EMPLOYEE:

```
#include<stdio.h>
#include<stdlib.h>
struct employee{
int id;
char name[20];
float salary;
};
void main( )
{
int i, n, ch, searchid;
struct employee emp[5];
printf("Enter the number of employees: ");
scanf("%d", &n);
printf("Enter %d employee details: \n", n);
for (i=0; i<n; i++)
{
printf("Enter employee id: ");
scanf("%d", &emp[i].id);
printf("Enter employee name: ");
scanf("%s", emp[i].name);
printf("Enter employee salary: ");
scanf("%f", &emp[i].salary);
}
while(1)
{
printf("\n1. Display \n 2. Search \n3. Exit \n Enter your choice ");
scanf("%d",&ch);

switch(ch)
```

```
{
case 1: for (i=0; i < n; i++)
          {
            printf("\nEmployee id: %d, Name: %s, Salary: Rs.%f", emp[i].id,
            emp[i].name,  emp[i].salary);
          }
          break;

case 2: printf("Enter Emp ID to be searched:");
          scanf("%d", &searchid);
          for (i=0; i < n; i++)
          {
            if(emp[i].id== searchid)
            {
              printf("\nEmployee id: %d, Name: %s, Salary: Rs.%f", emp[i].id,
              emp[i].name, emp[i].salary); break;
            }
          }
          if(i==n)
            printf("Empolyee ID not found");
          break;

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

SPARSE MATRIX:

```
#include<stdio.h>
#define SROW 50
#define MROW 20
#define MCOL 20
int main()
{
int mat[MROW][MCOL],sparse[SROW][3];
int i,j,nzero=0,mr,mc,sr,s,elem;
printf("Enter number of rows:\n");
scanf("%d",&mr);
printf("Enter number of column:\n");
scanf("%d",&mc);
printf("Enter the matrix\n");
for(i=1;i<=mr;i++)
{
for(j=1;j<=mc;j++)
{
scanf("%d",&mat[i][j]);
if(mat[i][j]!=0)
{
nzero++;
}
}
}
sr=nzero+1;
sparse[1][1]=mr;
sparse[1][2]=mc;
sparse[1][3]=nzero;
s=2;
```

```
for(i=1;i<=mr;i++)
{
for(j=1;j<=mc;j++)
{ if(mat[i][j]!=0)
{
sparse[s][1]=i;
sparse[s][2]=j;
sparse[s][3]=mat[i][j];
s++;
}
}
}
printf("\nSparse matrix is \n");
for(i=1;i<=sr;i++)
{
for(j=1;j<=3;j++)
{
printf("%d ",sparse[i][j]);
}
printf("\n");
}
printf("Enter the element to be searched \n");
scanf("%d",&elem);
for(i=2;i<sr;i++)
{
if(sparse[i][3]==elem)
{printf("Element found at (row,col)=(%d,%d)",sparse[i][1],sparse[i][2]);
return 1;
}
}
```

```
printf("Element not found");
return 0;
}
```

STACK:

```
#include<stdio.h>
#define STACK_SIZE 3
int s[STACK_SIZE];
int top=-1;
void push()
{
int n;
if(top==STACK_SIZE-1)
printf("\nStack overflow\n");
else
{
printf("\nEnter the data to be pushed\n");
scanf("%d",&n);
s[++top]=n;
}
}
void pop()
{
if(top==-1)
printf("\nStack empty\n");
else
printf("\n%d is popped\n", s[top--]);
}
void display()
{
int i;
if(top==-1)
printf("\nStack empty\n");
else
```

```
{
printf("\nStack elements are\n");
for(i=0;i<=top;i++)
printf("%d\n",s[i]);
}
}
int main()
{
int ch;
for(;;)
{
printf("\n1.PUSH\t2.POP\t3.DISPLAY\t4.EXIT\n");
printf("\nEnter your choice\n");
scanf("%d",&ch);
switch(ch)
{
case 1: push();
break;
case 2: pop();
break;
case 3: display();
break;
case 4: return 0;
default: printf("\nInvalid choice\n");
}
}
}
```

INFIX TO POSTFIX:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
int top = -1;
char stack[40];
void push(char x)
{
stack[++top] = x;
}
int pop( )
{
return stack[top--];
}
int prior(char x){
int p;
if(x=='(' || x=='#')
p=1;
if(x=='+' || x=='-')
p=2;
if(x=='*' || x=='/')
p=3;
if(x=='^' || x=='$')
p=4;
return p;
}
void main( )
{
char infix[30], postfix[30];
```

```
int i, j=0;
printf("Enter the infix expression:\n");
gets(infix);
push('#');
for(i=0; i<strlen(infix); i++)
{
if(isalnum(infix[i]))
postfix[j++] = infix[i];
else if(infix[i]=='(')
push(infix[i]);
else if(infix[i]==')')
{
while(stack[top]!='(')
postfix[j++]=pop( );
pop( );
}
else{
while(prior(stack[top]) == prior(infix[i]))
postfix[j++]=pop( );
push(infix[i]);
}
}
while(stack[top] != '#')
postfix[j++] = pop( );
postfix[j]='\0';
printf("The postfix expression is:\n");
puts(postfix);
}
```

