

Data Structures and Algorithms: Homework #3

Due on April 28, 2015 at 16:20

Instructors Hsuan-Tien Lin, Roger Jang

Tim Liou (b03902028)

3.1 Asymptotic Complexity

(1) Do Exercise R-4.28 of the textbook.

Consider $a_k \geq 0$ for $k \geq 0$

$$p(n) = a_0 + a_1n + a_2n^2 + a_3n^3 + \cdots + a_mn^m$$

For $n \geq (a_0 + a_1 + a_2 + \cdots + a_m) \geq 1$, we have

$$\begin{aligned} p(n) &\leq (a_0 + a_1 + a_2 + \cdots + a_m) \times n^m \\ \Rightarrow \log p(n) &\leq \log(a_0 + a_1 + a_2 + \cdots + a_m) + m \log n \\ &\leq \log n + m \log n \\ &= (m+1) \log n \end{aligned}$$

Take $c = m+1 > 0$, $n_0 = (a_0 + a_1 + a_2 + \cdots + a_m) \geq 1$

$$\log p(n) \leq c \log n \quad \text{for } n \geq n_0$$

That is, $\log p(n)$ is $O(\log n)$.

(2) Do Exercise R-4.34 of the textbook.

We have $f(n) > 1$ and $\lceil f(n) \rceil \leq f(n) + 1$ by definition. For $n \geq 1$,

$$\begin{aligned} \lceil f(n) \rceil &\leq f(n) + 1 \\ &\leq f(n) + f(n) \\ &= 2f(n) \end{aligned}$$

Take $c = 2 > 0$, $n_0 = 1 \geq 1$

$$\lceil f(n) \rceil \leq cf(n) \quad \text{for } n \geq n_0$$

That is, $\lceil f(n) \rceil$ is $O(f(n))$.

(3) Prove that $f(n) = \Theta(g(n))$.

By definition of limits at infinity,

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = A$$

means that for every $\epsilon > 0$ there is a corresponding N such that

$$\left| \frac{f(n)}{g(n)} - A \right| < \epsilon \quad \text{for } n > N$$

That is,

$$A - \epsilon < \frac{f(n)}{g(n)} < A + \epsilon \quad \text{for } n > N$$

Note that $g(n)$ is a strictly positive function. We have

$$(A - \epsilon)g(n) < f(n) < (A + \epsilon)g(n) \quad \text{for } n > N$$

Take $\epsilon \in (0, A)$, $c_1 = (A - \epsilon) > 0$, $c_2 = (A + \epsilon) > 0$, $n_0 > N$

$$c_1 g(n) \leq f(n) \leq c_2 g(n) \quad \text{for } n > n_0$$

This shows that $f(n) = \Theta(g(n))$.

(4) Do Exercise R-4.8 of the textbook.

If A is better than B for $n \geq n_0$, n_0 satisfies the following statement.

$$2n_0^3 - 40n_0^2 > 0$$

We can easily find that $n_0 > 20$. We choose $n_0 = 21$. It is a possible value for n_0 satisfying the statement that A is better than B for $n \geq n_0$.

(5) Do Exercise C-4.16(b) of the textbook.

This is the pseudo code of the Horner's method.

```

1: function HORNER'S-METHOD( $x$ ,  $CoefficientsOfPolynomial$ ,  $DegreeOfPolynomial$ )
2:    $Sum \leftarrow 0$ 
3:   for all  $CoefficientsOfPolynomial$  do
4:      $Sum \leftarrow Sum \times x + CoefficientsOfPolynomial$ 
5:   end for
6:   return  $Sum$ 
7: end function
```

Algorithm 1: Horner's method for computing polynomial

We can find there is only one **for loop** in this pseudo code, that is, the number of arithmetic operations is $O(n)$.

(6) Consider some $f(n)$ and $g(n)$ such that $\lg f(n) = O(\lg g(n))$ and $g(n) \geq 2$ for $n \geq 1$. Construct a counter-example to disprove that $f(n) = O(g(n))$.

