1)Infix to Postfix

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#define SIZE 20
struct stack
{
 int top;
 char data[SIZE];
};
void push(struct stack *s,char item)
{
 s->data[++(s->top)]=item;
}
char pop(struct stack *s)
{
 return s->data[(s->top)--];
}
int preced(char symbol)
{
 switch(symbol)
 {
 case '^':return 5;
 case '*':
 case '/':return 3;
 case '+':
 case '-':return 1;
 }
}
void infixtopostfix(struct stack *s,char infix[SIZE])
```

```
{
 int i,j=0;
 char postfix[SIZE],temp,symbol;
 for(i=0;infix[i]!='\0';i++)
 {
 symbol=infix[i];
 if(isalnum(symbol))
 {
 postfix[j++]=symbol;
 }
 else
 {
 switch(symbol)
 {
 case '(':push(s,symbol);
 break;
 case ')':temp=pop(s);
 while(temp!='(')
{
 postfix[j++]=temp;
temp=pop(s);
 }
 break;
 case '+':
 case '-':
 case '*':
 case '/':
 case '^': if (s->top ==-1 || s->data[s->top]=='(')
 push(s,symbol);
 else
```

```
{
 while(preced(s->data[s->top])>=
preced(symbol) && s->top!=-1 &&s->data[s->top]!='(')
{
 postfix[j++]=pop(s);
}
 push(s,symbol);
 }
break;
 default :printf("\n Invalid!!!!!");
 exit(0);
 }
 }
 }
 while(s->top!=-1)
 postfix[j++]=pop(s);
 postfix[j]='\0';
 printf("\n The postfix expression is %s\n",postfix);
}
int main()
{
 struct stack s;
 s.top=-1;
 char infix[SIZE];
 printf("\n Read Infix expression\n");
 scanf("%s",infix);
 infixtopostfix(&s,infix);
 return 0;
}
```

2)Evaluation of Prefix

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <math.h>
#include <string.h>
#define SIZE 20
struct stack
{
 int top;
 float data[SIZE];
};
void push(struct stack *s,float item)
{ s->data[++(s->top)]=item;}
float pop(struct stack *s)
{ return s->data[(s->top)--];}
float operate(float op1,float op2,char symbol)
{
 switch(symbol)
 {
 case '+':return op1+op2;
 case '-':return op1-op2;
 case '*':return op1*op2;
 case '/':return op1/op2;
 case '^':return pow(op1,op2);
 }
}
float eval(struct stack *s,char prefix[SIZE])
{
 int i;
```

```
 char symbol;
 float res,op1,op2;
 for(i=strlen(prefix)-1;i>=0;i--)
 {
 symbol=prefix[i];
 if(isdigit(symbol))
 push(s,symbol-'0');
 else
 {
 op1=pop(s);
 op2=pop(s);
 res=operate(op1,op2,symbol);
 push(s,res);
 }
 }
 return pop(s);
}
int main()
{
 char prefix[SIZE];
 struct stack s;
 float ans;
 s.top=-1;
 printf("\n Read prefix expression:\n");
 scanf("%s",prefix);
 ans=eval(&s,prefix);
 printf("\n The final answer is %f\n",ans);
 return 0;
}
```

3)Message Queueing System

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define SIZE 5
struct queue
{
 int front,rear;
 char data[SIZE][20];
};
void send(struct queue *q,char item[20])
{
 if(q->front==(q->rear+1) % SIZE )
 printf("\n Queue full");
 else
 {
 q->rear=(q->rear+1)%SIZE;
 strcpy(q->data[q->rear],item);
 if(q->front==-1)
 q->front=0;
 }
}
char *receive(struct queue *q)
{
 char *del;
 if(q->front==-1)
 {
 printf("\n Queue empty");
 return -1;
 }
```

```
 else
 {
 del=q->data[q->front];
 if(q->front==q->rear)
 {
 q->front=-1;
 q->rear=-1;
 }
 else
 q->front=(q->front+1)% SIZE;
 return del;
 }
}
void display(struct queue q)
{
 int i;
 if(q.front==-1)
 printf("\n Queue Empty");
 else
 {
 printf("\n Queue content are\n");
 for(i=q.front;i!=q.rear;i=(i+1)%SIZE)
 printf("%s\n",q.data[i]);
 printf("%s\n",q.data[i]);
 }
}
int main()
{
 int ch;
 char *del;
```

