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

```
/*C program to Reverse String using STACK*/

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

#define MAX 100       /*maximum no. of characters*/

/*stack variables*/
int top=-1;
int item;
/***************/

/*string declaration*/
char stack_string[MAX];

/*function to push character (item)*/
void pushChar(char item);

/*function to pop character (item)*/
char popChar(void);

/*function to check stack is empty or not*/
int isEmpty(void);

/*function to check stack is full or not*/
int isFull(void);

int main()
{
   char str[MAX];

   int i;

   printf("Input a string: ");
   scanf("%[^\n]s",str); /*read string with spaces*/
   /*gets(str);-can be used to read string with spaces*/

   for(i=0;i<strlen(str);i++)
      pushChar(str[i]);
```

```
    for(i=0;i<strlen(str);i++)
        str[i]=popChar();

    printf("Reversed String is: %s\n",str);

    return 0;
}

/*function definition of pushChar*/
void pushChar(char item)
{
    /*check for full*/
    if(isFull())
    {
        printf("\nStack is FULL !!!\n");
        return;
    }

    /*increase top and push item in stack*/
    top=top+1;
    stack_string[top]=item;
}

/*function definition of popChar*/
 char popChar()
{
    /*check for empty*/
    if(isEmpty())
    {
        printf("\nStack is EMPTY!!!\n");
        return 0;
    }

    /*pop item and decrease top*/
    item = stack_string[top];
    top=top-1;
    return item;
}

/*function definition of isEmpty*/
int isEmpty()
{
```

```
   if(top==-1)
      return 1;
   else
      return 0;
}

/*function definition of isFull*/
int isFull()
{
   if(top==MAX-1)
      return 1;
   else
      return 0;
}
```

Output:

Input string:swetha

Reverse string is :ahtews

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

```
#include<stdio.h>
#include<stdlib.h>      /* for exit() */
#include<ctype.h>     /* for isdigit(char ) */
#include<string.h>

#define SIZE 100
char stack[SIZE];
int top = -1;
void push(char item)
{
        if(top >= SIZE-1)
        {
                printf("\nStack Overflow.");
        }
        else
        {
                top = top+1;
                stack[top] = item;
        }
```

```
}
char pop()
{
	char item ;

	if(top <0)
	{
		exit(1);
	}
	else
	{
		item = stack[top];
		top = top-1;
		return(item);
	}
}
int is_operator(char symbol)
{
	if(symbol == '^' || symbol == '*' || symbol == '/' || symbol == '+' || symbol =='-')
	{
		return 1;
	}
	else
	{
	return 0;
	}
}
int precedence(char symbol)
{
	if(symbol == '^')	{
		return(3);
	}
	else if(symbol == '*' || symbol == '/')
	{
		return(2);
	}
	else if(symbol == '+' || symbol == '-')		{
		return(1);
	}
	else
	{
		return(0);
```

```
        }
}

void InfixToPostfix(char infix_exp[], char postfix_exp[])
{
        int i, j;
        char item;
        char x;

        push('(');
        strcat(infix_exp,")");
        i=0;
        j=0;
        item=infix_exp[i];
        while(item != '\0')
        {
                if(item == '(')
                {
                        push(item);
                }
                else if( isdigit(item) || isalpha(item))
                {
                        postfix_exp[j] = item;                                  j++;
                }
                else if(is_operator(item) == 1)                 {
                        x=pop();
                        while(is_operator(x) == 1 && precedence(x)>= precedence(item))
                        {
                                postfix_exp[j] = x;
                                j++;
                                x = pop();                                      }
                        push(x);
                        push(item);                             }
                else if(item == ')')                    {
                        x = pop();
                        while(x != '(')
                        {
                                postfix_exp[j] = x;
                                j++;
                                x = pop();
                        }
                }
```

```
                else
                { /* if current symbol is neither operand not '(' nor ')' and nor
                        operator */
                        printf("\nInvalid infix Expression.\n");
                        getchar();
                        exit(1);
                }
                i++;


                item = infix_exp[i];
        } /* while loop ends here */
        if(top>0)
        {
                printf("\nInvalid infix Expression.\n");
                getchar();
                exit(1);
        }
        if(top>0)
        {
                printf("\nInvalid infix Expression.\n");
                getchar();
                exit(1);
        }


        postfix_exp[j] = '\0';
}
int main()
{
        printf("\nEnter Infix expression : ");
        gets(infix);

        InfixToPostfix(infix,postfix);
        printf("Postfix Expression: ");
        puts(postfix);
        return 0;
}
```