Consider $f(n) = 4^n$, $g(n) = 2^n$, for $n \geq 1$, we can find

$$\begin{aligned}\lg f(n) &= n \lg 4 \\ &\leq n \lg 2 \\ &= 4 \lg 2^n \\ &= 4 \lg g(n)\end{aligned}$$

Take $c = 4 > 0$, $n_0 = 1 \geq 1$

$$\lg f(n) \leq c \lg g(n) \quad \text{for } n \geq n_0$$

That is, $\lg f(n)$ is $O(\lg g(n))$. Note that $g(n) \geq 2$ for $n \geq 1$.

If $f(n) = O(g(n))$, $\exists n_0 > 0, c > 0 \ni$

$$4^n \leq c 2^n \quad \text{for } n \geq n_0$$

Take \log_2 on both sides,

$$\begin{aligned}2n &\leq \log_2 c + n \quad \text{for } n \geq n_0 \\ \Rightarrow \log_2 c &\geq n\end{aligned}$$

That is, take $n' = \max(n_0, \lceil \log_2 c + 1 \rceil)$

$$\begin{aligned}n' \geq n_0 &\Rightarrow 4^{n'} \leq c 2^{n'} \\ n' > \log_2 c &\Rightarrow 4^{n'} > c 2^{n'}\end{aligned}$$

This is a contradiction. Therefore, we disprove that $f(n) = O(g(n))$.

3.2 Stack, Queue, Deque

(1) Do Exercise C-5.2 of the textbook.

Pop out the elements in the stack one by one and check if it is equal to element x . After that, enqueue the elements into the queue one by one. Use a variable to store the number of the elements we popped from the stack. Once we find the certain element or the stack is empty, we push the elements into stack from queue, and then enqueue the same number of element into queue from stack. Finally push all these elements from queue into stack again. This will maintain elements' original order.

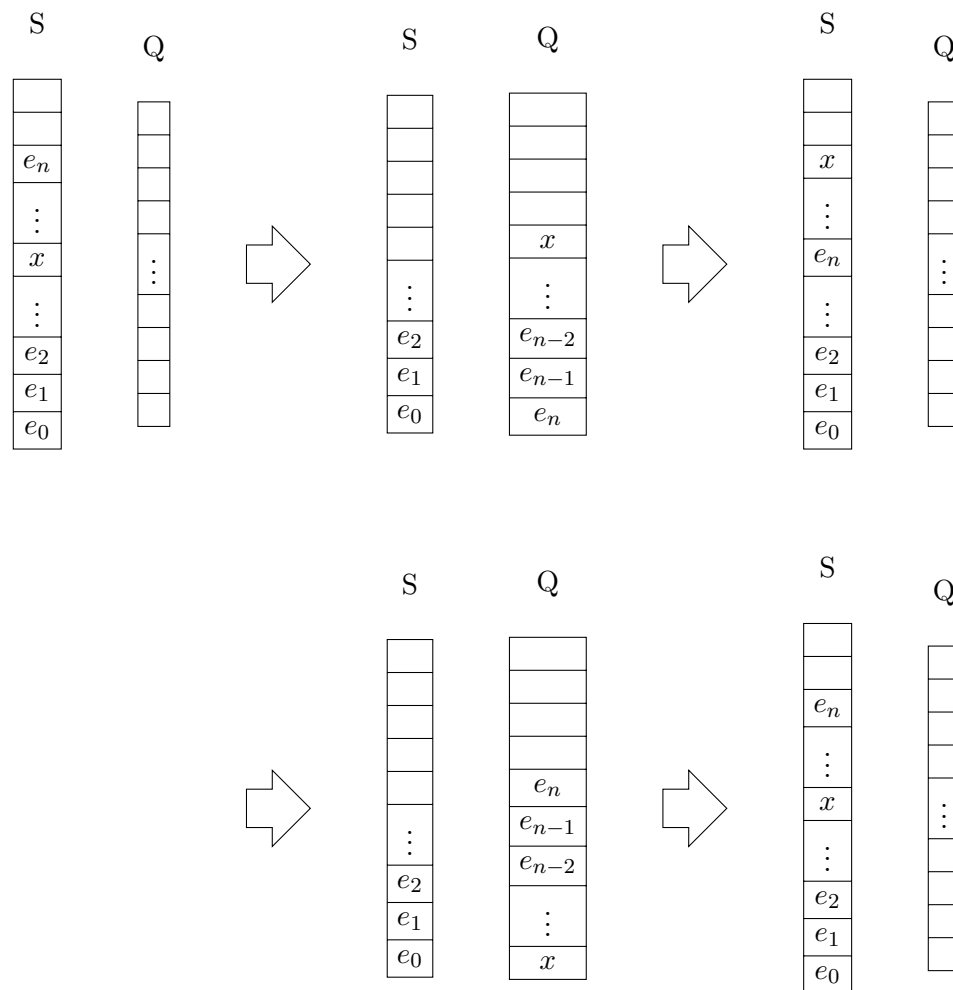


Figure 1: How this algorithm works

(2) Do Exercise C-5.9 of the textbook.

