1. write a c program to reverse a string using stack.

```
include<stdio.h>
#include<string.h>
#define MAX 20
int top = -1;
char stack[MAX];
char push(char item)
{
if(top == (MAX-1))
printf("Stack Overflow\n");
else
stack[++top] =item;
}
char pop()
{
if(top == -1)
printf("Stack Underflow\n");
else
return stack[top--];
}
main()
{
char str[20];
int i;
printf("Enter the string : " );
gets(str);
for(i=0;i<strlen(str);i++)
push(str[i]);
for(i=0;i<strlen(str);i++)
str[i]=pop();
printf("Reversed string is : ");
puts(str);
}
```

2.write a program for Infix To Postfix Conversion Using Stack.

```
#include<stdio.h>
char stack[20];
int top = -1;
void push(char x)
```

```
{
 stack[++top] = x;
}
char pop()
{
 if(top == -1)
 return -1;
 else
 return stack[top--];
}
int priority(char x)
{
 if(x == '(')
 return 0;
 if(x == '+' || x == '-')
 return 1;
 if(x == '*' || x == '/')
 return 2;
}
main()
{
 char exp[20];
 char *e, x;
 printf("Enter the expression :: ");
scanf("%s",exp);
 e = exp;
 while(*e != '\0')
 {
 if(isalnum(*e))
 printf("%c",*e);
 else if(*e == '(')
 push(*e);
 else if(*e == ')')
 {
 while((x = pop()) != '(')
 printf("%c", x);
 }
 else
 {
 while(priority(stack[top]) >= priority(*e))
 printf("%c",pop());
 push(*e);
 }
```

```
 e++;
 }
 while(top != -1)
 {
 printf("%c",pop());
 }
}
```

3.write a C Program to Implement Queue Using Two Stacks.

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
   int data;
   struct node *next;
};
void push(struct node** top, int data);
int pop(struct node** top);
struct queue
{
   struct node *stack1;
   struct node *stack2;
};
void enqueue(struct queue *q, int x)
{
   push(&q->stack1, x);
}
void dequeue(struct queue *q)
{
   int x;
   if (q->stack1 == NULL && q->stack2 == NULL) {
```

```
        printf("queue is empty");
        return;
    }
    if (q->stack2 == NULL) {
        while (q->stack1 != NULL) {
        x = pop(&q->stack1);
        push(&q->stack2, x);
        }
    }
    x = pop(&q->stack2);
    printf("%d\n", x);
}
void push(struct node** top, int data)
{
    struct node* newnode = (struct node*) malloc(sizeof(struct node));
        if (newnode == NULL) {
            printf("Stack overflow \n");
            return;
        }
    newnode->data = data;
    newnode->next = (*top);
    (*top) = newnode;
}
int pop(struct node** top)
{
    int buff;
    struct node *t;
```

```
    if (*top == NULL) {
        printf("Stack underflow \n");

    }
    else {
        t = *top;
        buff = t->data;
        *top = t->next;
        free(t);
        return buff;
    }
}
void display(struct node *top1,struct node *top2)
{
    while (top1 != NULL) {
        printf("%d\n", top1->data);
        top1 = top1->next;
    }
    while (top2 != NULL) {
        printf("%d\n", top2->data);
        top2 = top2->next;
    }
}
int main()
{
    struct queue *q = (struct queue*)malloc(sizeof(struct queue));
    int f = 0, a;
```

