LAB – 4

AIM: **Applications of Stack**

Problems:

Q.1 Write a program to convert expression represented in infix notation to postfix (reverse polish) notation.

Ans:

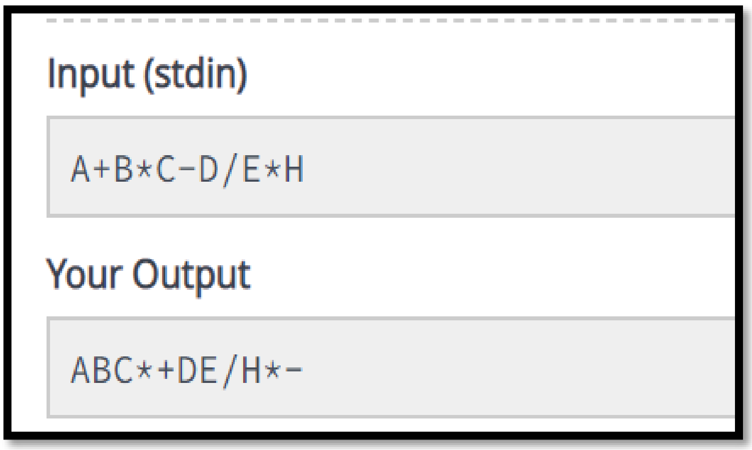
CODE:

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

int top = 1;
char s[100000] ;

void PUSH(char c) {
   //printf("PUSH is called\n");
   top++;
   s[top] = c;
   /*printf("Content of stack is : ");
   for(i=0;i<=top;i++)
      printf("%c",s[i]);
   printf("\n");*/
}

char POP() {
   //printf("pop is called\n");
   /*if(top <= 0 ) {
      printf("Stack is empty\n");
      exit(0);
   }*/
   top--;
   return (s[top+1]);
}

int input_precedence(char c) {
   //printf("IP\n");
   switch(c) {
      case '+':
      case '-':
            return 1;
        case '*':
        case '/':
            return 3;
        case '^' :
            return 6;
        case '(':
            return 9;
        case ')':
            return 0;
        case '%':
            printf("INPUT EXPRESSION IS INVALID\n");
            exit(0);
        /*default :
            if((c >= 65 && c <= 90) || (c >= 95 && c <= 122))
                return 7;*/
    }
    return 7;
}

int stack_precedence(char c) {
    //printf("SP\n");
    switch(c) {
        case '+':
        case '-':
            return 2;
        case '*':
        case '/':
            return 4;
        case '^' :
            return 5;
        case '(':
            return 0;
        /*default :
                if((c >= 65 && c <= 90) || (c >= 95 && c <= 122))
                    return 8;*/
    }
    return 8;
}

int RANK(char c) {
    //printf("RANK\n");
    switch(c) {
        case '+':
        case '-':
        case '*':
        case '/':
        case '^':
            return -1;
        //default :
          //  return 1;
    }
    return 1;
}

int main() {
    int i,rank = 0,length,j=0;
    s[top] = '(';
    char polish[100000],infix[100000],temp;
    gets(infix);
    length = strlen(infix);
    //printf("%d\n",length);
    infix[length] = ')';
    //printf("%s\n",infix);
    for(i=0;i<=length;i++) {
       if(top < 1) {
          printf("INPUT EXPRESSION IS INVALID\n");
          exit(0);
       }
       while(input_precedence(infix[i]) < stack_precedence(s[top]) ) {
          temp = POP();
          polish[j] = temp;
          //printf("%c",temp);
          j++;
          rank += RANK(temp);
          if(rank < 1) {
             printf("INPUT EXPRESSION IS INVALID\n");
             exit(0);
          }
       }
       if(input_precedence(infix[i]) != stack_precedence(s[top]))
          PUSH(infix[i]);
       else
          POP();
    }

    if(top != 0 || rank != 1) {
       printf("INPUT EXPRESSION IS INVALID\n");
       exit(0);
    }
    else
       printf("%s\n",polish);