output:

Enter Infix Expression :(3^2*5)/(3*2-3)+5

Postfix expression :32^5*32*3-5/5

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

```
/* C Program to implement a queue using two stacks */
#include <stdio.h>
#include <stdlib.h>

/* structure of a stack node */
struct sNode {
        int data;
        struct sNode* next;
};

/* Function to push an item to stack*/
void push(struct sNode** top_ref, int new_data);

/* Function to pop an item from stack*/
int pop(struct sNode** top_ref);

/* structure of queue having two stacks */
struct queue {
        struct sNode* stack1;
        struct sNode* stack2;
};
```

```
/* Function to enqueue an item to queue */
void enQueue(struct queue* q, int x)
{
        push(&q->stack1, x);
}

/* Function to deQueue an item from queue */
int deQueue(struct queue* q)
{
        int x;

        /* If both stacks are empty then error */
        if (q->stack1 == NULL && q->stack2 == NULL) {
        printf("Q is empty");
        getchar();
        exit(0);
        }

        /* Move elements from stack1 to stack 2 only if
        stack2 is empty */
        if (q->stack2 == NULL) {
        while (q->stack1 != NULL) {
        x = pop(&q->stack1);
```

```
        push(&q->stack2, x);
        }
        }

        x = pop(&q->stack2);
        return x;
}


/* Function to push an item to stack*/
void push(struct sNode** top_ref, int new_data)
{
        /* allocate node */
        struct sNode* new_node = (struct Node*)malloc(sizeof(struct sNode));
        if (new_node == NULL) {
        printf("Stack overflow \n");
        getchar();
        exit(0);
        }

        /* put in the data */
        new_node->data = new_data;

        /* link the old list off the new node */
```

```
        new_node->next = (*top_ref);

        /* move the head to point to the new node */
        (*top_ref) = new_node;
}

/* Function to pop an item from stack*/
int pop(struct Node** top_ref)
{
        int res;
        struct sNode* top;

        /*If stack is empty then error */
        if (*top_ref == NULL) {
        printf("Stack underflow \n");
        getchar();
        exit(0);
        }
        else {
        top = *top_ref;
        res = top->data;
        *top_ref = top->next;
        free(top);
        return res;
```

```
        }
}


/* Driver function to test above functions */
int main()
{
        /* Create a queue with items 1 2 3*/
        struct queue* q = (struct queue*)malloc(sizeof(struct queue));
        q->stack1 = NULL;
        q->stack2 = NULL;
        enQueue(q, 1);
        enQueue(q, 2);
        enQueue(q, 3);

        /* Dequeue items */
        printf("%d ", deQueue(q));
        printf("%d ", deQueue(q));
        printf("%d ", deQueue(q));

        return 0;
}
```

Output:

1 2 3

4.

```
# 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()*/
```

Output:

```
1.Insert
2.Delete
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 1
Enter the number to be inserted : 323

1.Insert
2.Delete
3.Inorder Traversal
4.Preorder Traversal
```

5.Postorder Traversal

6.Display

7.Quit

Enter your choice : 1

Enter the number to be inserted : 45 35

1.Insert

2.Delete

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Display

7.Quit

Enter your choice : 4

323 435

1.Insert

2.Delete

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Display

7.Quit

Enter your choice : 6

435

323

1.Insert

2.Delete

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Display

7.Quit

Enter your choice :