ASSIGNMENT:- 5

Question 1: Write a C program to reverse a string using stack.

Answer:

```
#include <stdio.h>
#include <string.h>
#define MAX 100

int top=-1;
int alphabet;
char string[MAX];
void push(char alphabet);
char pop(void);

void main()
{
char str[MAX];
int i;
printf("Input the desired string: ");
scanf("%s",str);
for(i=0;i<strlen(str);i++)
{
push(str[i]);
```

```
}
for(i=0;i<strlen(str);i++)
{
str[i]=pop();
}
printf("The reversed string is: %s\n",str);
}

void push(char alphabet)
{
if(top==MAX-1)
{
printf("\nOVERFLOW..!!!\n");
return;
}
top=top+1;
string[top]=alphabet;
}

char pop()
{
if(top==-1)
{
printf("\nUNDERFLOW..!!!\n");
return 0;
}
```

```
alphabet=string[top];
top=top-1;
return alphabet;
}
```

Output:

```
Input the desired string: Suyash

The reversed string is: hsayuS
```

Question 2: Write a program for Infix to Postfix conversion using Stack.

Answer:

```
#include<stdio.h>
#define MAX 100

char stack[MAX];
int top = -1;

void push(char p)
{
stack[++top] = p;
}

char pop()
{
if(top == -1)
return -1;
```

```
else
return stack[top--];
}

int priority(char x)
{
if(x == '(')
return 0;
else if(x == '+' || x == '-')
return 1;
else if(x == '*' || x == '/')
return 2;
}

void main()
{
char expression[20];
char *ep, x;
printf("Enter your expression : ");
scanf("%s",expression);
ep = expression;
printf("Postfix of the expression is \n");
while(*ep != '\0')
{
if(isalnum(*ep))
printf("%c",*ep);
else if(*ep == '(')
```

```
push(*ep);
else if(*ep == ')')
{
while((x = pop()) != '(')
printf("%c", x);
}
else
{
while(priority(stack[top]) >= priority(*ep))
printf("%c",pop());
push(*ep);
}
ep++;
}
while(top != -1)
{
printf("%c",pop());
}
}
```

Output:

```
Enter your expression: a+b*c-
d/e

Postfix of the expression is
abc*+de/-
```

Question 3: Write a C Program to implement Queue using two Stacks.

Answer:

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

struct sNode
{
int data;
struct sNode* next;
};

void push(struct sNode** topr, int new_data)
{
struct sNode* new_node = (struct sNode*)malloc(sizeof(struct sNode));
if (new_node == NULL)
{
printf("OVERFLOW...! \n");
getchar();
exit(0);
}
new_node->data = new_data;
new_node->next = (*topr);
(*topr) = new_node;
}

int pop(struct sNode** topr)
```

```
{
int r;
struct sNode* top;
if (*topr == NULL)
{
printf("UNDERFLOW...!! \n");
getchar();
exit(0);
}
else
{
top = *topr;
r = top->data;
*topr = top->next;
free(top);
return r;
}
}

struct queue
{
struct sNode* stack1;
struct sNode* stack2;
};

void enQueue(struct queue* q, int x)
{
```

```
push(&q->stack1, x);
}

int deQueue(struct queue* q)
{
int x;
if (q->stack1 == NULL && q->stack2 == NULL)
{
printf("Queue is empty");
getchar();
exit(0);
}

if (q->stack2 == NULL)
{
while (q->stack1 != NULL)
{
x = pop(&q->stack1);
push(&q->stack2, x);
}
}
x = pop(&q->stack2);
return x;
}

void main()
{
```

```
struct queue* s = (struct queue*)malloc(sizeof(struct queue));
s->stack1 = NULL;
s->stack2 = NULL;
enQueue(s, 10);
enQueue(s, 20);
enQueue(s, 30);
enQueue(s, 40);
printf("%d ", deQueue(s));
printf("%d ", deQueue(s));
}
```

Output:

```
10 20
```

Question 4: Write a C program for insertion and deletion of BST.

Answer:

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

struct node
{
int key;
struct node *left, *right;
};

struct node *newNode(int item)
```

```
{
struct node *temp =  (struct node *)malloc(sizeof(struct node));
temp->key = item;
temp->left = temp->right = NULL;
return temp;
}

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

struct node* insert(struct node* node, int key)
{
if (node == NULL) return newNode(key);
if (key < node->key)
node->left  = insert(node->left, key);
else if (key > node->key)
node->right = insert(node->right, key);
return node;
}
```

```
struct node * minNode(struct node* node)
{
struct node* g = node;
while (g && g->left != NULL)
g = g->left;
return g;
}

struct node* deleteNode(struct node* root, int key)
{
if (root == NULL) return root;
if (key < root->key)
root->left = deleteNode(root->left, key);
else if (key > root->key)
root->right = deleteNode(root->right, key);
else
{
if (root->left == NULL)
{
struct node *temp = root->right;
free(root);
return temp;
}
else if (root->right == NULL)
{
struct node *temp = root->left;
free(root);
```

```
return temp;
}

struct node* temp = minNode(root->right);
root->key = temp->key;
root->right = deleteNode(root->right, temp->key);
}
return root;
}

void main()
{
struct node *root = NULL;
root = insert(root, 1000);
insert(root, 500);
insert(root, 200);
insert(root, 800);
insert(root, 2000);
insert(root, 1500);
insert(root, 4000);
inorder(root);
printf("\nDeleted the number 2000\n");
root = deleteNode(root, 2000);
printf("Inorder traversal of the modified tree is \n");
inorder(root);
}
```

Output:

```
200

500

800

1000

1500

2000

4000

Deleted the number 2000

Inorder traversal of the modified tree is

200

500

800

1000

1500

4000
```