    return 0;
 }
```

TESTCASE:

TESTCASE	INPUT	OUTPUT
1	A+B*C-D/E*H	ABC*+DE/H*-

TESTCASE – 1: OUTPUT



Input (stdin)

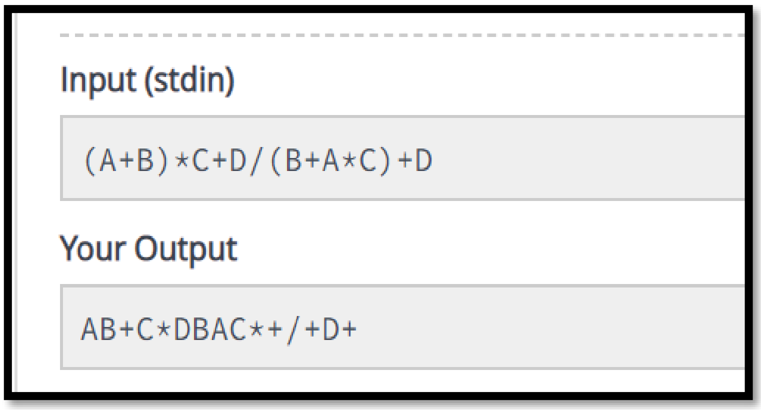
```
A+B*C-D/E*H
```

Your Output

```
ABC*+DE/H*-
```

TESTCASE	INPUT	OUTPUT
2	(A+B)*C+D/(B+A*C)+D	AB+C*DBAC*+/+D+

<u>TESTCASE – 2: OUTPUT</u>



Input (stdin)

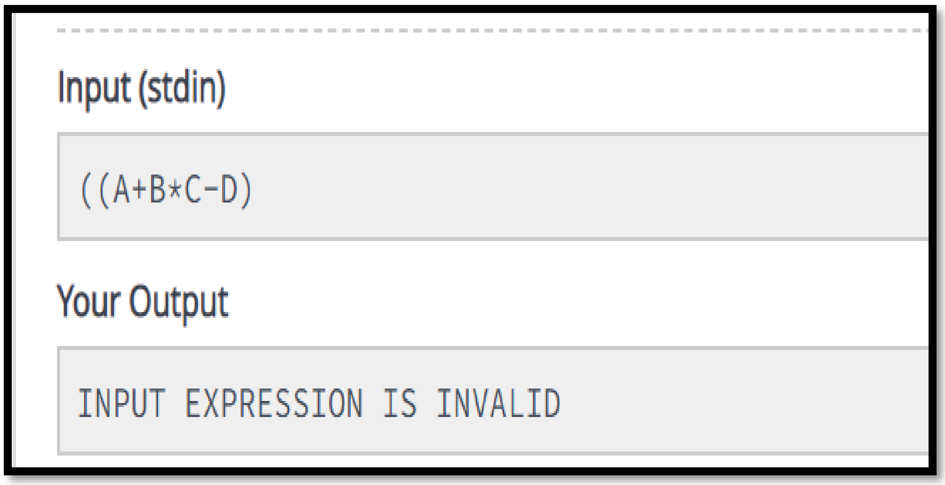
```
(A+B)*C+D/(B+A*C)+D
```

Your Output

```
AB+C*DBAC*+/+D+
```

TESTCASE	INPUT	OUTPUT
3	((A+B*C-D)	INPUT EXPRESSION IS INVALID

<u>TESTCASE – 3: OUTPUT</u>



Input (stdin)

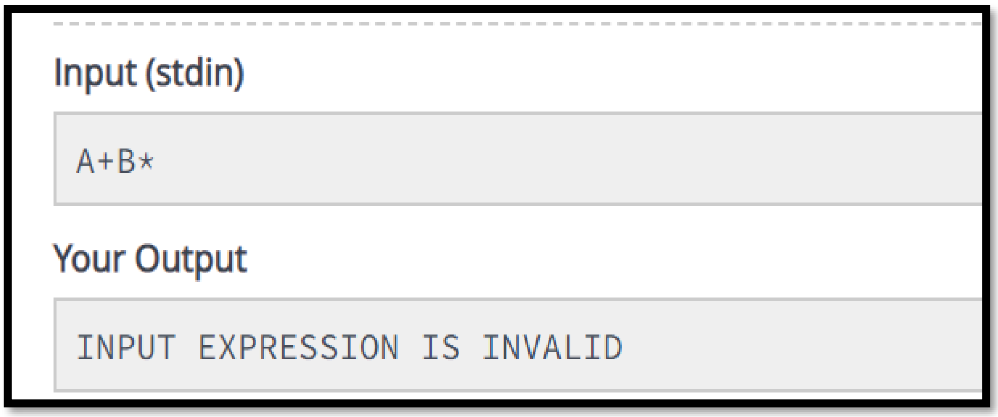
```
((A+B*C-D)
```

Your Output