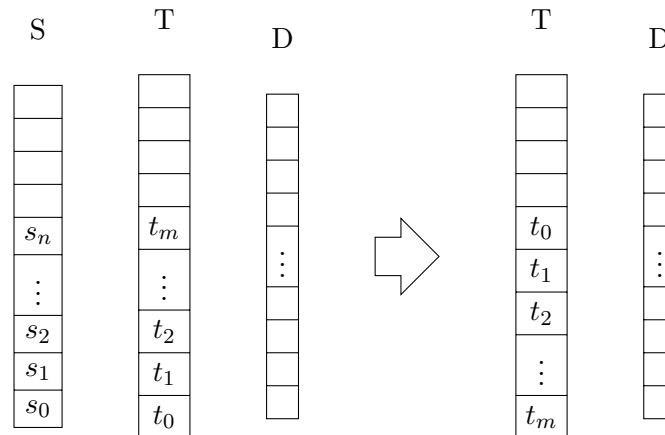


Figure 2: **Pop** all elements from T and **Push_front** them to D. Then **Pop_back** them back to T.

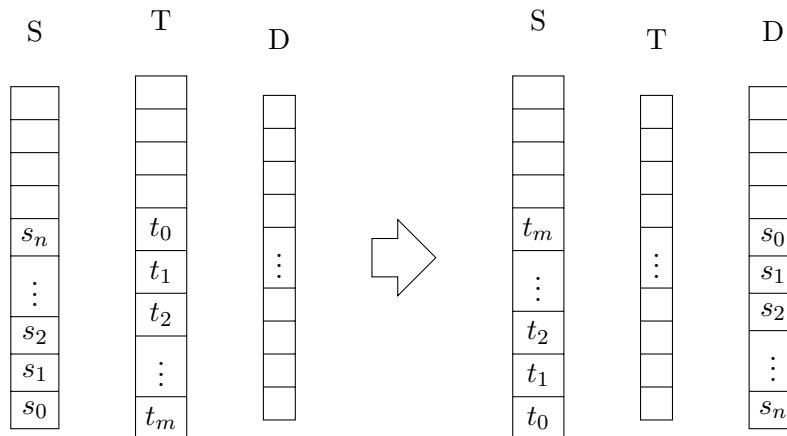


Figure 3: **Pop** all elements from S and **Push_front** them to D. Then **Pop** all elements from T to S.

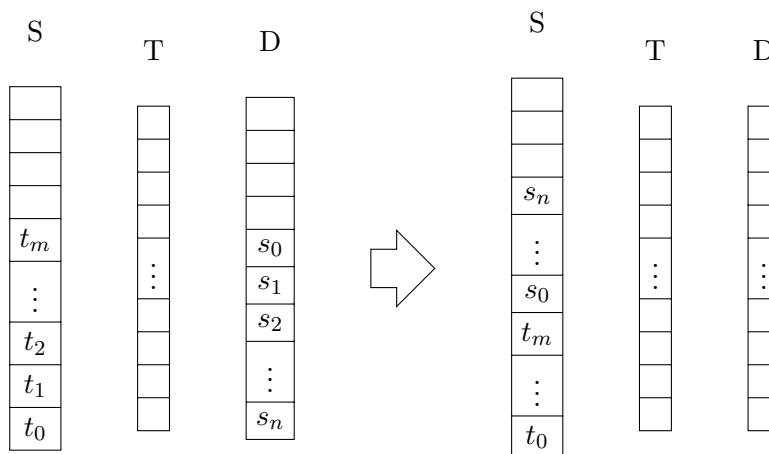


Figure 4: **Pop_front** all elements from D to S

(3) Use any pseudocode to write down an algorithm that uses two stacks (with push, pop and isempty operations but no others) to simulate one deque (for push/pop front and push/pop back operations). What is the total running time after N operations?