```
char item[20];
struct queue q;
q.front=-1;
q.rear=-1;
for(;;)
{
printf("\n1. Send\n2. Receive\n3. Display\n4. Exit");
printf("\nRead Choice :");
scanf("%d",&ch);
getchar();
switch(ch)
{
case 1:printf("\n Read message to be send :");
gets(item);
send(&q,item);
break;
case 2:del=receive(&q);
if(del!=NULL)
printf("\n Element deleted is %s\n",del);
break;
case 3:display(q);
break;
default:exit(0);
}
}
return 0;
}
```

4) Multiplication of Polynomials using Singly Linked List

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5
int count;
struct node
{
 int co,po;
 struct node *addr;
};
struct node * insertend(struct node * start,int co,int po)
{
 struct node * temp,* cur;
 temp=(struct node *)malloc(sizeof(struct node));
 temp->co=co;
 temp->po=po;
 temp->addr=NULL;
 if(start==NULL)
 return temp;
 cur=start;
 while(cur->addr!=NULL)
 {
 cur=cur->addr;
 }
 cur->addr=temp;
 return start;
}
void display(struct node * start)
{
 struct node * temp;
```

```
 if(start==NULL)
 printf("\n Polynomial Empty");
 else
 {
 temp=start;
 while(temp->addr!=NULL)
 {
 printf("%dx^%d+",temp->co,temp->po);
 temp=temp->addr;
 }
 printf("%dx^%d\n",temp->co,temp->po);
 }
}
struct node * addterm(struct node * res,int co,int po)
{
 struct node * temp,* cur;
 temp=(struct node *)malloc(sizeof(struct node));
 temp->co=co;
 temp->po=po;
 temp->addr=NULL;
 if(res==NULL)
 return temp;
 cur=res;
 while(cur!=NULL)
 {
 if(cur->po==po)
 {
 cur->co=cur->co+co;
 return res;
 }
```

```
 cur=cur->addr;
 }
 if(cur==NULL)
 res=insertend(res,co,po);
 return res;
}
struct node * multiply(struct node * poly1,struct node * poly2)
{
 struct node * p1,* p2,* res=NULL;
 for(p1=poly1;p1!=NULL;p1=p1->addr)
 for(p2=poly2;p2!=NULL;p2=p2->addr)
 res=addterm(res,p1->co*p2->co,p1->po+p2->po);
 return res;
}
int main()
{
 struct node * poly1=NULL,* poly2=NULL,* poly;
 int co,po;
 int i,n,m;
 printf("\nRead no of terms of first polynomial:");
 scanf("%d",&n);
 for(i=1;i<=n;i++)
 {
 printf("\n Read CO and PO of %d term : ",i);
 scanf("%d%d",&co,&po);
 poly1=insertend(poly1,co,po);
 }
 printf("\n First polynomial is\n");
 display(poly1);
 printf("\nRead no of terms of second polynomial:");
```

```
scanf("%d",&m);
for(i=1;i<=m;i++)
{
printf("\n Read CO and PO of %d term : ",i);
scanf("%d%d",&co,&po);
poly2=insertend(poly2,co,po);
}
printf("\n Second polynomial is\n");
display(poly2);
poly=multiply(poly1,poly2);
printf("\n Resultant polynomial is\n");
display(poly);
return 0;
}
```

5)Queue of Integers using Circular List

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5
int count;
struct node
{
 int data;
 struct node *addr;
};
typedef struct node *NODE;
NODE insertend(NODE last,int item)
{
 NODE temp;
 if(count>=SIZE)
 {
 printf("\n Queue full");
 return last;
 }
 count=count+1;
 temp=(NODE)malloc(sizeof(struct node));
 temp->data=item;
 if(last==NULL)
 {
 temp->addr=temp;
 return temp;
 }
 else
 {
 temp->addr=last->addr;
```