```
    char ch = 'y';
    q->stack1 = NULL;
    q->stack2 = NULL;
    while (ch == 'y'||ch == 'Y') {
        printf("enter ur choice\n1.add to queue\n2.remove \n3.display\n4.exit\n");
        scanf("%d", &f);
        switch(f) {
            case 1 : printf("enter the element to be added to queue\n");
                     scanf("%d", &a);
                     enqueue(q, a);
                     break;
            case 2 : dequeue(q);
                     break;
            case 3 : display(q->stack1, q->stack2);
                     break;
            case 4 : exit(1);
                     break;
            default : printf("invalid\n");
                      break;
        }
    }
}
```

4.write a c program for insertion and deletion of BST.

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

struct node
{
        int info;
        struct node *leftchild;
```

```
        struct node *rightchild;
};
struct node*root;



void find(int item,struct node **sa,struct node **loc)
{
        struct node *p,*ptr;

        if(root==NULL)
        {
                *loc=NULL;
                *sa=NULL;
                return;
        }
        if(item==root->info)
        {
                *loc=root;
                *sa=NULL;
                return;
        }

        if(item<root->info)
                p=root->leftchild;
        else
                p=root->rightchild;
        ptr=root;

        while(p!=NULL)
        {
                if(item==p->info)
                {       *loc=p;
                        *sa=ptr;
                        return;
                }
                ptr=p;
                if(item<p->info)
                        p=p->leftchild;
                else
                        p=p->rightchild;
         }
         *loc=NULL;
```

```
        *sa=ptr;
}

void insert(int item)
{       struct node *tmp,*parent,*location;
        find(item,&parent,&location);
        if(location!=NULL)
        {
                printf("Item already present");
                return;
        }

        tmp=(struct node *)malloc(sizeof(struct node));
        tmp->info=item;
        tmp->leftchild=NULL;
        tmp->rightchild=NULL;

        if(parent==NULL)
                root=tmp;
        else
                if(item<parent->info)
                        parent->leftchild=tmp;
                else
                        parent->rightchild=tmp;
}


void case_a(struct node *arg,struct node *loc )
{
        if(arg==NULL)
                root=NULL;
        else
                if(loc==arg->leftchild)
                        arg->leftchild=NULL;
                else
                        arg->rightchild=NULL;
}

void case_b(struct node *par,struct node *loc)
{
        struct node *child;
```

```
        if(loc->leftchild!=NULL)
                child=loc->leftchild;
        else
                child=loc->rightchild;

        if(par==NULL )
                root=child;
        else
                if( loc==par->leftchild)
                        par->leftchild=child;
                else
                        par->rightchild=child;
}

void case_c(struct node *par,struct node *loc)
{
        struct node *p,*ptr,*suc,*parsuc;


        ptr=loc;
        p=loc->rightchild;
        while(p->leftchild!=NULL)
        {
                ptr=p;
                p=p->leftchild;
        }
        suc=p;
        parsuc=ptr;

        if(suc->leftchild==NULL && suc->rightchild==NULL)
                case_a(parsuc,suc);
        else
                case_b(parsuc,suc);

        if(par==NULL)
                root=suc;
        else
                if(loc==par->leftchild)
                        par->leftchild=suc;
                else
                        par->rightchild=suc;

        suc->leftchild=loc->leftchild;
```

```
        suc->rightchild=loc->rightchild;
}
int del(int item)
{
        struct node *parent,*location;
        if(root==NULL)
        {
                printf("Tree is empty");
                return 0;
        }

        find(item,&parent,&location);
        if(location==NULL)
        {
                printf("Item not present");
                return 0;
        }

        if(location->leftchild==NULL && location->rightchild==NULL)
                case_a(parent,location);
        if(location->leftchild!=NULL && location->rightchild==NULL)
                case_b(parent,location);
        if(location->leftchild==NULL && location->rightchild!=NULL)
                case_b(parent,location);
        if(location->leftchild!=NULL && location->rightchild!=NULL)
                case_c(parent,location);
        free(location);
}
void main()
{
        int choice,num;
        root=NULL;
        while(1)
        {
                printf("\n");
                printf("1.Insert\n");
                printf("2.Delete\n");
                printf("Enter your choice : ");
                scanf("%d",&choice);

                switch(choice)
                {
                 case 1:
```

```
                printf("Enter the number to be inserted : ");
                scanf("%d",&num);
                insert(num);
                break;
            case 2:
                printf("Enter the number to be deleted : ");
                scanf("%d",&num);
                del(num);
                break;
            case 3:
        break;
            default:
                printf("Wrong choice\n");
            }
        }
}
```