    }
else
    {
    printf("\nThe hash table is:\n");
    printf("\nHTKey\tEmpID\tEmpName");
    for(i=0; i<MAX; i++)
    if(a[i]!= -1)
    printf("\n%d\t%d\t %s", i, emp[i].id, emp[i].name);
    }
}


void linear_prob(int key)
{
    int i;
    if(count==MAX)
    {
    printf("Hash Table is full");
    return;
    }
        if(a[key] == -1)
        {
        a[key]=getemp(emp,key);
        count++;
        }
        else
        {
        printf("\nCollision Detected...!!!\n");
        i=(key+1)%MAX;
        printf("%d",i);
            while(i < MAX||i<key)
```

```
                {
                if (a[i]==-1)
                    {
                    a[i]=getemp(emp,i);
                    count++;
                    break;
                    }
                else
                i=(i+1)%MAX;
                }
        }
}
void main()
{
int num, key, i,x;
printf("\nCollision handling by linear probing: ");
    for (i=0; i < MAX; i++)
        a[i] = -1;
    do
    {
    printf("\nEnter the data: ");
    scanf("%d", &num);
    key=create(num);
    linear_prob(key);
    display(emp);
    printf("\nDo you wish to continue? (1/0): ");
    scanf("%d",&x);
    }while(x!=0);
}
```

DSA MANUAL

18CSL38

2019-2020