```
INPUT EXPRESSION IS INVALID
```

TESTCASE	INPUT	OUTPUT
4	A+B*	INPUT EXPRESSION IS INVALID

TESTCASE – 4: OUTPUT



Input (stdin)

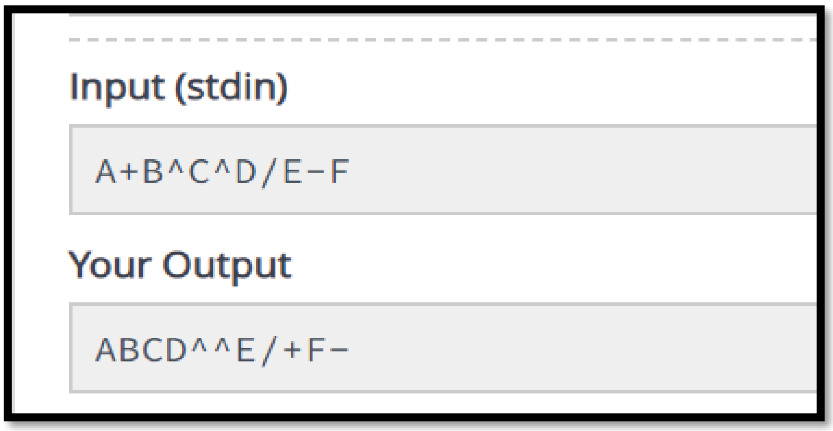
```
A+B*
```

Your Output

```
INPUT EXPRESSION IS INVALID
```

TESTCASE	INPUT	OUTPUT
5	A+B^C^D/E-F	ABCD^^E/+F

TESTCASE – 5: OUTPUT



Input (stdin)

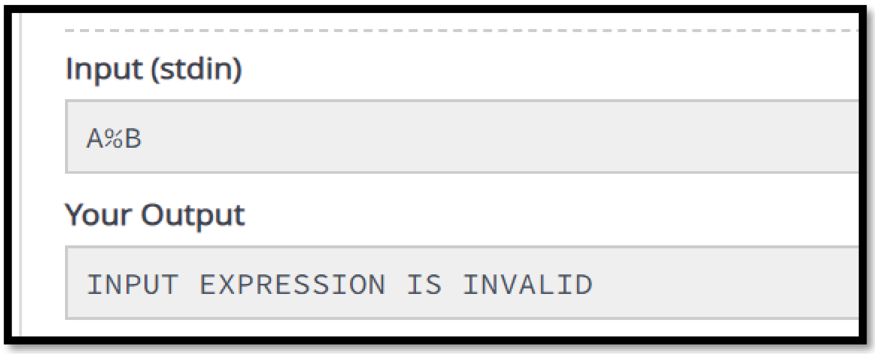
```
A+B^C^D/E-F
```

Your Output

```
ABCD^^E/+F-
```

TESTCASE	INPUT	OUTPUT
6	A%B	INPUT EXPRESSION IS INVALID

TESTCASE – 6: OUTPUT



Input (stdin)

```
A%B
```

Your Output

```
INPUT EXPRESSION IS INVALID
```

Q.2 Write a program to evaluate value of expression represented in postfix (reverse polish) notation.

Ans:

CODE:

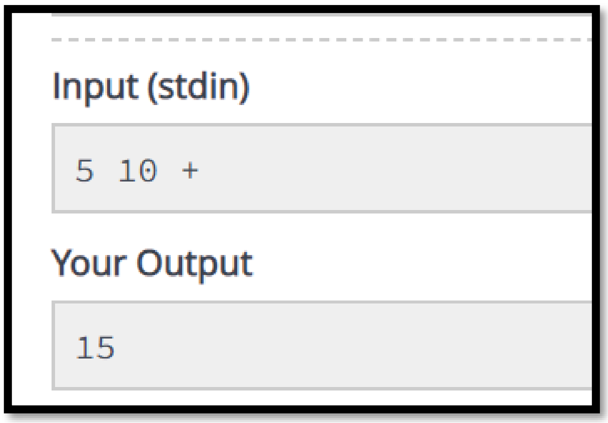
```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <ctype.h>

int top = 0;
int s[1000000];

void PUSH(int val) {
   //printf("PUSH is called\n");
   top++;
   s[top] = val;
}

int POP() {
   //printf("POP is called\n");
   if(top <= 0) {
      printf("Stack is Empty\n");
      exit(0);
   }
   top--;
   return s[top + 1];
}

