Assignment 1(Stack)

1) Implementation of Stack & it's Functions (Push, Pop, Peek) + Check conditions of overflow & underflow (IsEmpty / IsFull) :

CODE :

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

// Global Variables
int stack[100];
int top = -1;

int isEmpty()
{
    return (top == -1);
}

int isFull()
{
    return (top == MAX_SIZE - 1);
}

void push(int data)
{
    if (isFull())
    {
        printf("Stack Overflow/Full!\n");
        return;
    }
    stack[++top] = data;
    printf("%d pushed succesfully.\n", data);
}

int pop()
{
    if (isEmpty())
    {
        printf("Stack Underflow/Empty!\n");
        return -1;
    }
    int data = stack[top--];

    return data;
}

int peek()
{
```

```
    return stack[top];
}

void printStack()
{
    if (isEmpty())
    {
        printf("The Stack is empty, nothing to print!\n");
        return;
    }
    while (!isEmpty())
    {
        printf("%d\n", pop());
    }
}

int main()
{
    int data;
    int ch;

    printf("1 for push.\n");
    printf("2 for pop.\n");
    printf("3 for peek.\n");
    printf("4 for print stack.\n");
    printf("0 to exit.\n");

loop:
    printf("\nEnter operation to perform : ");
    scanf("%d", &ch);

    switch (ch)
    {
    case 1:
        printf("Enter data to push : ");
        scanf("%d", &data);
        push(data);
        goto loop;

    case 2:
        printf("%d poped succesfully.\n", pop());
        goto loop;

    case 3:
        printf("%d peeked.\n", peek());
        goto loop;

    case 4:
```

```
        printf("Your stack is :\n");
        printStack();
        goto loop;

    case 0:
        printf("Exited Successfully!!\n");
        break;

    default:
        printf("Error, Try again!!\n");
        goto loop;
    }

    return 0;
}
```

OUTPUT :

```
C:\Users\DARSH PATEL\Desktop\DS Assignment\Stack>Stack
1 for push.
2 for pop.
3 for peek.
4 for print stack.
0 to exit.

Enter operation to perform : 1
Enter data to push : 33
33 pushed succesfully.

Enter operation to perform : 2
33 poped succesfully.

Enter operation to perform : 3
0 peeked.

Enter operation to perform : 4
Your stack is :
The Stack is empty, nothing to print!

Enter operation to perform : 0
Exited Successfully!!

C:\Users\DARSH PATEL\Desktop\DS Assignment\Stack>
```

2) Implementation of Postfix notation.

CODE :

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

#define MAX_STACK_SIZE 100

// Global Variables
char stack[100];
int top = -1;
```

```c
int isEmpty()
{
    return (top == -1);
}

int isFull()
{
    return (top == MAX_STACK_SIZE - 1);
}

void push(char data)
{
    if (isFull())
    {
        printf("Stack Overflow/Full!\n");
        return;
    }
    stack[++top] = data;
}

char pop()
{
    if (isEmpty())
    {
        printf("Stack Underflow/Empty!\n");
        return -1;
    }
    char data = stack[top--];

    return data;
}

char peek()
{
    return stack[top];
}

void printStack()
{
    if (isEmpty())
    {
        printf("The Stack is empty, nothing to print!\n");
        return;
    }
    while (!isEmpty())
    {
        printf("%c\n", pop());
```

```
    }
}

int precedence(char operator)
{
    switch (operator)
    {
    case '+': case '-': return 1;
    case '*': case '/': return 2;
    case '^': return 3;
    default: return 0;
    }
}

int isOperator(char ch)
{
    if (ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '^')
    {
        return 1;
    }
    return 0;
}

char *infixToPostfix(char *infix)
{
    int i,j=0;
    int len = strlen(infix);
    char *postfix = malloc(len+2);
    for ( i = 0; i < len; i++)
    {
        char ch = infix[i];

        if(ch == '(')
        {
            push(ch);
        }
        else if(ch==')')
        {
            while (peek()!='(')
            {
                postfix[j++]=pop();
            }
            pop();
        }
        else if(isOperator(ch))
        {
            while (!isEmpty() && precedence(peek())>=precedence(ch))
            {
```

```
                postfix[j++]=pop();
            }
            push(ch);
        }
        else
        {
            postfix[j++] = ch;
        }
    }
    while (!isEmpty())
    {
        if(postfix[j]=='(' || postfix[j]==')')
        {
            printf("The expression has a parenthesis error!!");
            return NULL;
        }
        postfix[j++] = pop();
    }
    postfix[j] = '\0';

    return postfix;
}

int main(int argc, char const *argv[])
{
    char *infix = malloc(MAX_STACK_SIZE-1);
    printf("Enter any expression in infix form : ");
    scanf("%s",infix);

    printf("Your Postfix expression is : %s",infixToPostfix(infix));

    return 0;
}
```

