Evaluation of INFIX expression

//code

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
#define MAXSTACK 100
#define SIZE 100

char stack[MAXSTACK];
int top = -1;
int topNum = -1;

void pushNum(int item) {
    if (topNum >= MAXSTACK-1) {
        printf("OVERFLOW");
        return;
    } else {
        topNum++;
        stack[topNum] = item;
    }
}

int popNum() {
    int num;
    if (topNum < 0) {
```

```
        printf("UNDERFLOW");
    } else {
        num = stack[topNum];
        topNum--;
        return num;
    }
}

void push(char item) {
    if (top >= MAXSTACK-1) {
        printf("OVERFLOW");
        return;
    } else {
        top++;
        stack[top] = item;
    }
}

char pop() {
    char item;
    if (top < 0) {
        printf("UNDERFLOW");
    } else {
        item = stack[top];
        top--;
        return item;
```

```
    }
}

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

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

void infixToPostfix(char infix[], char postfix[]) {
    int i=0, j=0;
    char item, x;
    strcat(infix, ")");
    push('(');
```

```
for(i=0 ; infix[i] != '\0' ; i++) {
    item = infix[i];
    if(item == '(') {
        push('(');
    } else if(isdigit(item)) {
        postfix[j++] = item;
    } else if(isOperator(item)) {
        x = pop();
        while(isOperator(x)==1 && precedence(x)>=precedence(item)) {
            postfix[j++] = x;
            x=pop();
        }
        push(x);
        push(item);
    } else if(item == ')') {
        x = pop();
        while (x != '(') {
            postfix[j++] = x;
            x = pop();
        }
    }
}
if(top > 0) {
    printf("Invalid expression");
}
postfix[j]='\0';
```

```
}

void evalPostfix(char postfix[]) {
    int i, a, b, val;
    char ch;

    for(i=0 ; postfix[i]!=')' ; i++) {
        ch = postfix[i];
        if(isdigit(ch)) {
            pushNum(ch-'0');
        } else if(ch=='+' || ch=='-' || ch=='*' || ch=='/') {
            a = popNum();
            b = popNum();
            switch(ch) {
                case '+':
                    val = b+a;
                    break;
                case '-':
                    val = b-a;
                    break;
                case '*':
                    val = b*a;
                    break;
                case '/':
                    val = b/a;
                    break;
```

```
            }
            pushNum(val);
        }
    }

    if (topNum > 0) {
        printf("Invalid Input");
    } else {
        printf("\nResult of given Infix expression is :  %d", popNum());
    }
}

int main() {
    char infix[SIZE], postfix[SIZE];
    printf("Enter infix Expression :  ");
    gets(infix);
    infixToPostfix(infix, postfix);
    puts(postfix);
    strcat(postfix, ")");
    evalPostfix(postfix);
}
//output
```

```
Enter infix Expression :  (2*(4-2)+7)

Result of given Infix expression is :  11


Process returned 0 (0x0)   execution time : 10.093 s
Press any key to continue.
```

```
Enter infix Expression :  7-4*2+3*(4+2)

Result of given Infix expression is :  17


Process returned 0 (0x0)   execution time : 18.402 s
Press any key to continue.
```

****

Evaluation of PREFIX expression:

//code

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define MAXSTACK 100
#define PREFIXSIZE 100

int stack[MAXSTACK];
int top = -1;

void push(int item) {
   if (top >= MAXSTACK-1) {
```

```
        printf("OVERFLOW");
        return;
    } else {
        top++;
        stack[top] = item;
    }
}

int pop() {
    int num;
    if (top < 0) {
        printf("UNDERFLOW");
    } else {
        num = stack[top];
        top--;
        return num;
    }
}

void evalPrefix(char prefix[]) {
    int i, a, b, val;
    char ch;

    for(i=0 ; prefix[i]!=')' ; i++) {
        ch = prefix[i];
        if(isdigit(ch)) {
            push(ch-'0');
        } else if(ch=='+' || ch=='-' || ch=='*' || ch=='/') {
            a = pop();
```

```
            b = pop();
            switch(ch) {
               case '+':
                  val = a+b;
                  break;
               case '-':
                  val = a-b;
                  break;
               case '*':
                  val = a*b;
                  break;
               case '/':
                  val = a/b;
                  break;
            }
            push(val);
         }
      }
      if (top > 0) {
         printf("Invalid input");
      } else {
         printf("\nResult of given prefix expression is :  %d\n\n", pop());
      }
   }

   int main() {
      int i;
      char prefix[PREFIXSIZE];
      printf("Enter prefix expression :  ");
```

```
    gets(prefix);
    strrev(prefix);
    strcat(prefix, ")");
    evalPrefix(prefix);
    return 0;
}
```

//output

```
Enter prefix expression :  *+23-54

Result of given prefix expression is :  5


Process returned 0 (0x0)   execution time : 6.520 s
Press any key to continue.
```

```
Enter prefix expression :  +-27*8/48

Result of given prefix expression is :  -5


Process returned 0 (0x0)   execution time : 11.804 s
Press any key to continue.
```

****