Assignment 5

#C program to reverse a String using Stack

```
#include<stdio.h>
#include<string.h>
#define MAX 20
int top = -1;
char stack[MAX];
char push(char item)
{
   if(top == (MAX-1))
      printf("Stack Overflow\n");
   else
      stack[++top] =item;
}
char pop()
{
    if(top == -1)
        printf("Stack Underflow\n");
    else
        return stack[top--];
}
main()
{
        char str[20];
        int i;
        printf("Enter the String : " );
        gets(str);
        for(i=0;i<strlen(str);i++)
                push(str[i]);
        for(i=0;i<strlen(str);i++)
                str[i]=pop();
        printf("Reversed String is : ");
        puts(str);
}
```

Output :

Enter the String : noushika
Reversed String is : akihsuon

#C program for Infix to Postfix Conversion using Stack

```
#include <stdio.h>
#include <ctype.h>
#define SIZE 50
char stack[SIZE];
int top=-1;
push(char elem)
{
   stack[++top]=elem;
}

char pop()
{
   return(stack[top--]);
}
int pr(char symbol)
{

        if(symbol == '^')
        {
                return(3);
        }
        else if(symbol == '*' || symbol == '/')
        {
                return(2);
        }
        else if(symbol == '+' || symbol == '-')
        {
                return(1);
        }
        else
        {
                return(0);
        }
}

void main()
{
   char infix[50],postfix[50],ch,elem;
```

```
    int i=0,k=0;
    printf("Enter Infix Expression : ");
    scanf("%s",infix);
    push('#');
    while( (ch=infix[i++]) != '\0')
    {
        if( ch == '(') push(ch);
        else
            if(isalnum(ch)) postfix[k++]=ch;
            else
                if( ch == ')')
                {
                    while( stack[top] != '(')
                        postfix[k++]=pop();
                    elem=pop();
                }
                else
                {
                    while( pr(stack[top]) >= pr(ch) )
                        postfix[k++]=pop();
                    push(ch);
                }
    }
    while( stack[top] != '#')
        postfix[k++]=pop();
    postfix[k]='\0';
    printf("\nPostfix Expression =  %s\n",postfix);
}
```

Output :

```
Enter Infix Expression : A / B * C * D + E
Enter Post Expression : AB / C * D * E +
```

#C program to Implement Queue using two Stack

```
#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;
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");
      }}}

void create()
{
   top1 = top2 = -1;
}

void push1(int element)
{
```

```
    stack1[++top1] = element;
}

int pop1()
{
    return(stack1[top1--]);
}

void push2(int element)
{
    stack2[++top2] = element;
}

int pop2()
{
    return(stack2[top2--]);
}

void enqueue()
{
    int data, i;
    printf("Enter the data : ");
    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;
   if(top1 == -1)
   {
      printf("\nEMPTY QUEUE\n");
   }
   else
   {
      printf("\nQUEUE ELEMENTS : ");
      for (i = 0;i <= top1;i++)
      {
         printf(" %d ", stack1[i]);
      }
      printf("\n");
   }
}
```

Output :

```
Enter your choice : 1
Enter the data : 15
Enter your choice : 1
Enter the data : 24
Enter your choice : 1
Enter the data : 3
Enter your choice : 3
QUEUE ELEMENTS : 15 24 3
Enter your choice : 2
Enter your choice : 3
QUEUE ELEMENTS : 15 0 0
Enter your choice : 4
```

#C program for insertion and deletion of BST

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

struct node{
   int data;
   struct node *left;
   struct node *right;
};

struct node *root= NULL;
```

```
struct node* createNode(int data){

    struct node newNode = (struct node)malloc(sizeof(struct node));

    newNode->data= data;
    newNode->left = NULL;
    newNode->right = NULL;

    return newNode;
}

void insert(int data) {

    struct node *newNode = createNode(data);

    if(root == NULL){
        root = newNode;
        return;
      }
    else {

        struct node *current = root, *parent = NULL;

        while(true) {

            parent = current;

            if(data < current->data) {
                current = current->left;
                if(current == NULL) {
                    parent->left = newNode;
                    return;
                }
            }

            else {
                current = current->right;
                if(current == NULL) {
                    parent->right = newNode;
                    return;
                }
            }
```

```
        }
    }
}

struct node* minNode(struct node *root) {
    if (root->left != NULL)
        return minNode(root->left);
    else
        return root;
}

struct node* deleteNode(struct node *node, int value)
{
    if(node == NULL){
         return NULL;
     }
    else {

        if(value < node->data)
        {
            node->left = deleteNode(node->left, value);
        }
        else if(value > node->data)
        {
            node->right = deleteNode(node->right, value);
        }
        else if(node->left == NULL && node->right == NULL)
        {
              node = NULL;
         }

        else if(node->left == NULL)
        {
              node = node->right;
           }

        else if(node->right == NULL)
        {
              node = node->left;
           }

        else {
```

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


void inorderTraversal(struct node *node) {

   if(root == NULL){
      printf("Tree is empty\n");
        return;
    }
   else {

      if(node->left!= NULL)
         inorderTraversal(node->left);
      printf("%d ", node->data);
      if(node->right!= NULL)
        inorderTraversal(node->right);

   }
}

int main()
{

   insert(20);
   insert(10);
   insert(30);
   insert(80);
   insert(55);
   insert(60);

   printf("Binary search tree after insertion: \n");
   inorderTraversal(root);

   struct node *deletedNode = NULL;
   deletedNode = deleteNode(root, 30);
   printf("\nBinary search tree after deleting node 30: \n");
   inorderTraversal(root);
```

```
    deletedNode = deleteNode(root, 20);
    printf("\nBinary search tree after deleting node 20: \n");
    inorderTraversal(root);

    deletedNode = deleteNode(root, 80);
    printf("\nBinary search tree after deleting node 80: \n");
    inorderTraversal(root);

    return 0;
}
```

Output :

```
Binary search tree after insertion : 20 10 30 80 55 60
Binary search tree after deleting node 30 : 20 10 80 55 60
Binary search tree after deleting node 20 : 10 80 55 60
Binary search tree after deleting node 80 : 10 55 60
```