Imagine we divide a deque into two stacks named S_f and S_b . **PushFront**, **PopFront** are processed in S_f while **PushBack**, **PopBack** are processed in S_b . However, we need to transport elements from a stack to the other one if we want to **Pop** elements from an empty stack. Note that both stacks are empty means the deque is empty.

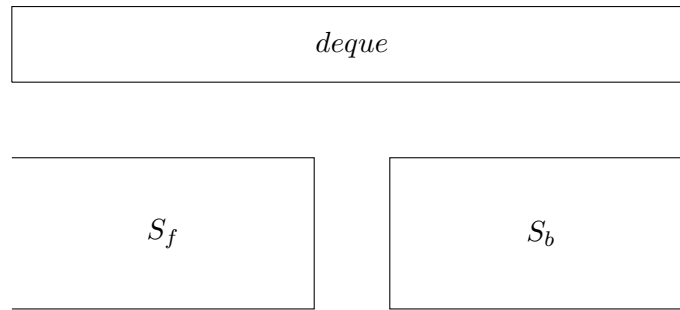


Figure 5: Use two stacks to simulate a deque

```

1: function POPFRONT
2:   if  $S_f$  and  $S_b$  aren't both empty then
3:     if  $S_f$  is empty then
4:       pop all elements from  $S_b$  to  $S_f$ 
5:     end if
6:     pop from  $S_f$ 
7:   end if
8: end function

```

Algorithm 2: PopFront of deque using two stacks

```

1: function POPBACK
2:   if  $S_f$  and  $S_b$  aren't both empty then
3:     if  $S_b$  is empty then
4:       pop all elements from  $S_f$  to  $S_b$ 
5:     end if
6:     pop from  $S_b$ 
7:   end if
8: end function

```

Algorithm 3: PopBack of deque using two stacks

```

1: function PUSHFRONT( $e$ )
2:   push  $e$  into  $S_f$ 
3: end function

```

Algorithm 4: PushFront

```

1: function PUSHBACK( $e$ )
2:   push  $e$  into  $S_b$ 
3: end function

```

Algorithm 5: PushBack

Suppose **Pop**/**Push** both take t (Time Unit). There are some cases result in different running time.

Case 1: all the operations are either **PushBack** or **PushFront**

Since these two operations are just a **Push** operation for a stack, the time complexity is constant. After N operation, the total running time is simply Nt .

Case 2: operations contain **PopBack** or **PopFront**, but never make any stack empty

Like Case 1, all the operations are just **Push** or **Pop** for a stack. Therefore, the total running time is also Nt .

Case 3: operations contain **PopBack** or **PopFront**, and try to **Pop** from an empty stack

In this situation, it would **Pop** all elements from the other stack to its first then **Pop** the desired elements. Suppose this situation happened k times. We denote the number of elements from the other stack in the i th time a_i . The total time is $N + \sum_{i=1}^k a_i$

(4) Do Exercise C-5.9 of the textbook, but with three stacks instead of two stacks and one deque.

Suppose three stacks are big enough for all elements.

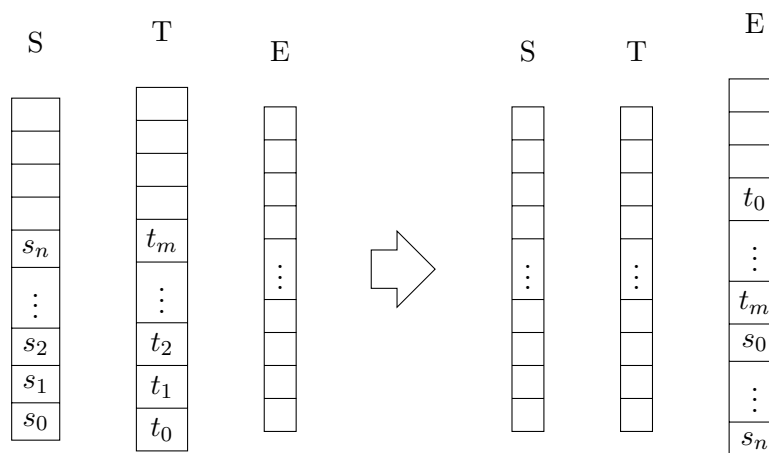


Figure 6: **Pop** all elements from S to E and then **Pop** all elements from T to E.