```
 last->addr=temp;
 return temp;
 }
}
NODE deletebegin(NODE last)
{
 NODE temp;
 if(last==NULL)
 {
 printf("\n Queue empty");
 return NULL;
 }
 if(last->addr==last)
 {
 printf("\n Element deleted is %d\n",last->data);
 free(last);
 return NULL;
 }
 else
 {
 temp=last->addr;
 last->addr=temp->addr;
 printf("\n Element deleted is %d\n",temp->data);
 free(temp);
 return last;
 }
}
void display(NODE last)
{
 NODE temp;
```

```
 if(last==NULL)
 printf("\n Queue is empty");
 else
 {
 printf("\n Queue Content are\n");
 temp=last->addr;
 while(temp!=last)
 {
 printf("%d\t",temp->data);
 temp=temp->addr;
 }
 printf("%d\t",temp->data);
 }
}
int main()
{
 NODE last=NULL;
 int item,ch;
 for(;;)
 {
 printf("\n1.Insert\n2.Delete\n3.Display\n4.Exit");
 printf("\nRead Choice :");
 scanf("%d",&ch);
 switch(ch)
 {
 case 1:printf("\n Read data to be inserted:");
 scanf("%d",&item);
last=insertend(last,item);
break;
 case 2:last=deletebegin(last);
```

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

6)Hashing

```
#include <stdio.h>
#include<stdlib.h>
#define SIZE 13

int h[SIZE]={0};

void insert()
{

 int x,hx1,hx2,hx,i;
 printf("\nEnter a value to insert into hash table\n");
 scanf("%d",&x);
 hx1=x%SIZE;
 hx2 = 1+x%(SIZE-2);
 for(i=0;i<SIZE;i++)
 {
   hx=(hx1+i*hx2)%SIZE;
   if(h[hx] == 0)
   {
      h[hx]=x;
      break;
   }
 }
 if(i == SIZE)
    printf("\nElement cannot be inserted\n");
}
void search()
{
```

```
 int x,hx1,hx2,hx,i;
 printf("\nEnter search element\n");
 scanf("%d",&x);
 hx1 = x % SIZE;
 hx2 = 1+x%(SIZE-2);
 for(i=0;i<SIZE; i++)
 {
   hx=(hx1+i*hx2)%SIZE;
   if(h[hx]==x)
   {
     printf("Value is found at index %d",hx);
     break;
   }
 }
 if(i == SIZE)
   printf("\n value is not found\n");
}
void display()
{

 int i;
 printf("\nElements in the hash table are: \n");
 for(i=0;i< SIZE; i++)
   printf("\nAt index %d \t value =  %d",i,h[i]);

}
void main()
{
   int opt,i;
   while(1)
```

```
    {
        printf("\nPress 1. Insert\t 2. Display \t3. Search \t4.Exit \n");
        scanf("%d",&opt);
        switch(opt)
        {
            case 1:
                insert();
                break;
            case 2:
                display();
                break;
            case 3:
                search();
                break;
            case 4:exit(0);
        }
    }
}
```

7) Priority Queue using Heap

```
#include <stdio.h>
#include <stdlib.h>
void heapify(int a[10],int n)
{
 int i,k,v,j,flag=0;
 for(i=n/2;i>=1;i--)
 {
 k=i;
 v=a[k];
 while(!flag && 2*k <= n)
 {
 j=2*k;
 if(j<n)
 {
 if(a[j]<a[j+1])
 j=j+1;
 }
 if(v>=a[j])
 flag=1;
 else
 {
 a[k]=a[j];
 k=j;
 }
 }
 a[k]=v;
 flag=0;
 }
}
```

```
int main()
{
 int n,i,a[10],ch;
 for(;;)
 {
 printf("\n 1. Create Heap");
 printf("\n 2. Extractmax");
 printf("\n 3. Exit");
 printf("\n Read Choice :");
 scanf("%d",&ch);
 switch(ch)
 {
 case 1:printf("\n Read no of elements :");
 scanf("%d",&n);
printf("\n Read Elements\n");
 for(i=1;i<=n;i++)
 scanf("%d",&a[i]);
 heapify(a,n);
printf("\n Elements after heap\n");
 for(i=1;i<=n;i++)
 printf("%d\t",a[i]);
 break;
 case 2:if(n>=1)
 {
 printf("\n Element deleted is %d\n",a[1]);
 a[1]=a[n];
n=n-1;
heapify(a,n);
if(n!=0)
{
```

