1)

```
#include <stdio.h>
#include <string.h>
#define SIZE 50
int top=-1;
char list[SIZE],a;
int i;
char stack[SIZE];
void push(char a){
    if (top== SIZE-1){
        printf("stack over flow");
    }
    else{
        top=top+1;
        stack[top]= a;
    }

}
char reverse(int i){
    if (top==-1){
        printf("stack is under flow");
    }
    else{
        a =stack[top];
        top=top-1;
    }
    return a;
}
int main()
{
    char list[SIZE];
    printf("enter a string:");
    scanf("%s",list);

    for(int i=0;i<strlen(list);i++){
        push(list[i]);
    }
    for(int i=0;i<strlen(list);i++){
        list[i]=reverse(i);
    }

printf("Reversed String is: %s\n",list);
return 0;
}
```

2)

```
#include <stdio.h>
#include <ctype.h>
#define SIZE 20
char stack[SIZE];
int top=-1,k;
char push(char var)
{
   if (top== SIZE-1){
      printf("stack is over flow");
   }
else{
   stack[++top]=var;
}
}

char pop()
{
if (top==-1){
      printf("stack is underflow");
   }
   else{

   return(stack[top--]);
}
}

int infix_to_postfix(char operation)
{

        if(operation == '^')
        {
                return(3);
        }
        else if(operation == '*' || operation == '/')
        {
                return(2);
        }
        else if(operation == '+' || operation == '-')
        {
                return(1);
        }
        else
        {
                return(0);
        }
```

```
}
int main()
{
    char infix[50],postfix[50],ch,var;
    int i=0,k=0;
    printf("Enter Infix to convert it to postfix : ");
    scanf("%s",infix);
    push('#');
    while( (ch=infix[i++]) != '\0')
    {
        if( ch == '('){
            push(ch);
        }
        else{
            if(isalnum(ch)){
                postfix[k++]=ch;
            }
            else{
                if( ch == ')')
                {
                    while( stack[top] != '(')
                        postfix[k++]=pop();
                    var=pop();
                }
                else
                {
                    while( infix_to_postfix(stack[top]) >= infix_to_postfix(ch) )
                        postfix[k++]=pop();
                    push(ch);
                }
            }

        }
    }
    while( stack[top] != '#'){
        postfix[k++]=pop();
    }
    postfix[k]='\0';
    printf("\nPostfix Expression =  %s\n",postfix);
    return 0;
}
```

3)

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 50
int enq();
int deq();
int push_1(int);
int push_2(int);
int pop1();
int pop2();
int show();
int create();
int stack1[SIZE], stack2[SIZE];
int top1 = -1, top2 = -1, count = 0;
int main()
{
   int choice;
   printf("\nQUEUE USING STACKS IMPLEMENTATION\n\n");
   printf("\n1.ENQUEUE");
   printf("\n2.DEQUEUE");
   printf("\n3.DISPLAY");
   printf("\n4.EXIT");
   printf("\n");
   create();
   while (1)
   {
      printf("\nEnter your choice : ");
      scanf("%d", &choice);
      switch (choice)
      {
         case 1:
            enq();
            break;
         case 2:
            deq();
            break;
         case 3:
            show();
            break;
         case 4:
            exit(0);
         default:
            printf("\nInvalid Choice\n");
      }}
return 0;
}
```

```
int create()
{
   top1 = top2 = -1;
return 0;
}
int enq()
{
   int data, i;
   printf("Enter the data : ");
   scanf("%d", &data);
   push_1(data);
   count++;
return 0;
}
int deq()
{
   int i;
   for (i = 0;i <= count;i++)
   {
      push_2(pop1());
   }
   pop2();
   count--;
   for (i = 0;i <= count;i++)
   {
      push_1(pop2());
return 0;
}}
int push_1(int val)
{
   stack1[++top1] = val;
return 0;
}
int push_2(int val)
{
   stack2[++top2] = val;
return 0;
}
int pop1()
{
   return(stack1[top1--]);
}

int pop2()
{
   return(stack2[top2--]);
```

```
}
int show()
{
   int i;
   if(top1 == -1)
   {
      printf("\nEMPTY QUEUE\n");
   }
   else
   {
      printf("\nQUEUE ELEMENTS : ");
      for (i = 0;i <= top1;i++)
      {
         printf(" %d ", stack1[i]);
      }
      printf("\n");
return 0;
}}
```

4)