OUTPUT :

```
C:\Users\DARSH PATEL\Desktop\DS Assignment\Stack>ITPO
Enter any expression in infix form : (a+b)*c-d/q
Your Postfix expression is : ab+c*dq/-
C:\Users\DARSH PATEL\Desktop\DS Assignment\Stack>
```

3) Implementation of Tower of Hanoi.

CODE :

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

#define MAXSIZE 100
```

```c
// Stack Structure
struct Stack
{
    int *arr;
    int top;
    char name;
};

// Creates a Stack
struct Stack initializer(char name, int size)
{
    struct Stack stack;
    stack.top = -1;
    stack.name = name;
    stack.arr = (int *)malloc(size * sizeof(int));

    return stack;
}

// Checks stack is full or not
int isFull(struct Stack *stack)
{
    return (stack->top == MAXSIZE - 1);
}

// Checks if Stack is Empty
int isEmpty(struct Stack *stack)
{
    return (stack->top == -1);
}

// Pushes data on top of Stack
void push(struct Stack *stack, int data)
{
    if (isFull(stack))
    {
        printf("Stack Overflow/Full!!\n");
        return;
    }
    stack->top++;
    stack->arr[stack->top] = data;
}

// Removes and Returns top data of Stack
int pop(struct Stack *stack)
{
    if (isEmpty(stack))
```

```
    {
        printf("Stack Underflow/Empty!!\n");
        return -1;
    }

    int data = stack->arr[stack->top];
    stack->top--;

    return data;
}

// Returns top data of Stack
int peek(struct Stack *stack)
{
    return stack->arr[stack->top];
}

// Prints Stack
void printStack(struct Stack *stack)
{
    while (!isEmpty(stack))
    {
        int data = pop(stack);
        printf("\n%d", data);
    }
    printf("\n");
}

// Add disks to Source Tower
void addDisksInSrc(int disks, struct Stack *src)
{
    while (disks)
    {
        push(src,disks--);
    }
}

// Tower of Hanoi Solver
void TowerOfHanoi(int disks, struct Stack *src, struct Stack *aux, struct
Stack *dest)
{
    if (disks == 1)
    {
        printf("Move no. %d disk from %c to %c.\n", peek(src), src->name,
dest->name);
        push(dest, pop(src));
        return;
    }
```

```
    TowerOfHanoi(disks - 1, src, dest, aux);
    printf("Move no. %d disk from %c to %c.\n", peek(src), src->name, dest-
>name);
    push(dest, pop(src));
    TowerOfHanoi(disks - 1, aux, src, dest);
}

// Main Function
int main(int argc, char const *argv[])
{
    int disks;

    printf("Welcome to Tower of Hanoi solver!\nHere,\nS means Source Tower,\nA
means Auxilary Tower, \nD means Destination Tower.");
    printf("\nEnter no. of disks in tower of hanoi : ");
    scanf("%d", &disks);

    struct Stack src = initializer('S', MAXSIZE);
    struct Stack aux = initializer('A', MAXSIZE);
    struct Stack dest = initializer('D', MAXSIZE);

    addDisksInSrc(disks, &src);
    TowerOfHanoi(disks, &src, &aux, &dest);

    printf("Your Destination Tower after solving is :");
    printStack(&dest);

    return 0;
}
```

OUTPUT :

```
C:\Users\DARSH PATEL\Desktop\DS Assignment\Stack>TOHS
Welcome to Tower of Hanoi solver!
Here,
S means Source Tower,
A means Auxilary Tower,
D means Destination Tower.
Enter no. of disks in tower of hanoi : 3
Move no. 3 disk from S to D.
Move no. 2 disk from S to A.
Move no. 3 disk from D to A.
Move no. 1 disk from S to D.
Move no. 3 disk from A to S.
Move no. 2 disk from A to D.
Move no. 3 disk from S to D.
Your Destination Tower after solving is :
3
2
1

C:\Users\DARSH PATEL\Desktop\DS Assignment\Stack>
```