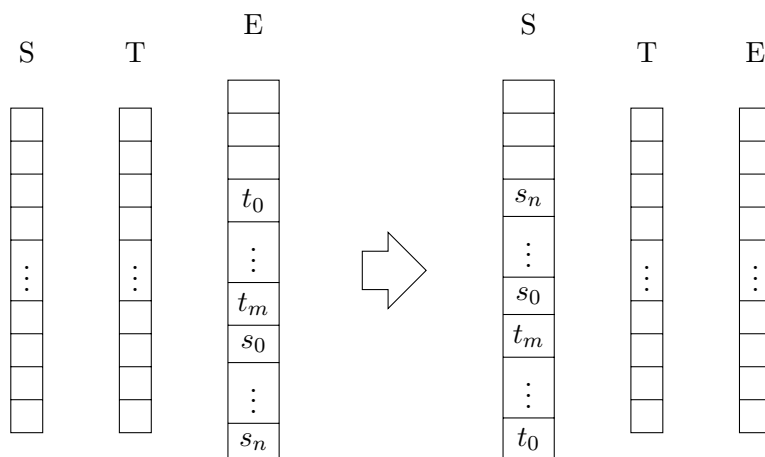


Figure 7: **Pop** all elements from E to S

3.3 List, Iterator

(1) Do Exercise C-6.7 of the textbook.

===== Pending =====

(2) Do Exercise C-6.13 of the textbook by Googling the *Knuth Shuffle*.

Knuth Shuffle guarantees that every possible ordering is equally likely. The running time of this function is $O(n)$, n is the number of cards.

```

1: function KNUTH-SHUFFLE( $V, LengthOfV$ )
2:   for  $i \leftarrow 0$  to  $LengthOfV - 1$  do
3:      $r \leftarrow \text{RANDOMINTEGER}(i + 1)$ 
4:     Exchange  $V[i]$  and  $V[r]$ 
5:   end for
6: end function

```

Algorithm 6: Knuth-Shuffle

3.4 Calculators

(1) Three test case of integer calculator

```

1  Input:
2
3  123+2*3*(5-3+48/2) + 34
4  ~( (1024 >> (23 % 3)) + !0)
5  ((3 && +1 || 1) ^ 12 << 1 ) & 16 | 2
6  ---
7  Output:
8
9  --- postfix expression transforming ---
10 encounter 123: push to output
11     current output: 123
12 encounter +: push to the stack directly
13     current output: 123
14     current stack: +
15 encounter 2: push to output
16     current output: 123 2
17     current stack: +
18 encounter *: push to the stack directly
19     current output: 123 2
20     current stack: + *
21 encounter 3: push to output
22     current output: 123 2 3
23     current stack: + *
24 encounter *: stack.top() has greater or the same precedence, after pop
    something out to output, then push to the stack
25     current output: 123 2 3 *
26     current stack: + *
27 encounter (: push to the stack directly
28     current output: 123 2 3 *
29     current stack: + * (
30 encounter 5: push to output

```

```

31         current output: 123 2 3 * 5
32         current stack: + * (
33     encounter -: push to the stack directly
34         current output: 123 2 3 * 5
35         current stack: + * ( -
36     encounter 3: push to output
37         current output: 123 2 3 * 5 3
38         current stack: + * ( -
39     encounter +: stack.top() has greater or the same precedence, after pop
        something out to output, then push to the stack
40         current output: 123 2 3 * 5 3 -
41         current stack: + * ( +
42     encounter 48: push to output
43         current output: 123 2 3 * 5 3 - 48
44         current stack: + * ( +
45     encounter /: push to the stack directly
46         current output: 123 2 3 * 5 3 - 48
47         current stack: + * ( + /
48     encounter 2: push to output
49         current output: 123 2 3 * 5 3 - 48 2
50         current stack: + * ( + /
51     encounter ): flush the stack to output until meeting '('
52         current output: 123 2 3 * 5 3 - 48 2 / +
53         current stack: + *
54     encounter +: stack.top() has greater or the same precedence, after pop
        something out to output, then push to the stack
55         current output: 123 2 3 * 5 3 - 48 2 / + * +
56         current stack: +
57     encounter 34: push to output
58         current output: 123 2 3 * 5 3 - 48 2 / + * + 34
59         current stack: +
60     encounter NOTHING: flush the stack to output
61         current output: 123 2 3 * 5 3 - 48 2 / + * + 34 +
62     --- postfix expression transforming complete :) ---
63     Postfix Exp: 123 2 3 * 5 3 - 48 2 / + * + 34 +
64     RESULT: 313
65     --- postfix expression transforming ---
66     encounter U-: push to the stack directly
67         current output:
68         current stack: -
69     encounter ~: push to the stack directly
70         current output:
71         current stack: - ~