int perform_operation(int op1,int op2,char operator) {
   //printf("Operation is called\n");
   int res;
   switch(operator) {
      case '+' : res = op1 + op2;
               break;
      case '-' : res = op1 - op2;
               break;
      case '*' : res = op1 * op2;
               break;
      case '/' : res = op1 / op2;
               break;
      case '^' : res = pow(op1,op2);
               break;
      case '%' : printf("INPUT EXPRESSION IS INVALID\n");
               exit(0);
   }
   return res;
}

int main() {
   char polish[1000000];
   //scanf("%s",polish);
```

```
    gets(polish);
    int length = strlen(polish);
    int i,op1,op2,value;
    //printf("%s\n",polish);

    for(i=0;i<length;i++) {
        if(polish[i] == ' ')  {
            //printf("Space\n");
            continue;
        }

        else if(isdigit(polish[i])) {
            //printf("Digit\n");
            int num = 0;
            while(isdigit(polish[i])) {
                num = num * 10 + (int) (polish[i] - '0');
                i++;
            }
            i--;
            PUSH(num);
        }
        else if(top > 1) {
            op2 = POP();
            op1 = POP();
            value = perform_operation(op1,op2,polish[i]);
            //printf("%d\n",value);

            PUSH(value);
        }
        else  {
            printf("INPUT EXPRESSION IS INVALID\n");
            exit(0);
        }
    }
    if(top != 1) {
        printf("INPUT EXPRESSION IS INVALID\n");
        exit(0);
    }
    else {
        printf("%d\n",POP());
    }


    return 0;
}
```

TESTCASE:

TESTCASE	INPUT	OUTPUT
1	5 10 +	15

<u>TESTCASE – 1: OUTPUT</u>


Input (stdin)

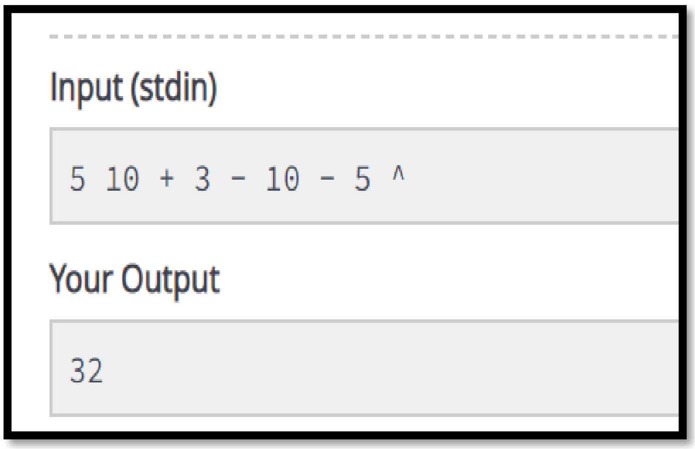
```
5 10 +
```

Your Output

```
15
```

TESTCASE	INPUT	OUTPUT
2	5 10 + 3 - 10 - 5 ^	32

TESTCASE – 2: OUTPUT


Input (stdin)

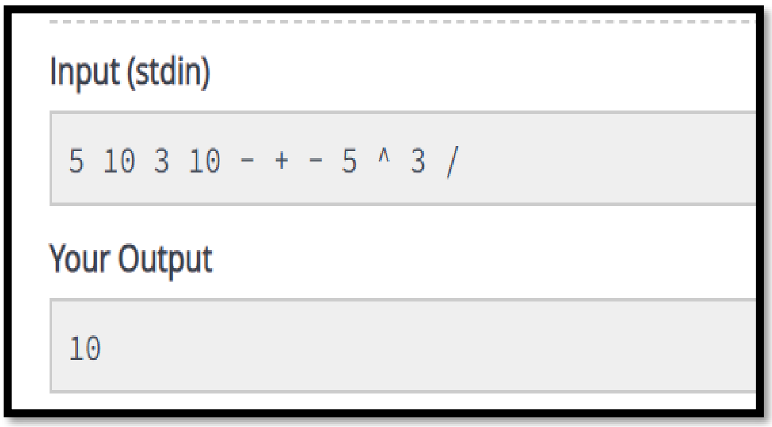
```
5 10 + 3 - 10 - 5 ^
```

Your Output