```
#include <stdio.h>

#include <stdlib.h>

struct binary_tree_node

{

int value;

struct binary_tree_node *left;

struct binary_tree_node *right;

}*root = NULL, *temp = NULL, *t2, *t1;

void delete1();

void insert();

void delete();

void create();

void search(struct binary_tree_node *t);

void search1(struct binary_tree_node *t,int data);

int smallest(struct binary_tree_node *t);
```

```
int largest(struct binary_tree_node *t);

int flag = 1;

void main()

{

int ch;

printf("\nOPERATIONS ---");

printf("\n1 - Insert elements\n");

printf("2 - Delete an element\n");

printf("3 - Exit\n");while(1)

{

printf("\nEnter your option : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

insert();

break;

case 2:

delete();

break;

case 3:

exit(0);

default :

printf("Wrong option, Please enter correct option ");

break;

}

}
```

```
}

void insert()
{
create();
if (root == NULL)
root = temp;
else
search(root);
}
void create()
{
int data;
printf("Enter data of node to be inserted : ");
scanf("%d", &data);
temp = (struct binary_tree_node *)malloc(1*sizeof(struct binary_tree_node));
temp->value = data;
temp->left = temp->right = NULL;
}
void search(struct binary_tree_node *t)
{
if ((temp->value > t->value) && (t->right != NULL))
search(t->right);
else if ((temp->value > t->value) && (t->right == NULL))t->right = temp;
else if ((temp->value < t->value) && (t->left != NULL))
search(t->left);
else if ((temp->value < t->value) && (t->left == NULL))
```

```
t->left = temp;
}
void delete()
{
int data;
if (root == NULL)
{
printf("No elements in a tree to delete");
return;
}
printf("Enter the data to be deleted : ");
scanf("%d", &data);
t1 = root;
t2 = root;
search1(root, data);
}
void search1(struct binary_tree_node *t, int data)
{
if ((data>t->value))
{
t1 = t;
search1(t->right, data);
}
else if ((data < t->value))
{
t1 = t;
search1(t->left, data);
```

```
}
else if ((data==t->value))
{
delete1(t);
}
}
void delete1(struct binary_tree_node *t)
{
int k;
{
if (t1->left == t)
{
t1->left = NULL;
}
else
{
t1->right = NULL;
}
t = NULL;
free(t);
return;
}
else if ((t->right == NULL))
{
if (t1 == t)
{
root = t->left;
```

```
t1 = root;
}
else if (t1->left == t)
{
t1->left = t->left;
}
else
{
t1->right = t->left;
}
t = NULL;
free(t);
return;
}
else if (t->left == NULL)
{
if (t1 == t)
{
root = t->right;
t1 = root;
}
else if (t1->right == t)
t1->right = t->right;
else
t1->left = t->right;t == NULL;
free(t);
return;
```

```
}
else if ((t->left != NULL) && (t->right != NULL))
{
t2 = root;
if (t->right != NULL)
{
k = smallest(t->right);
flag = 1;
}
else
{
k =largest(t->left);
flag = 2;
}
search1(root, k);
t->value = k;
}
}
int smallest(struct binary_tree_node *t)
{
t2 = t;
if (t->left != NULL)
{
t2 = t;
return(smallest(t->left));
}
else
```

```
return (t->value);

}

int largest(struct binary_tree_node *t)

{

if (t->right != NULL)

{

t2 = t;

return(largest(t->right));

}

else

return(t->value);

}
```