```

```

72     encounter (: push to the stack directly
73         current output:
74         current stack: - ~ (
75     encounter (: push to the stack directly
76         current output:
77         current stack: - ~ ( (
78     encounter 1024: push to output
79         current output: 1024
80         current stack: - ~ ( (
81     encounter >>: push to the stack directly
82         current output: 1024
83         current stack: - ~ ( ( >>
84     encounter (: push to the stack directly
85         current output: 1024
86         current stack: - ~ ( ( >> (
87     encounter 23: push to output
88         current output: 1024 23
89         current stack: - ~ ( ( >> (
90     encounter %: push to the stack directly
91         current output: 1024 23
92         current stack: - ~ ( ( >> ( %
93     encounter 3: push to output
94         current output: 1024 23 3
95         current stack: - ~ ( ( >> ( %
96     encounter ): flush the stack to output until meeting '('
97         current output: 1024 23 3 %
98         current stack: - ~ ( ( >>
99     encounter ): flush the stack to output until meeting '('
100        current output: 1024 23 3 % >>
101        current stack: - ~ (
102    encounter +: push to the stack directly
103        current output: 1024 23 3 % >>
104        current stack: - ~ ( +
105    encounter !: push to the stack directly
106        current output: 1024 23 3 % >>
107        current stack: - ~ ( + !
108    encounter 0: push to output
109        current output: 1024 23 3 % >> 0
110        current stack: - ~ ( + !
111    encounter ): flush the stack to output until meeting '('
112        current output: 1024 23 3 % >> 0 ! +
113        current stack: - ~
114    encounter NOTHING: flush the stack to output

```

```

115         current output: 1024 23 3 % >> 0 ! + ~ -
116     --- postfix expression transforming complete :) ---
117     Postfix Exp: 1024 23 3 % >> 0 ! + ~ -
118     RESULT: 258
119     --- postfix expression transforming ---
120     encounter (: push to the stack directly
121         current output:
122         current stack: (
123     encounter (: push to the stack directly
124         current output:
125         current stack: ( (
126     encounter 3: push to output
127         current output: 3
128         current stack: ( (
129     encounter &&: push to the stack directly
130         current output: 3
131         current stack: ( ( &&
132     encounter U+: push to the stack directly
133         current output: 3
134         current stack: ( ( && +
135     encounter 1: push to output
136         current output: 3 1
137         current stack: ( ( && +
138     encounter ||: stack.top() has greater or the same precedence, after pop
        something out to output, then push to the stack
139         current output: 3 1 + &&
140         current stack: ( ( ||
141     encounter 1: push to output
142         current output: 3 1 + && 1
143         current stack: ( ( ||
144     encounter ): flush the stack to output until meeting '('
145         current output: 3 1 + && 1 ||
146         current stack: (
147     encounter ^: push to the stack directly
148         current output: 3 1 + && 1 ||
149         current stack: ( ^
150     encounter 12: push to output
151         current output: 3 1 + && 1 || 12
152         current stack: ( ^
153     encounter <<: push to the stack directly
154         current output: 3 1 + && 1 || 12
155         current stack: ( ^ <<
156     encounter 1: push to output