```
32
```

TESTCASE	INPUT	OUTPUT
3	5 10 3 10 - + - 5 ^ 3 /	10

TESTCASE – 3: OUTPUT


Input (stdin)

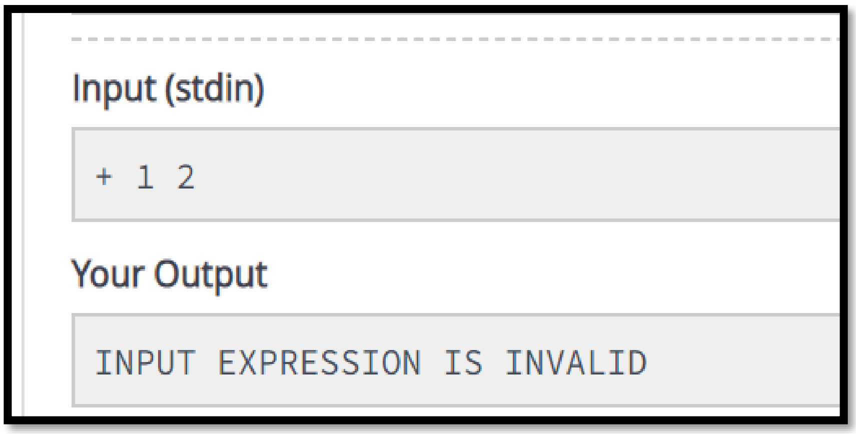
```
5 10 3 10 - + - 5 ^ 3 /
```

Your Output

```
10
```

TESTCASE	INPUT	OUTPUT
4	+ 1 2	INPUT EXPRESSION IS INVALID

TESTCASE – 4: OUTPUT



Input (stdin)

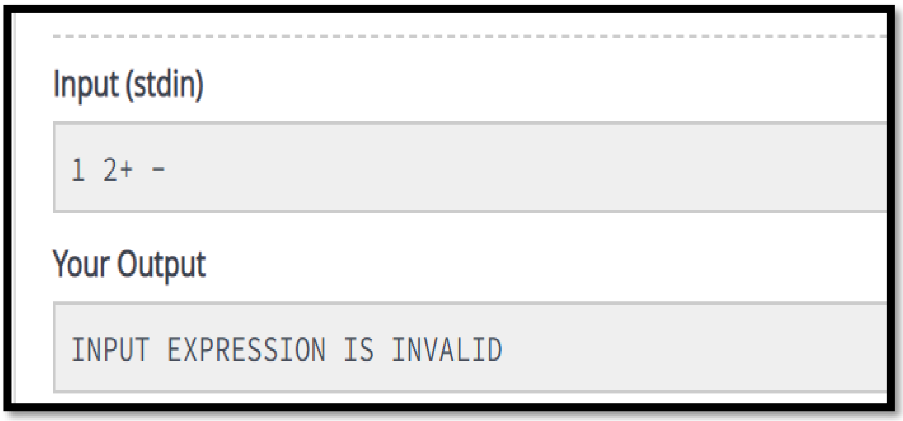
```
+ 1 2
```

Your Output

```
INPUT EXPRESSION IS INVALID
```

TESTCASE	INPUT	OUTPUT
5	1 2+ -	INPUT EXPRESSION IS INVALID

TESTCASE – 5: OUTPUT



Input (stdin)

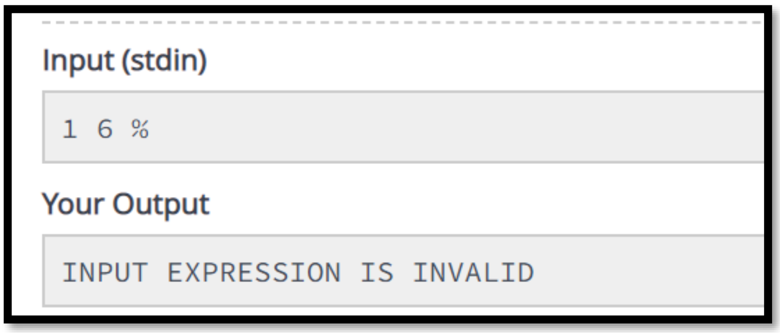
```
1 2+ -
```

Your Output

