1) Write a c program to reverse a string using stack?

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

#define max 100
int top,stack[max];

void push(char x){

      // Push(Inserting Element in stack) operation
      if(top == max-1){
          printf("stack overflow");
      }  else {
          stack[++top]=x;
      }

}

void pop(){
    // Pop (Removing element from stack)
      printf("%c",stack[top--]);
}


main()
{
   char str[]="sri lanka";
   int len = strlen(str);
   int i;

   for(i=0;i<len;i++)
        push(str[i]);

   for(i=0;i<len;i++)
      pop();
}
```

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>
void push1(int);
void push2(int);
int pop1();
int pop2();
void enqueue();
void dequeue();
void display();
void create();
int stack1[100], stack2[100];
int top1 = -1, top2 = -1;
int count = 0;

/* Main Function */
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:
                enqueue();
                break;
            case 2:
                dequeue();
```

```
            break;
        case 3:
            display();
            break;
        case 4:
            exit(0);
        default:
            printf("\nInvalid Choice\n");
    }}}

/* Function to initialize top of two stacks*/
void create()
{
   top1 = top2 = -1;
}

/* Function to push an element to stack */
void push1(int element)
{
   stack1[++top1] = element; // Pushing the element to stack1
}

/* Function to pop element from stack */
int pop1()
{
   return(stack1[top1--]); // Pop element from stack1
}

/* Function to push an element on to stack */
void push2(int element)
{
   stack2[++top2] = element; // Pushing the element to stack2
}

/* Function to pop an element from stack */
int pop2()
{
   return(stack2[top2--]); // pop element from stack2
}

/* Function to enqueue an element into the queue using stack */
void enqueue()
{
   int data, i;
   printf("Enter the data : ");
   scanf("%d", &data);
```

```
    push1(data); // Push data from stack to the queue
    count++;
}

/* Function to dequeue an element from the queue using stack */
void dequeue()
{
    int i;
    for (i = 0;i <= count;i++)
    {
        push2(pop1()); // Pop elements from stack1 and push them to stack2
    }
    pop2(); // Pop the element from stack2 which is the element to be dequeued
    count--;
    for (i = 0;i <= count;i++)
    {
        push1(pop2()); // Push back all the elements from stack2 to stack1
    }}

/Function to display the elements in the queue/
void 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");
}}
```

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

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

struct node
{
```

```
        int info;
        struct node *lchild;
        struct node *rchild;
}*root;



void find(int item,struct node **par,struct node **loc)
{
        struct node *ptr,*ptrsave;

        if(root==NULL)  /tree empty/
        {
                *loc=NULL;
                *par=NULL;
                return;
        }
        if(item==root->info) /item is at root/
        {
                *loc=root;
                *par=NULL;
                return;
        }
        /Initialize ptr and ptrsave/
        if(item<root->info)
                ptr=root->lchild;
        else
                ptr=root->rchild;
        ptrsave=root;

        while(ptr!=NULL)
        {
                if(item==ptr->info)
                {       *loc=ptr;
                        *par=ptrsave;
                        return;
                }
                ptrsave=ptr;
                if(item<ptr->info)
                        ptr=ptr->lchild;
                else
                        ptr=ptr->rchild;
         }/*End of while */
         loc=NULL;   /*item not found/
         *par=ptrsave;
}/End of find()/
```

```
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->lchild=NULL;
        tmp->rchild=NULL;

        if(parent==NULL)
                root=tmp;
        else
                if(item<parent->info)
                        parent->lchild=tmp;
                else
                        parent->rchild=tmp;
}/End of insert()/


void case_a(struct node *par,struct node *loc )
{
        if(par==NULL) /item to be deleted is root node/
                root=NULL;
        else
                if(loc==par->lchild)
                        par->lchild=NULL;
                else
                        par->rchild=NULL;
}/End of case_a()/

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

        /Initialize child/
        if(loc->lchild!=NULL) /*item to be deleted has lchild */
                child=loc->lchild;
        else            /*item to be deleted has rchild */
                child=loc->rchild;
```

```
        if(par==NULL )   /Item to be deleted is root node/
                root=child;
        else
                if( loc==par->lchild)   /item is lchild of its parent/
                        par->lchild=child;
                else                    /item is rchild of its parent/
                        par->rchild=child;
}/End of case_b()/

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

        /Find inorder successor and its parent/
        ptrsave=loc;
        ptr=loc->rchild;
        while(ptr->lchild!=NULL)
        {
                ptrsave=ptr;
                ptr=ptr->lchild;
        }
        suc=ptr;
        parsuc=ptrsave;

        if(suc->lchild==NULL && suc->rchild==NULL)
                case_a(parsuc,suc);
        else
                case_b(parsuc,suc);

        if(par==NULL) /*if item to be deleted is root node */
                root=suc;
        else
                if(loc==par->lchild)
                        par->lchild=suc;
                else
                        par->rchild=suc;

        suc->lchild=loc->lchild;
        suc->rchild=loc->rchild;
}/End of case_c()/
int del(int item)
{
        struct node *parent,*location;
        if(root==NULL)
        {
                printf("Tree empty");
```

```
                return 0;
        }

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

        if(location->lchild==NULL && location->rchild==NULL)
                case_a(parent,location);
        if(location->lchild!=NULL && location->rchild==NULL)
                case_b(parent,location);
        if(location->lchild==NULL && location->rchild!=NULL)
                case_b(parent,location);
        if(location->lchild!=NULL && location->rchild!=NULL)
                case_c(parent,location);
        free(location);
}/End of del()/
void display(struct node *ptr,int level)
{
        int i;
        if ( ptr!=NULL )
        {
                display(ptr->rchild, level+1);
                printf("\n");
                for (i = 0; i < level; i++)
                        printf("    ");
                printf("%d", ptr->info);
                display(ptr->lchild, level+1);
        }/End of if/
}/End of display()/
main()
{
        int choice,num;
        root=NULL;
        while(1)
        {
                printf("\n");
                printf("1.Insert\n");
                printf("2.Delete\n");
                printf("3.Display\n");
                printf("4.Quit\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:
                        display(root,1);
                        break;
                 case 4:
        exit(0);
                 default:
                        printf("Wrong choice\n");
                }/*End of switch */
        }/*End of while */
}/End of main()/
```