1) Code

```
#include <stdio.h>
#include <string.h>
#define SIZE 300
int top=-1;
char character[SIZE],toy;
int i;
char stack[SIZE];
void push(char toy){
    if (top== SIZE-1){
        printf("stack is full");
    }
    else{
        top=top+1;
        stack[top]= toy;
    }

}
char reverse(int i){
    if (top==-1){
        printf("stack is underflow");
    }
    else{
        toy =stack[top];
        top=top-1;
    }
    return toy;
}
int main()
{
    char character[SIZE];
    printf("pleas enter a string to reverse it :::");
    scanf("%s",character);

    for(int i=0;i<strlen(character);i++){
        push(character[i]);
    }
    for(int i=0;i<strlen(character);i++){
        character[i]=reverse(i);
    }
    printf("Reversed String is: %s\n",character);

    return 0;
}
```

2) Code

```
#include <stdio.h>
#include <ctype.h>
#define SIZE 300
char stack[SIZE];
int top=-1,k;
char push(char var)
{
    if (top== SIZE-1){
        printf("stack is full");
    }
else{
    stack[++top]=var;
}
}

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

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

int post_from_int(char operatation)
{

            if(operatation == '^')
            {
                        return(3);
            }
            else if(operatation == '*' || operatation == '/')
            {
                        return(2);
            }
            else if(operatation== '+' || operatation == '-')
            {
                        return(1);
            }
            else
            {
                        return(0);
            }
}
int main()
{
    char infix[50],postfix[50],ch,var;
    int i=0,k=0;
    printf("Enter Infix charater to convert it to postfix charater : ");
    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( post_from_int(stack[top]) >= post_from_int(ch) )
                        postfix[k++]=pop();
                    push(ch);
                }
            }

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

    return 0;
}
```

3) Code

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 100
int push1(int);
int push2(int);
int pop1();
int pop2();
int enq();
int deq();
int display();
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:
                display();
                break;
            case 4:
                exit(0);
            default:
                printf("\nInvalid Choice\n");
        }}
return 0;
}
int create()
{
    top1 = top2 = -1;
return 0;
}
int push1(int val)
{
    stack1[++top1] = val;
return 0;
}
int pop1()
{
    return(stack1[top1--]);
}
int push2(int val)
{
    stack2[++top2] = val;
return 0;
}
int pop2()
{
    return(stack2[top2--]);
}
int enq()
{
    int data, i;
    printf("Enter the data : ");
    scanf("%d", &data);
    push1(data);
    count++;
return 0;

}
int deq()
{
    int i;
    for (i = 0;i <= count;i++)
    {
        push2(pop1());
    }
    pop2();
    count--;
    for (i = 0;i <= count;i++)
    {
        push1(pop2());
return 0;
    }}
int display()
{
    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) Code

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

struct btnode
{
    int value;
    struct btnode *leaf;
    struct btnode *r;
}*root = NULL, *temp = NULL, *t2, *t1;

void delete1();
void insert();
void delete();
void create();
void display(struct btnode *t);
void search(struct btnode *t);
void search1(struct btnode *t,int data);
int find_small(struct btnode *t);
int find_large(struct btnode *t);
int flag = 1;

int main()
{
    int choice;

    printf("\nOPERATIONS ---");
    printf("\n1 - Insert an element into tree\n");
    printf("2 - Delete an element from the tree\n");
    printf("3 - display\n");
    printf("4-Exit\n");
    while(1)
    {
        printf("\nEnter your choice : ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            insert();
            break;
        case 2:
            delete();
            break;
        case 3:
            display(root);
            break;
        case 4:
            exit(0);
        default :
            printf("Wrong choice, Please enter correct choice  ");
            break;
        }
    }

    return 0;
}
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 btnode *)malloc(1*sizeof(struct btnode));
    temp->value = data;
    temp->leaf = temp->r = NULL;
}
void search(struct btnode *t)
{
    if ((temp->value > t->value) && (t->r != NULL))
        search(t->r);
    else if ((temp->value > t->value) && (t->r == NULL))
        t->r = temp;
    else if ((temp->value < t->value) && (t->leaf != NULL))
        search(t->leaf);
    else if ((temp->value < t->value) && (t->leaf == NULL))
        t->leaf = 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 btnode *t, int data)
{
    if ((data>t->value))
    {
        t1 = t;
        search1(t->r, data);
    }
    else if ((data < t->value))
    {
        t1 = t;
        search1(t->leaf, data);
    }
    else if ((data==t->value))
    {
        delete1(t);
    }
}
void delete1(struct btnode *t)
{
    int k;
    if ((t->leaf == NULL) && (t->r == NULL))
    {
        if (t1->leaf == t)
        {
            t1->leaf = NULL;
        }
        else
        {
            t1->r = NULL;
        }
        t = NULL;
        free(t);
        return;
    }
    else if ((t->r == NULL))
```

```
{
    if (t1 == t)
    {
        root = t->leaf;
        t1 = root;
    }
    else if (t1->leaf == t)
    {
        t1->leaf = t->leaf;

    }
    else
    {
        t1->r = t->leaf;
    }
    t = NULL;
    free(t);
    return;
}
else if (t->leaf == NULL)
{
    if (t1 == t)
    {
        root = t->r;
        t1 = root;
    }
    else if (t1->r == t)
        t1->r = t->r;
    else
        t1->leaf = t->r;
    t == NULL;
    free(t);
    return;
}
else if ((t->leaf != NULL) && (t->r != NULL))
{
    t2 = root;
    if (t->r != NULL)
    {
        k = find_small(t->r);
        flag = 1;
    }
    else
```

```
        {
            k =find_large(t->leaf);
            flag = 2;
        }
        search1(root, k);
        t->value = k;
    }
}
int find_small(struct btnode *t)
{
    t2 = t;
    if (t->leaf != NULL)
    {
        t2 = t;
        return(find_small(t->leaf));
    }
    else
        return (t->value);
}
int find_large(struct btnode *t)
{
    if (t->r != NULL)
    {
        t2 = t;
        return(find_large(t->r));
    }
    else{
        return(t->value);
    }
}
void display(struct btnode *t)
{
    if (root == NULL)
    {
        printf("No elements in a tree to display");
        return;
    }
    if (t->leaf != NULL)
        display(t->leaf);
    printf("%d -> ", t->value);
    if (t->r != NULL){
        display(t->r);
    }
```

```
}
```