```
INPUT EXPRESSION IS INVALID
```

TESTCASE	INPUT	OUTPUT
6	1 6 %	INPUT EXPRESSION IS INVALID

TESTCASE – 6: OUTPUT



Input (stdin)

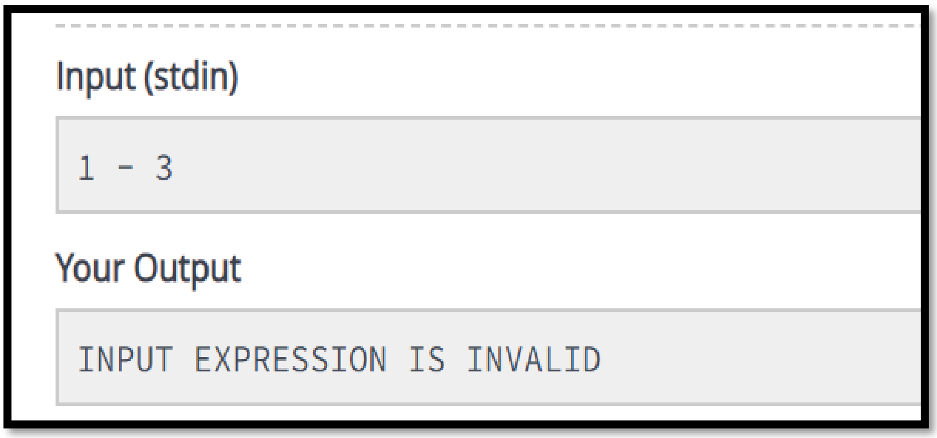
```
1 6 %
```

Your Output

```
INPUT EXPRESSION IS INVALID
```

TESTCASE	INPUT	OUTPUT
7	1 - 3	INPUT EXPRESSION IS INVALID

TESTCASE – 7: OUTPUT


Input (stdin)

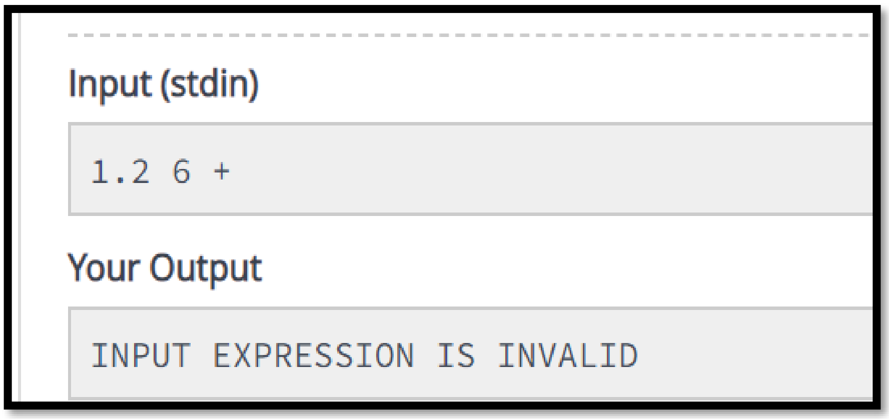
```
1 - 3
```

Your Output

```
INPUT EXPRESSION IS INVALID
```

TESTCASE	INPUT	OUTPUT
8	1.2 6 +	INPUT EXPRESSION IS INVALID

TESTCASE – 8: OUTPUT


Input (stdin)

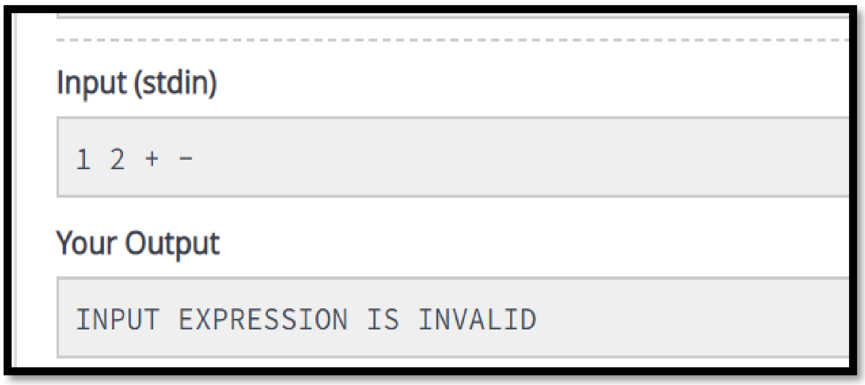
```
1.2 6 +
```

Your Output

```
INPUT EXPRESSION IS INVALID
```

TESTCASE	INPUT	OUTPUT
9	1 2 + -	INPUT EXPRESSION IS INVALID

TESTCASE – 9: OUTPUT


Input (stdin)

```
1 2 + -
```

Your Output

```
INPUT EXPRESSION IS INVALID
```