EVALUATION OF POSTFIX:

```
#include<stdio.h>
#include<string.h>
#include<math.h>
#define MAXSIZE 30
int s[MAXSIZE];
int top=-1;
int pop()
{
   if(top!=-1)
   {
      return s[top--];
   }
   else
   {
    printf("underflow\n");
    return 0;
   }
}
void push(int item)
{
   if(top!=MAXSIZE-1)
   {
      s[++top]=item;
   }
   else{
      printf("overflow\n");
      return;
   }
}
```

```
int op(int op1,int op2,char symbol)
{
   switch(symbol)
   {
      case '+':return op1+op2;
      case '-':return op1-op2;
      case '*':return op1*op2;
      case '/':return op1/op2;
   }
}
int isdig(char symbol1)
{
   return(symbol1>='0'&&symbol1<='9');
}
void main()
{
   char symbol,postfix[30];
   int a,b,res,i;
   printf("Enter the postfix expresssion:\n");
   scanf("%s",&postfix);
   for(i=0;i<strlen(postfix);i++)
   {
      symbol=postfix[i];
      if(isdig(symbol))
       push(symbol-'0');
      else
      {
         a=pop();
         b=pop();
         res=op(b,a,symbol);
```

```
            push(res);
        }
    }
    printf("The result of the expression is:\n");
    printf("%d\n",pop());
}
```

QUEUE:

```
#include<stdio.h>
#define MAXSIZE 3
int q[maxsize], f=0,r=-1;
void insert()
{
int n;
if(rear==MAXSIZE-1)
printf("\nQueue full\n");
else
{
printf("\nEnter the data to be added\n");
scanf("%d", &n);
q[++r]=n;
}
}
void delete()
{
if(f>r)
printf("\nQueue is empty\n");
else
{
printf("\n%d is deleted\n",q[f++]);
if(f>r && r==MAXSIZE-1)
{
printf("\nRein it\n");
f=0; r=-1;
}
}
}
```

```
void display()
{
int i;
if(f>r)
printf("\nQueue is empty\n");
else
{
printf("\nQueue status is\n");
for(i=f;i<=r;i++)
printf("%d\t",q[i]);
}
}
int main()
{
int ch;
while(1)
{
printf("1.Insert\n2.Delete\n3.Display\n4.Exit\n");
puts("\nEnter your choice\n");
scanf("%d",&ch);
switch(ch){
case 1: insert(); break;
case 2: delete(); break;
case 3: display(); break;
case 4: return 0;
default: printf("\nInvalid choice\n");
}
}
}
```

SINGLY LINKED LIST:

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
int info;
struct node *link;
};
typedef struct node *NODE;
NODE insertLoc(NODE first)
{
int loc,count;
NODE temp,cur;
printf("\nEnter the location\n");
scanf("%d",&loc);
temp = (NODE)malloc(sizeof(struct node));
printf("\nEnter the data\n");
scanf("%d",&temp->info);
temp->link=NULL;
if(first==NULL)
{
if(loc==1)
first = temp;
else
printf("Invalid location\n");
}
else if(loc==1)
{
temp->link=first;
first=temp;
```

```
}
else
{
cur=first;
count=1;
while(cur!=NULL)
{
if(count==loc-1)
{
temp->link=cur->link;
cur->link=temp;
break;
}
cur=cur->link;
count++;
}
if(cur==NULL)
printf("Invalid location\n");
}
return first;
}
NODE delete (NODE first)
{
NODE temp;
if (first == NULL)
{
printf ("List Empty\n");
return first;
}
temp = first;
```

```
first = first->link;
printf ("%d is deleted\n",temp->info);
free (temp);
return first;
}
void display (NODE first)
{
NODE temp;
if (first == NULL)
printf ("List is Empty\n");
else
{
printf ("Content of List\n");
temp = first;
while (temp != NULL)
{
printf ("%d\t",temp->info);
temp = temp->link;
}
printf ("\n");
}
}
int main ()
{
int ch;
NODE first = NULL;
for (;;)
{
printf ("1:INSERT 2:DELETE 3:DISPLAY 4.EXIT\n");
scanf ("%d",&ch);
```

