Data Structure HW3

1

(a)

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	S	1	1	1	4	5	1	1	8	1	22	1	25	1			1
1	1	1	2	3	1	15	6	7	1	1	21	23	24	1	1		1
1	1	1	1	1	14	1	1	9	1	1	20	1	1	1	1		1
1	76	75	1	1	1	13	12	10	1	1	19	1	1	1	1	1	1
1	1	74	1	1	1	1	11	1	1	18	26	1	1	33	1	37	1
1	77	73	1	66	67	1	80	16	17	27	1	1	1	32	1	36	1
1	1	72	1	65	1	1	1	60	1	28	1	30	31	34	35	38	1
1	78	71	1	1	64	1	59	1	1	1	29	1	1	1	1	39	1
1	1	70	1	63	1	61	58	57	54	55	56	1	46	44	42	40	1
1	79	69	68	1	62	1	1	1	53	1	1	1	45	43	41	1	1
1	1	1	1	1	1	1	1	1	52	51	50	49	1	47	48	G	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

(b)

Cell	i (of stack[i])	$\langle x_i, y_i, dir \rangle$
(2,8)	6	$\langle 2, 7, 2 \rangle$
(4,7)	10	$\langle 5, 7, 0 \rangle$
(7,13)	16	$\langle 7, 12, 2 \rangle$
(9,7)	21	$\langle 8, 13, 5 \rangle$
(10,15)	33	$\langle 10, 16, 6 \rangle$

dir

7 0 1
6 2
5 4 3

2

(a)

8 4 2 + 5 3 - / * 4 3 2 5 - + / +

(b)

a b k - + m n p * / n + d e + / *

3

(1*2)+(3-(4+(5-((6+7)+(8-9)))) = 8

4

((a+b)*c)/(d-e)

5

(a)

```
string infix_to_prefix(string infix) {
    stack st_oper;
    string prefix;
    reverse(infix.begin(), infix.end());

    for(int i = 0; i < infix.size(); i++) {
        if(infix[i] == '(') infix[i] = ')';
        if(infix[i] == ')') infix[i] = '(';
    }

    //做postfix
    for(int i = 0; i < infix.size(); i++) {
        if(infix[i] is number) {
            prefix += infix[i];
        }
        if(infix[i] is operator) {
            if(infix[i] == ')') {
                while(st_oper.top() != '(') {
                    prefix += st._oper.top();
                }
            }
            else {
                while(!st_oper.empty() && isp(st_oper.top()) >= icp(infix[i]))
                    prefix += st._oper.top();
                }
                st_oper.push(infix[i]);
            }
        }
    }
    while(!st_oper.empty()) {
        st_oper.push(infix[i]);
    }
    reverse(prefix.begin(), prefix.end());
    return prefix;
}
```

(b)

(a+b)/(c*d)/e-f
-> reverse -> f-e/)d*c(/)b+a(
-> transform -> f-e/(d*c)/(b+a)

做infix to postfix

當前字符	字符類型	st_oper	prefix	reason
f	運算元		f	因f為運算元，所以直接加入prefix
-	運算子	-	f	因st_oper為空，所以'-'直接加入st_oper
e	運算元	-	fe	因e為運算元，所以直接加入prefix
/	運算子	-/	fe	因isp('-') < icp('/')，所以'/'加入st_oper
(	運算子	-/(	fe	因isp('/') < icp('('），所以'('加入st_oper
d	運算元	-/(	fed	因d為運算元，所以直接加入prefix
*	運算子	-/(*	fed	因isp('(') < icp('*')，所以'*'加入st_oper
c	運算元	-/(*	fedc	因c為運算元，所以直接加入prefix
)	運算子	-/	fedc*	因遇到')'，因此直接output直到'('
/	運算子	-/	fedc*/	因isp('/') = icp('/')，所以原先'/'加入prefix，然後當前'/'加入st_oper
(	運算子	-/(	fedc*/	因isp('/') < icp('('），所以'('加入
b	運算元	-/(	fedc*/b	因b為運算元，所以直接加入prefix
+	運算子	-/(+	fedc*/b	因isp('(') < icp('+')，所以'+'加入st_oper
a	運算元	-/(+	fedc*/ba	因a為運算元，所以直接加入prefix
)	運算子	-/	fedc*/ba+	因遇到')'，因此直接output直到'('
			fedc*/ba+/-	最終將st_oper裡所有的運算元pop

fedc*/ba+/-
-> reverse -> -/+ab/*cde

prefix: -/+ab/*cde