```

```

157         current output: 3 1 + && 1 || 12 1
158         current stack: ( ^ <<
159     encounter ): flush the stack to output until meeting '('
160         current output: 3 1 + && 1 || 12 1 << ^
161     encounter &: push to the stack directly
162         current output: 3 1 + && 1 || 12 1 << ^
163         current stack: &
164     encounter 16: push to output
165         current output: 3 1 + && 1 || 12 1 << ^ 16
166         current stack: &
167     encounter |: stack.top() has greater or the same precedence, after pop
        something out to output, then push to the stack
168         current output: 3 1 + && 1 || 12 1 << ^ 16 &
169         current stack: |
170     encounter 2: push to output
171         current output: 3 1 + && 1 || 12 1 << ^ 16 & 2
172         current stack: |
173     encounter NOTHING: flush the stack to output
174         current output: 3 1 + && 1 || 12 1 << ^ 16 & 2 |
175     --- postfix expression transforming complete :) ---
176     Postfix Exp: 3 1 + && 1 || 12 1 << ^ 16 & 2 |
177     RESULT: 18

```

(2) Three test case of scientific calculator

```

1     Input:
2
3     - pow( (2.3 + 3) *2 , exp( log(2) ) )
4     sqrt(1/16) + fabs(sin(3 / 2 * 3.1415926)) + +cos(3.1415926)
5     0.00 + 1.2
6     ---
7     Output:
8
9     --- postfix expression transforming ---
10    encounter U-: push to the stack directly
11        current output:
12        current stack: -
13    encounter pow: push to the stack directly
14        current output:
15        current stack: - pow
16    encounter (: push to the stack directly
17        current output:
18        current stack: - pow (
19    encounter (: push to the stack directly

```

```

20         current output:
21         current stack: - pow ( (
22 encounter 2.3: push to output
23         current output: 2.300000
24         current stack: - pow ( (
25 encounter +: push to the stack directly
26         current output: 2.300000
27         current stack: - pow ( ( +
28 encounter 3: push to output
29         current output: 2.300000 3.000000
30         current stack: - pow ( ( +
31 encounter ): flush the stack to output until meeting '('
32         current output: 2.300000 3.000000 +
33         current stack: - pow (
34 encounter *: push to the stack directly
35         current output: 2.300000 3.000000 +
36         current stack: - pow ( *
37 encounter 2: push to output
38         current output: 2.300000 3.000000 + 2.000000
39         current stack: - pow ( *
40 encounter exp: push to the stack directly
41         current output: 2.300000 3.000000 + 2.000000
42         current stack: - pow ( * exp
43 encounter (: push to the stack directly
44         current output: 2.300000 3.000000 + 2.000000
45         current stack: - pow ( * exp (
46 encounter log: push to the stack directly
47         current output: 2.300000 3.000000 + 2.000000
48         current stack: - pow ( * exp ( log
49 encounter (: push to the stack directly
50         current output: 2.300000 3.000000 + 2.000000
51         current stack: - pow ( * exp ( log (
52 encounter 2: push to output
53         current output: 2.300000 3.000000 + 2.000000 2.000000
54         current stack: - pow ( * exp ( log (
55 encounter ): flush the stack to output until meeting '(' and pop function 'log
    ' to output
56         current output: 2.300000 3.000000 + 2.000000 2.000000 log
57         current stack: - pow ( * exp (
58 encounter ): flush the stack to output until meeting '(' and pop function 'exp
    ' to output
59         current output: 2.300000 3.000000 + 2.000000 2.000000 log exp
60         current stack: - pow ( *

```

```

61     encounter ): flush the stack to output until meeting '(' and pop function 'pow
        ' to output
62         current output: 2.300000 3.000000 + 2.000000 2.000000 log exp * pow
63         current stack: -
64     encounter NOTHING: flush the stack to output
65         current output: 2.300000 3.000000 + 2.000000 2.000000 log exp * pow -
66     --- postfix expression transforming complete :) ---
67     Postfix Exp: 2.300000 3.000000 + 2.000000 2.000000 log exp * pow -
68     RESULT: -789.048100
69     --- postfix expression transforming ---
70     encounter sqrt: push to the stack directly
71         current output:
72         current stack: sqrt
73     encounter (: push to the stack directly
74         current output:
75         current stack: sqrt (
76     encounter 1: push to output
77         current output: 1.000000
78         current stack: sqrt (
79     encounter /: push to the stack directly
80         current output: 1.000000
81         current stack: sqrt ( /
82     encounter 16: push to output
83         current output: 1.000000 16.000000
84         current stack: sqrt ( /
85     encounter ): flush the stack to output until meeting '(' and pop function '
        sqrt' to output
86         current output: 1.000000 16.000000 / sqrt
87     encounter +: push to the stack directly
88         current output: 1.000000 16.000000 / sqrt
89         current stack: +
90     encounter fabs: push to the stack directly
91         current output: 1.000000 16.000000 / sqrt
92         current stack: + fabs
93     encounter (: push to the stack directly
94         current output: 1.000000 16.000000 / sqrt
95         current stack: + fabs (
96     encounter sin: push to the stack directly
97         current output: 1.000000 16.000000 / sqrt
98         current stack: + fabs ( sin
99     encounter (: push to the stack directly
100        current output: 1.000000 16.000000 / sqrt
101        current stack: + fabs ( sin (