```
switch (ch)
{
case 1: first = insertLoc (first);
break;
case 2: first = delete (first);
break;
case 3: display (first);
break;
default: exit(0);
}
}
}
```

DOUBLY LINKED LIST:

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
int info;
struct node *llink;
struct node *rlink;
};
typedef struct node *NODE;
NODE first = NULL,last = NULL;
void insert (int data)
{
NODE newnode;
newnode = (NODE)malloc(sizeof(struct node));
newnode->info = data;
newnode->llink = NULL;
newnode->rlink = NULL;
if(first == NULL)
{
first=last=newnode;
return;
}
newnode->rlink = first;
first->llink = newnode;
first = newnode;
}
void delete (int key)
{
int flag =0;
```

```
NODE prev,cur,next;
if (first == NULL)
{
printf ("List Empty\n");
return;
}
if(first->rlink == NULL) // one node in the list
{
if (first->info == key)
{
printf ("%d is deleted\n",first->info);
free (first);
first=last=NULL;
return ;
}
}
if(key == first->info)
{
printf("\n%d is deleted\n",first->info);
cur = first;
first = first->rlink;
first->llink = NULL;
free(cur);
cur=NULL;
return;
}
if(key == last -> info)
{
printf("\n%d is deleted\n",last->info);
cur = last;
```

```
last = last->llink;
last->rlink = NULL;
free(cur);
cur=NULL;
return;
}
cur = first->rlink;
while(cur!=last)
{
if(cur->info==key)
{
prev = cur->llink;
next = cur->rlink;
printf("\n%d is deleted\n",cur->info);
prev->rlink = next;
next->llink = prev;
free(cur);
cur = NULL;
flag =1;
break;
}
cur=cur->rlink;
}
if(flag==0)
printf("\nKey not found\n");
}
void display ()
{
NODE temp;
if (first == NULL)
```

```
printf ("List is Empty\n");
else
{
printf ("Content of List\n");
temp = first;
while (temp != NULL)
{
printf ("%d\t",temp->info);
temp = temp->rlink;
}
printf ("\n");
}
}
int main ()
{
int ch,data;
for (;;)
{
printf ("1:INSERT 2:DELETE 3:DISPLAY 4:EXIT\n");
scanf ("%d",&ch);
switch (ch)
{
case 1: printf ("Enter the data\n");
scanf ("%d",&data);
insert (data);
break;
case 2: printf ("Enter the data to be deleted\n");
scanf ("%d",&data);
delete (data);
break;
```

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

MAX HEAP:

```
#include<stdio.h>
#include<stdlib.h>
#define MAXSIZE 10
int insertion(int item, int a[ ], int n)
{
int c,p;
if(n==MAXSIZE)
{
printf("HEAP IS FULL!!!\n");
return;
}
c=n;
p=(c-1)/2;
while(c!=0&&item>a[p])
{
a[c]=a[p];
c=p;
p=(c-1)/2;
}
a[c]=item;
return n+1;
}
void display(int a[],int n)
{
int i;
if(n==0)
{
printf("HEAP IS EMPTY!!!\n");
return;
```

```
}
printf("The array elements are: \n");
for(i=0;i<n;i++)
printf("%d ",a[i]);
}
void main( )
{
int a[MAXSIZE],n=0,ch,item;
for(;;)
{
printf("\n1.INSERT\t2.DISPLAY\t3.EXIT\n");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("Enter the element:");
        scanf("%d",&item);
        n=insertion(item,a,n);
        break;
case 2: display(a,n);
        break;
default: exit(0);
}
}
}
```

BINARY TREE:

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

struct node {
   int info;
   struct node *left;
   struct node *right;
};

typedef struct node NODE;
NODE *root=NULL;

void insert(int x) {
   NODE *temp,*prev,*cur;
   temp = (NODE*)malloc(sizeof(NODE));
   temp->left = NULL;
   temp->right = NULL;
   temp->info = x;

   if(root == NULL) {
      root = temp;
      return;
   }
   prev = NULL;
   cur = root;

   while(cur != NULL) {
      prev = cur;
      if(x < cur->info)
```

```
            cur = cur->left;
        else if(x > cur->info)
            cur = cur->right;
        else {
            printf("\nDuplicate value not allowed.");
            return;
        }
    }

    if(x < prev->info)
        prev->left = temp;
    else
        prev->right = temp;
}

void preorder(NODE *root) {
    if(root != NULL) {
        printf("%d ", root->info);
        preorder(root->left);
        preorder(root->right);
    }
}

void inorder(NODE *root) {
    if(root != NULL) {
        inorder(root->left);
        printf("%d ", root->info);
        inorder(root->right);
    }
}
```

```
void postorder(NODE *root) {
    if(root != NULL) {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->info);
    }
}

int main() {
    int item, ch;
    while(1)
    {
        printf("\n\n1.Insert\n2.Preorder\n3.Inorder\n4.Postorder\n5.Exit\n");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);

        switch(ch) {
            case 1:  printf("\nEnter element to be inserted: ");
                     scanf("%d", &item);
                     insert(item);
                     break;
            case 2: printf("\nPreorder traversal: ");
                     preorder(root);
                     break;
            case 3: printf("\nInorder traversal: ");
                inorder(root);
                break;
            case 4: printf("\nPostorder traversal: ");
                     postorder(root);
```

```
                    break;
            case 5: exit(0);
            default: printf("\nInvalid choice");

        }
    }
    return 0;
}
```