```
 printf("\n Elements after reconstructing heap\n");
 for(i=1;i<=n;i++)
 printf("%d\t",a[i]);
 }
 }
else
 printf("\n No element to delete");
 break;
 default:exit(0);
 }
 }
 return 0;
}
```

8)Expression Tree

```
#include <stdio.h>
#include <stdlib.h>
#include<ctype.h>
struct node
{
 char data;
 struct node *left;
 struct node *right;
};
struct stack
{
 int top;
 struct node * data[10];
};
void push(struct stack *s,struct node * item)
{
 s->data[++(s->top)]=item;
}
struct node * pop(struct stack *s)
{
 return s->data[(s->top)--];
}
int preced(char symbol)
{
 switch(symbol)
 {
 case '$':return 5;
 case '*':
 case '/':return 3;
```

```
 case '+':
 case '-':return 1;
 }
}
struct node * createnode(char item)
{
 struct node * temp;
 temp=(struct node *)malloc(sizeof(struct node));
 temp->data=item;
 temp->left=NULL;
 temp->right=NULL;
 return temp;
}
void preorder(struct node * root)
{
 if(root!=NULL)
 {
 printf("%c",root->data);
 preorder(root->left);
 preorder(root->right);
 }
}
void inorder(struct node * root)
{
 if(root!=NULL)
 {
 inorder(root->left);
 printf("%c",root->data);
 inorder(root->right);
 }
```

```
}
void postorder(struct node * root)
{
 if(root!=NULL)
 {
 postorder(root->left);
 postorder(root->right);
 printf("%c",root->data);
 }
}
struct node * create_expr_tree(struct node * root,char infix[10])
{
 struct stack TS,OS;
 TS.top=-1;
 OS.top=-1;
 int i;
 char symbol;
 struct node * temp,* t;
 for(i=0;infix[i]!='\0';i++)
 {
 symbol=infix[i];
 temp=createnode(symbol);
 if(isalnum(symbol))
 push(&TS,temp);
 else
 {
 if(OS.top==-1)
 push(&OS,temp);
 else
 {
```

```
 while(OS.top!=-1 && preced(OS.data[OS.top]->data)>=
preced(symbol))
 {
 t=pop(&OS);
t->right=pop(&TS);
t->left=pop(&TS);
push(&TS,t);
 }
 push(&OS,temp);
 }
 }
 }
 while(OS.top!=-1)
 {
 t=pop(&OS);
 t->right=pop(&TS);
 t->left=pop(&TS);
 push(&TS,t);
 }
 return pop(&TS);
}
int main()
{
 char infix[10];
 struct node * root=NULL;
 printf("\n Read the infix expression :");
 scanf("%s",infix);
 root=create_expr_tree(root,infix);
 printf("\n The preorder traversal is\n");
 preorder(root);
```

