DSA ASSIGNMENT

NAME :- E HEMASUNDAR YADAV

ROL :- AP19110010481

SEC :-CSE-H

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

```
#include <stdio.h>
#include <string.h>
#define max 100
int top=-1;
char str[max];

void push(char x){
   if(top == max-1){
      printf("str overflow");
   } else {
      str[++top] =x;
   }
}
 void pop(){
   printf("%c", str[top--]);
```

```
}
 int main(){
   int i;
   printf("Enter the string:");
   scanf("%s",str);
   for(i=0; i<strlen(str); i++)
   push(str[i]);
   for(i=0; i<strlen(str); i++)
   pop();
   printf("The reversed string is %s",str);
}
```

Output:-

Enter the string: airplane

The reversed string is :- enalpira

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

```
// C program to convert infix expression to postfix
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
```

```
// Stack type
struct Stack
{
    int top;
    unsigned capacity;
    int* array;
};

// Stack Operations
struct Stack* createStack( unsigned capacity )
{
    struct Stack* stack = (struct Stack*) malloc(sizeof(struct Stack));

    if (!stack)
        return NULL;

    stack->top = -1;
    stack->capacity = capacity;

    stack->array = (int*) malloc(stack->capacity * sizeof(int));

    return stack;
```

```
}
int isEmpty(struct Stack* stack)
{
   return stack->top == -1 ;
}
char peek(struct Stack* stack)
{
   return stack->array[stack->top];
}
char pop(struct Stack* stack)
{
   if (!isEmpty(stack))
      return stack->array[stack->top--] ;
   return '$';
}
void push(struct Stack* stack, char op)
{
   stack->array[++stack->top] = op;
}


// A utility function to check if the given character is operand
int isOperand(char ch)
```

```
{
    return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z');
}

// A utility function to return precedence of a given operator
// Higher returned value means higher precedence
int Prec(char ch)
{
    switch (ch)
    {
    case '+':
    case '-':
        return 1;

    case '*':
    case '/':
        return 2;

    case '^':
        return 3;
    }
    return -1;
}
```

```
// The main function that converts given infix expression
// to postfix expression.
int infixToPostfix(char* exp)
{
    int i, k;

    // Create a stack of capacity equal to expression size
    struct Stack* stack = createStack(strlen(exp));
    if(!stack) // See if stack was created successfully
        return -1 ;

    for (i = 0, k = -1; exp[i]; ++i)
    {
        // If the scanned character is an operand, add it to output.
        if (isOperand(exp[i]))
            exp[++k] = exp[i];

        // If the scanned character is an '(', push it to the stack.
        else if (exp[i] == '(')
            push(stack, exp[i]);
```

```
    // If the scanned character is an ‘)’, pop and output from the stack
    // until an ‘(‘ is encountered.
    else if (exp[i] == ')')
    {
        while (!isEmpty(stack) && peek(stack) != '(')
            exp[++k] = pop(stack);
        if (!isEmpty(stack) && peek(stack) != '(')
            return -1; // invalid expression
        else
            pop(stack);
    }
    else // an operator is encountered
    {
        while (!isEmpty(stack) && Prec(exp[i]) <= Prec(peek(stack)))
            exp[++k] = pop(stack);
        push(stack, exp[i]);
    }

}

// pop all the operators from the stack
while (!isEmpty(stack))
    exp[++k] = pop(stack );
```

```
    exp[++k] = '\0';
    printf( "%s", exp );
}

// Driver program to test above functions
int main()
{
    char exp[] = " a+b-c/e*f ";
    infixToPostfix(exp);
    return 0;
}
```

Output :-

```
Enter the expression you want to convert: a+b-c/e*f
ab+ce/f*-
```

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 sNode*)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 sNode** 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 anove 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, 25);
    enQueue(q, 35);
    enQueue(q, 45);

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

    return 0;
}
```

Output :-

25 35 45

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

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

```
#include<stdlib.h>
struct node{
   int data;
   struct node *leftlink;
   struct node *rightlink;
}*root=NULL;
struct node* insert(struct node* root,int e)
{
   if(root==NULL)
   {
      root=(struct node*)malloc(sizeof(struct node));
      root->data=e;
      root->leftlink=root->rightlink=NULL;
      return root;
  }
   else if(root->data>e)
   {
      root->leftlink=insert(root->leftlink,e);
   }
   else if(root->data<e)
   {
      root->rightlink=insert(root->rightlink,e);
   }
```

```
    return root;
}
int minimum(struct node* root)
{
    if(root->leftlink==NULL)
    {
        return root->data;
    }
    else
    {
        return minimum(root->leftlink);
    }

}
struct node* delete(struct node* root,int e)
{
    if(root==NULL)
    {
        return root;
    }
    else if(root->data>e)
    {
        root->leftlink=delete(root->leftlink,e);
```

```
}
else if(root->data<e)
{
   root->rightlink=delete(root->rightlink,e);

}
else
{
   if(root->leftlink==NULL && root->rightlink==NULL)
   {
   remove;
   root;
   return NULL;
   }
   else if(root->leftlink==NULL)
   {
      root=root->rightlink;
   }
   else if(root->rightlink==NULL)
   {
      root=root->leftlink;
```

```
    }
    else
    {
        int key=minimum(root->rightlink);
        root->data=key;
        root->rightlink=delete(root->rightlink,key);
    }

}
return root;
}
void inorder(struct node *root)
{
    if(root==NULL)
    {
        return;

    }
    inorder(root->leftlink);
    printf("%d",root->data);
    inorder(root->rightlink);
}
int main()
```

```
{
    int n,i,e;
    printf("Enter the number of elements:");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("Enter the element:");
        scanf("%d",&e);
        root=insert(root,e);

    }
    inorder(root);
    printf("\n");
    printf("Enter the element that has to remove:");
    scanf("%d",&e);
    root=delete(root,e);
    inorder(root);

}
```

Output :-

```
Enter the number of elements:8
Enter the element:9
```

Enter the element:8

Enter the element:7

Enter the element:6

Enter the element:5

Enter the element:4

Enter the element:3

Enter the element:2

98765432

Enter the element that has to remove:5

9876432