Data Structure and Algorithm, Spring 2024
Mini-Homework E

TA E-mail: dsa_ta@csie.ntu.edu.tw

Due: 13:00:00, Tuesday, April 16, 2024

Rules and Instructions

- Any form of cheating, lying, or plagiarism will not be tolerated. Students can get zero scores and/or fail the class and/or be kicked out of school and/or receive other punishments for those kinds of misconducts.
- Discussions on course materials and homework solutions are encouraged. But you should write the final solutions alone and understand them fully. Books, notes, and Internet resources (including chatGPT) can be consulted, but not copied from.
- Since everyone needs to write the final solutions alone, there is **absolutely no need to lend your homework solutions and/or source codes to your classmates at any time.** In order to maximize the level of fairness in this class, lending and borrowing homework solutions are both regarded as dishonest behaviors and will be punished according to the honesty policy.
- In each mini-homework, we will have one programming problem that is closely connected to what we have taught in the class. The purpose of the problem is for you to *practice* what you have learned, before conquering the more challenging problems in other homework sets.
- Like other programming problems, you should write your code in C programming language, and then submit the code via the *DSA Judge*:

  https://dsa2024.csie.org/.

  Each day, you can submit up to 5 times for each problem. The judge system will compile your code with

  ```
  gcc main.c -static -O2 -std=c11
  ```

- For debugging in the programming part, we strongly suggest you produce your own test-cases with programs and/or use tools like `gdb` or compile with flag `-fsanitize=address` to help you solve issues like illegal memory usage. For `gdb`, you are strongly encouraged to use compile flag `-g` to preserve symbols after compiling.

  Note: For students who use IDE (VScode, DevC++...) you can modify the compile command in the settings/preference section. Below are some examples:

  - `gcc main.c -O2 -std=c11 -fsanitize=address # Works on Unix-like OS`
  - `gcc main.c -O2 -std=c11 -g # use it with gdb`

- The score of the part that is submitted after the deadline will get some penalties according to the following rule:

$$Late\ Score = \max\left(\left(\frac{5 \times 86400 - Delay\ Time(sec.)}{5 \times 86400}\right) \times Original\ Score, 0\right)$$

- If you have questions about this mini-homework, please go to the the Discord channel and discuss the issues with your classmates. This is *strongly preferred* because it will provide everyone with a more interactive learning experience. If you really need an email answer, please follow the rules outlined below to get a fast response:

  - The subject should contain one tag, `"[hwe]"`, specifying the problem where you have questions. For example, `"[hwe] Will there be a repeated number?"`. Adding these tags allows the TAs to track the status of each email and to provide faster responses to you.
  - If you want to provide your code segments to us as part of your question, please upload it to Gist or similar platforms and provide the link. Please remember to protect your uploaded materials with proper permission settings. Screenshots or code segments directly included in the email are discouraged and may not be reviewed.

Programming Part

Arithmetic Expression (20 pts)

Problem Description

Given an infix arithmetic expression that contains non-negative integers, binary operators $+, -, *, /, \%$, and parentheses, convert it to the postfix expression by

INFIX-TO-POSTFIX($infix$)

```
1  S = empty stack
2  for each token in infix
3      if token is a number
4          output token
5      elseif token is an operator
6          if token is '('
7              PUSH(S, token)
8          elseif token is ')'
9              while NOT(IS-EMPTY(S)) and PEEP(S) is not '('
10                 output POP(S)
11             POP(S) // which is '('
12         else
13             while NOT(IS-EMPTY(S)) and PEEP(S) ≥ token in terms of precedence
14                 output POP(S)
15             PUSH(S, token)
16 while NOT(IS-EMPTY(S))
17     output POP(S)
```

Then, evaluate the value of the postfix expression by

POSTFIX-EVAL($postfix$)
1 S = empty stack
2 **for** each *token* in $postfix$
3 **if** *token* is a number
4 PUSH($S, token$)
5 **else //** *token* is an operator
6 $n2$ = POP(S)
7 $n1$ = POP(S)
8 PUSH(S, CALCULATE($n1, token, n2$))
9 **return** POP(S)

Note that $/, \%$ follows the *integer arithmetic* in C and evaluates to integer values.

Input

Each line contains a valid infix expression without any spaces.

Output

For each infix expression in the input, output a line of

`[postfix]=[value]`

where `[postfix]` is the postfix expression without spaces, and `[value]` is the final value.

Constraint

- There are always 3 expressions (i.e. 3 lines) in each testcase.
- The length of each expression will not exceed 4000.
- Each given number is a non-negative integer that holds within `unsigned int`.
- Within the operation of the POSTFIX-EVAL algorithm, any number will not exceed the range of `long long`.
- When running / or %, there will not be any division by zero.

Sample Testcases

Sample Input 1

```
8+9
8+9*6
8+9*6-4
```

Sample Output 1

```
89+=17
896*+=62
896*+4-=58
```

Sample Input 2

```
(8+9)*6
(8+9)*(6-4)
(8-9)%(6-4)
```

Sample Output 2

```
89+6*=102
89+64-*=34
89-64-%=-1
```

Sample Input 3

```
(8+9)/6%4
(8-9)/6%4
(8-9-6)%4
```

Sample Output 3

```
89+6/4%=2
89-6/4%=0
89-6-4%=-3
```

Hint

- By design, you can pass this homework by simulating the two algorithms properly. There is no need for other arithmetic calculations or cuts.