```
 printf("\n The inorder traversal is\n");
 inorder(root);
 printf("\n The postorder traversal is\n");
 postorder(root);
 return 0;
}
```

9)Binary Tree

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
 int data;
 struct node *left;
 struct node *right;
};
struct node * create_node(int item)
{
 struct node * temp;
 temp=(struct node *)malloc(sizeof(struct node));
 temp->data=item;
 temp->left=NULL;
 temp->right=NULL;
 return temp;
}
struct node * insertleft(struct node * root,int item)
{
 root->left=create_node(item);
 return root->left;
}
struct node * insertright(struct node * root,int item)
{
 root->right=create_node(item);
 return root->right;
}
void display(struct node * root)
{
```

```
 if(root!=NULL)
 {
 display(root->left);
 printf("%d\t",root->data);
 display(root->right);
 }
}
int count_nodes(struct node * root)
{
 if (root == NULL)
 return 0;
 else
 return (count_nodes(root->left) + count_nodes(root->right) + 1);
}
int height(struct node * root)
{
 int leftht,rightht;
 if(root == NULL)
 return -1;
 else
 {
 leftht = height(root->left);
 rightht = height(root->right);
 if(leftht > rightht)
 return leftht + 1;
 else
 return rightht + 1;
 }
}
int leaf_nodes(struct node * root)
```

```
{
 if(root==NULL)
 return 0;
 else if(root->left == NULL && root->right == NULL)
 return 1;
 else
 return leaf_nodes(root->left) + leaf_nodes(root->right);
}
int nonleaf_nodes(struct node * root)
{
 if(root==NULL || (root->left == NULL && root->right == NULL))
 return 0;
 else
 return nonleaf_nodes(root->left) + nonleaf_nodes(root->right) + 1;
}
int main()
{
 struct node * root=NULL;
 root=create_node(45);
 insertleft(root,39);
 insertright(root,78);
 insertleft(root->right,54);
 insertright(root->right,79);
 insertright(root->right->left,55);
 insertright(root->right->right,80);
 printf("\n The tree(inorder) is\n");
 display(root);
 printf("\n");
 printf("\n The total number of nodes is %d\n",count_nodes(root));
 printf("\n The height of the tree is %d\n",height(root));
```

```
 printf("\n The total number of leaf nodes is %d\n",leaf_nodes(root));
 printf("\n The total number of non-leaf nodes is %d\n",nonleaf_nodes(root));
 return 0;
}
```

10)Binary Search Tree

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
 int data;
 struct node *left;
 struct node *right;
};
struct node * create_node(int item)
{
 struct node * temp;
 temp=(struct node *)malloc(sizeof(struct node));
 temp->data=item;
 temp->left=NULL;
 temp->right=NULL;
 return temp;
}
struct node * Insertbst(struct node * root,int item)
{
 struct node * temp;
 temp=create_node(item);
 if(root==NULL)
 return temp;
 else
 {
 if(item < root->data)
 root->left=Insertbst(root->left,item);
 else
 root->right=Insertbst(root->right,item);
```

```
 }
 return root;
}
void preorder(struct node * root)
{
 if(root!=NULL)
 {
 printf("%d\t",root->data);
 preorder(root->left);
 preorder(root->right);
 }
}
void inorder(struct node * root)
{
 if(root!=NULL)
 {
 inorder(root->left);
 printf("%d\t",root->data);
 inorder(root->right);
 }
}
void postorder(struct node * root)
{
 if(root!=NULL)
 {
 postorder(root->left);
 postorder(root->right);
 printf("%d\t",root->data);
 }
}
```

```
struct node * inordersuccessor(struct node * root)
{
 struct node * cur=root;
 while(cur->left != NULL)
 cur = cur->left;
 return cur;
}
struct node * deletenode(struct node * root,int key)
{
 struct node * temp;
 if(root == NULL)
 return NULL;
 if(key<root->data)
 root->left = deletenode(root->left,key);
 else if(key > root->data)
 root->right = deletenode(root->right,key);
 else
 {
 if(root->left == NULL)
 {
 temp=root->right;
 free(root);
 return temp;
 }
 if(root->right == NULL)
 {
 temp=root->left;
 free(root);
 return temp;
 }
```

```
 temp=inordersuccessor(root->right);
 root->data=temp->data;
 root->right=deletenode(root->right,temp->data);
 }
 return root;
}
int main()
{
 struct node * root = NULL;
 int ch,item,key;
 for(;;)
 {
 printf("\n 1. Insert");
 printf("\n 2. Preorder");
 printf("\n 3. Inorder");
 printf("\n 4. Postorder");
 printf("\n 5. Delete");
 printf("\n 6. Exit");
 printf("\n Read ur choice:");
 scanf("%d",&ch);
 switch(ch)
 {
 case 1:printf("\n Read element to be inserted :");
 scanf("%d",&item);
 root=Insertbst(root,item);
 break;
 case 2:printf("\n The Preorder traversal is\n");
 preorder(root);
break;
 case 3:printf("\n The Inorder traversal is\n");
```

```
 inorder(root);
break;
 case 4:printf("\n The Postorder traversal is\n");
 postorder(root);
break;
 case 5:printf("\n Read node to be deleted : ");
 scanf("%d",&key);
 root=deletenode(root,key);
break;
 default :exit(0);
 }
 }
 return 0;
}
```