```



```

102   encounter 3: push to output
103       current output: 1.000000 16.000000 / sqrt 3.000000
104       current stack: + fabs ( sin (
105   encounter /: push to the stack directly
106       current output: 1.000000 16.000000 / sqrt 3.000000
107       current stack: + fabs ( sin ( /
108   encounter 2: push to output
109       current output: 1.000000 16.000000 / sqrt 3.000000 2.000000
110       current stack: + fabs ( sin ( /
111   encounter *: stack.top() has greater or the same precedence, after pop
        something out to output, then push to the stack
112       current output: 1.000000 16.000000 / sqrt 3.000000 2.000000 /
113       current stack: + fabs ( sin ( *
114   encounter 3.1415926: push to output
115       current output: 1.000000 16.000000 / sqrt 3.000000 2.000000 / 3.141593
116       current stack: + fabs ( sin ( *
117   encounter ): flush the stack to output until meeting '(' and pop function 'sin
        ' to output
118       current output: 1.000000 16.000000 / sqrt 3.000000 2.000000 / 3.141593
        * sin
119       current stack: + fabs (
120   encounter ): flush the stack to output until meeting '(' and pop function '
        fabs' to output
121       current output: 1.000000 16.000000 / sqrt 3.000000 2.000000 / 3.141593
        * sin fabs
122       current stack: +
123   encounter +: stack.top() has greater or the same precedence, after pop
        something out to output, then push to the stack
124       current output: 1.000000 16.000000 / sqrt 3.000000 2.000000 / 3.141593
        * sin fabs +
125       current stack: +
126   encounter U+: push to the stack directly
127       current output: 1.000000 16.000000 / sqrt 3.000000 2.000000 / 3.141593
        * sin fabs +
128       current stack: + +
129   encounter cos: push to the stack directly
130       current output: 1.000000 16.000000 / sqrt 3.000000 2.000000 / 3.141593
        * sin fabs +
131       current stack: + + cos
132   encounter (: push to the stack directly
133       current output: 1.000000 16.000000 / sqrt 3.000000 2.000000 / 3.141593
        * sin fabs +
134       current stack: + + cos (

```

```
135     encounter 3.1415926: push to output
136         current output: 1.000000 16.000000 / sqrt 3.000000 2.000000 / 3.141593
137             * sin fabs + 3.141593
138         current stack: + + cos (
139 encounter ): flush the stack to output until meeting '(' and pop function 'cos
140     ' to output
141         current output: 1.000000 16.000000 / sqrt 3.000000 2.000000 / 3.141593
142             * sin fabs + 3.141593 cos
143         current stack: + +
144 encounter NOTHING: flush the stack to output
145         current output: 1.000000 16.000000 / sqrt 3.000000 2.000000 / 3.141593
146             * sin fabs + 3.141593 cos + +
147 --- postfix expression transforming complete :) ---
148 Postfix Exp: 1.000000 16.000000 / sqrt 3.000000 2.000000 / 3.141593 * sin fabs
149     + 3.141593 cos + +
150 RESULT: 0.250000
151 --- postfix expression transforming ---
152 encounter 0.00: push to output
153     current output: 0.000000
154 encounter +: push to the stack directly
155     current output: 0.000000
156     current stack: +
157 encounter 1.2: push to output
158     current output: 0.000000 1.200000
159     current stack: +
160 encounter NOTHING: flush the stack to output
161     current output: 0.000000 1.200000 +
162 --- postfix expression transforming complete :) ---
163 Postfix Exp: 0.000000 1.200000 +
164 RESULT: 1.200000
```