THADOMAL SHAHANI
TSEC
ENGINEERING COLLEGE

Assignment 4

Aim: To write a C program to convert infix into postfix using ADT and reverse a string using stack. Understand infix, prefix and postfix notations with suitable examples.

Theory

Infix - Commonly used mathematical form
e.g. A+B*C read left to right

Postfix - Operator is written after the operands
e.g. ABC*+ read left to right

Prefix - Operator is written before the operands
e.g. +A*BC- read right to left

A stack ADT with push pop and peek functions have been used for the programs

~~Postfix evaluation~~ Infix to postfix
Create an empty stack
Scan the expression from left to right
If the element is an operand it gets added to the postfix. If it is an open bracket, it gets pushed to the stack. If close bracket is encountered it gets popped until the open bracket is found.
In case an operator is encountered, pop

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

operators from stack with higher or equal precedence, then push the current operator.

Once the the entire string is scanned, ~~push~~ add the remaining operators to the post fix.

The time complexity is O(n) as each character is processed once.

~~For~~ String reversal

The stack follows last in first out so the string is read from left to right and ~~pushed to the stack~~. the individual letters are pushed to the stack.
The last letter pushed will be the first one to exit the stack so it will be the first letter in the answer string so the string will get reversed ~~and print~~. This is done until the stack is empty and a '\0' is added after.

Time complexity is O(n) as each character is pushed and popped once

Conclusion
Implementation of stack in real world use cases to ~~make a problem~~ increase the efficiency and solve ~~a~~ problems using it.

A1.

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

#define SIZE 100

struct Stack {
   int top;
   char arr[SIZE];
};

void push(struct Stack *s, char ch);
char pop(struct Stack *s);
int precedence(char ch);

void main() {
   struct Stack s;
   s.top = -1;

   char infix[SIZE], postfix[SIZE];
   int i, j = 0;

   printf("Enter Infix: ");
   scanf("%s", infix);

   for (i = 0; infix[i] != '\0'; i++) {
      if (isalnum(infix[i])) {
         postfix[j++] = infix[i];
```

```
        }
        else if (infix[i] == '(') {
            push(&s, infix[i]);
        }
        else if (infix[i] == ')') {
            while (s.top != -1 && s.arr[s.top] != '(') {
                postfix[j++] = pop(&s);
            }
            pop(&s); // remove '('
        }
        else {
            while (s.top != -1 && precedence(s.arr[s.top]) >= precedence(infix[i])) {
                postfix[j++] = pop(&s);
            }
            push(&s, infix[i]);
        }
    }

    while (s.top != -1) {
        postfix[j++] = pop(&s);
    }

    postfix[j] = '\0';
    printf("Postfix : %s\n", postfix);
}

void push(struct Stack *s, char ch) {
    if (s->top < SIZE - 1) {
```

```
    s->arr[++s->top] = ch;
  }
}

char pop(struct Stack *s) {
  if (s->top != -1) {
    return s->arr[s->top--];
  }
  return -1;
}

int precedence(char ch) {
  if (ch == '+' || ch == '-')
    return 1;
  if (ch == '*' || ch == '/')
    return 2;
  return 0;
}
```

```
Enter Infix: 1+2*3
Postfix : 123*+
```

A2.

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

# define SIZE 100
struct Stack {
  int top;
  char arr[SIZE];
```

```
};

void push(struct Stack *s, char a);
char pop(struct Stack *s);
int check(struct Stack *s);
void main()
{
   struct Stack s;
   s.top = 0;
   char st[SIZE];
   printf("Enter the String: ");
   gets(st);
   for (int i = 0; st[i] != '\0'; i++){
      push(&s, st[i]);
   }
   for (int i = 0; s.top > 0; i++)
   {
      st[i] = pop(&s);
   }
   printf("Reversed String is: %s", st);
}

/***
Adds elements to the top of the stack
***/
void push(struct Stack *s, char a)
{
   // Checks if the stack is full
```

```
  if (s->top < SIZE)
  {
    s->arr[s->top] = a;
    s->top++;
  }
  else
  {
    printf("STACK IS FULL\n");
  }
}
/***
Remove the last element from the stack
***/
char pop(struct Stack *s)
{
  //Checks if the stack is not empty
  if (s->top > 0)
  {
    s->top--;
    return s->arr[s->top];
  }
  else
  {
    printf("NO ELEMENTS IN STACK\n");
  }
}
```

```
Enter the String: HELLO
Reversed String is: OLLEH
```