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

A

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
char stack[50];
int top=-1;
void push(char c)
{
   top=top-1;
   stack[top]=c;
}
void pop()
{
   char c;
   c=stack[top];
   top=top+1;
   printf("%c",c);
}
int main()

{
   char str[30];
   int i,len;
   printf("enter string\n");
   scanf("%s",str);
   len=strlen(str);
   for (i=0;i<len;i++)
   {
      push(str[i]);
   }
   printf("reverse of a string is: \n");
   for (i=0;i<len;i++)
   {
      pop();
   }
   return 0;
}
```

OUTPUT:

enter string

```
jayanth
reverse of a string is:
htnayaj

...Program finished with exit code 0
Press ENTER to exit console.
```

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

```
#include<stdio.h>
#include<stdlib.h>
#define max 25
int stack[max];
int top=-1;
void push(char symbol)
{
   top=top+1;
   stack[top]=symbol;
}
char pop()
{
   char k;
   k=stack[top];
   top=top-1;
   return k;
}
int isoperand(char symbol)
{
   if(symbol>='a'&&symbol<='z')
     return 1;
   else
     return 0;
}
int isoperator(char symbol)
{
   if(symbol=='+'||symbol=='-'||symbol=='*'||symbol=='/')
     return 1;
   else
     return 0;
}
int precedence(char symbol)
```

```
{
    int result;
    switch(symbol)
    {
        case '(':result=0;break;
        case '+':
        case '-':result=1;break;
        case '*':
        case '/':result=2;break;
    }
    return result;
}
void main()
{
    char infix[max],postfix[max],temp;
    char symbol;
    int i,j;
    i=j=0;
    printf("enter an infix expression\n");
    gets(infix);
    push('(');
    while(infix[i]!='\0')
    {
        symbol=infix[i];
        if(isoperand(symbol))
        {
            postfix[j]=symbol;
            j=j+1;
        }
        if(symbol=='(')
        {
            push(symbol);
        }
        if(isoperator(symbol))
        {
            while(precedence(stack[top])>=precedence(symbol))
            {
                temp=pop();
                postfix[j]=temp;
                j=j+1;
            }
            push(symbol);
        }
```

```
        if(symbol==')')
        {
            while(stack[top]!='(')
            {
                temp=pop();
                postfix[i]=temp;
                j=j+1;
            }
            temp=pop();
        }
        i=i+1;
    }
    while(stack[top]!='(')
    {
        temp=pop();
        postfix[j]=temp;
        j=j+1;

    }
    postfix[j]='\0';
    printf("%s\n",postfix);
}
```

OUTPUT:

```
enter an infix expression
a+b
ab+


...Program finished with exit code 0
Press ENTER to exit console.
```

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 st1[100], st2[100];
int top1 = -1, top2 = -1;
int count = 0;

void main()
{
   int ch;

   printf("\n1 - Enqueue element into queue");
   printf("\n2 - Dequeu element from queue");
   printf("\n3 - Display from queue");
   printf("\n4 - Exit");
   create();
   while (1)
   {
      printf("\nEnter choice");
      scanf("%d", &ch);
      switch (ch)
      {
      case 1:
         enqueue();
         break;
      case 2:
         dequeue();
         break;
      case 3:
         display();
         break;
      case 4:
         exit(0);
      default:
         printf("Wrong choice");
      }
   }
}
void create()
{
   top1 = top2 = -1;
}
```

```
void push1(int data)
{
   st1[++top1] = data;
}
int pop1()
{
   return(st1[top1--]);
}
void push2(int data)
{
   st2[++top2] = data;
}
int pop2()
{
   return(st2[top2--]);
}
 void enqueue()
{
   int data, i;
   printf("Enter data into queue");
   scanf("%d", &data);
   push1(data);
   count++;
}
void dequeue()
{
   int i;
   for (i = 0;i <= count;i++)
   {
      push2(pop1());
   }
   pop2();
   count--;
   for (i = 0;i <= count;i++)
   {
      push1(pop2());
   }
}
void display()
{
   int i;
   for (i = 0;i <= top1;i++)
   {
```

```
        printf(" %d ", st1[i]);
    }
}
```

OUTPUT:

```
1 - Enqueue element into queue
2 - Dequeu element from queue
3 - Display from queue
4 - Exit
Enter choice1
Enter data into queue55


Enter choice1
Enter data into queue45

Enter choice3
 55  45
Enter choice2
Enter choice3
 45
```

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

```
 #include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node *right;
    struct node *left;
};
struct node* find_min(struct node *root)
{
    if(root == NULL)
        return NULL;
    else if(root->left!= NULL)
        return find_min(root->left);
    return root;
}
struct node* new_node(int x)
{
    struct node *p;
    p = malloc(sizeof(struct node));
```

```
    p->data = x;
    p->left = NULL;
    p->right = NULL;
    return p;
}
struct node* insert(struct node *root, int x)
{
    if(root==NULL)
        return new_node(x);
    else if(x>root->data)
    {
        root->right = insert(root->right, x);
    }
    else if(x<root->data)
    {
        root->left = insert(root->left,x);
    }
    return root;
}

struct node* delete(struct node *root, int x)
{

    if(root==NULL)
        return NULL;
    if (x>root->data)
        root->right = delete(root->right, x);
    else if(x<root->data)
        root->left=delete(root->left, x);
    else

         {
        if(root->left==NULL && root->right==NULL)
        {
            free(root);
            return NULL;
        }


        else if(root->left==NULL || root->right==NULL)
        {
            struct node *temp;
            if(root->left==NULL)
```

```
            temp = root->right;
        else
            temp = root->left;
        free(root);
        return temp;
      }

     else
      {
        struct node *temp = find_min(root->right);
        root->data = temp->data;
        root->right = delete(root->right, temp->data);
      }
   }
   return root;
}

void inorder(struct node *root)
{
   if(root!=NULL)
   {
      inorder(root->left);
      printf(" %d-> ", root->data);
      inorder(root->right);
   }
}

int main()
{

   struct node *root;
   root = new_node(10);
   insert(root,20);
   insert(root,30);
   insert(root,40);
   insert(root,50);
   insert(root,60);
   insert(root,70);
   insert(root,80);
   insert(root,90);
   insert(root,100);
   insert(root,200);
   insert(root,300);
```

```
    printf("inorder traversal before deletion\n");
    inorder(root);
    root = delete(root, 100);
    root = delete(root, 40);
    root = delete(root, 50);
    root = delete(root, 10);
    printf("\ninorder traversalafter deletion\n");
    inorder(root);
    return 0;
}
```

Output:

Language
main.c

input
inorder traversal before deletion
10-> 20-> 30-> 40-> 50-> 60-> 70-> 80-> 90-> 100-> 200-> 300->
inorder traversalafter deletion
20-> 30-> 60-> 70-> 80-> 90-> 200-> 300->