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.

```
/* C program to implement queues 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");
        return;
    }
    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
            from queue\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 *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()*/

int preorder(struct node *ptr)
{
    if(root==NULL)
    {
        printf("Tree is empty");
        return 0;
    }
    if(ptr!=NULL)
    {
        printf("%d  ",ptr->info);
        preorder(ptr->lchild);
        preorder(ptr->rchild);
    }
}/*End of preorder()*/
```

```
void inorder(struct node *ptr)
{
   if(root==NULL)
   {
      printf("Tree is empty");
      return;
   }
   if(ptr!=NULL)
   {
      inorder(ptr->lchild);
      printf("%d  ",ptr->info);
      inorder(ptr->rchild);
   }
}/*End of inorder()*/

void postorder(struct node *ptr)
{
   if(root==NULL)
   {
      printf("Tree is empty");
      return;
   }
   if(ptr!=NULL)
   {
      postorder(ptr->lchild);
      postorder(ptr->rchild);
      printf("%d  ",ptr->info);
```

```
    }
}/*End of postorder()*/

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.Inorder Traversal\n");
        printf("4.Preorder Traversal\n");
```

```
printf("5.Postorder Traversal\n");
printf("6.Display\n");
printf("7.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:
   inorder(root);
   break;
 case 4:
   preorder(root);
   break;
 case 5:
   postorder(root);
   break;
```

```
        case 6:
          display(root,1);
          break;
        case 7:
          break;
        default:
          printf("Wrong choice\n");
       }/*End of switch */
    }/*End of while */
}/*End of main()*/
```