蛟龙四班

递归与栈

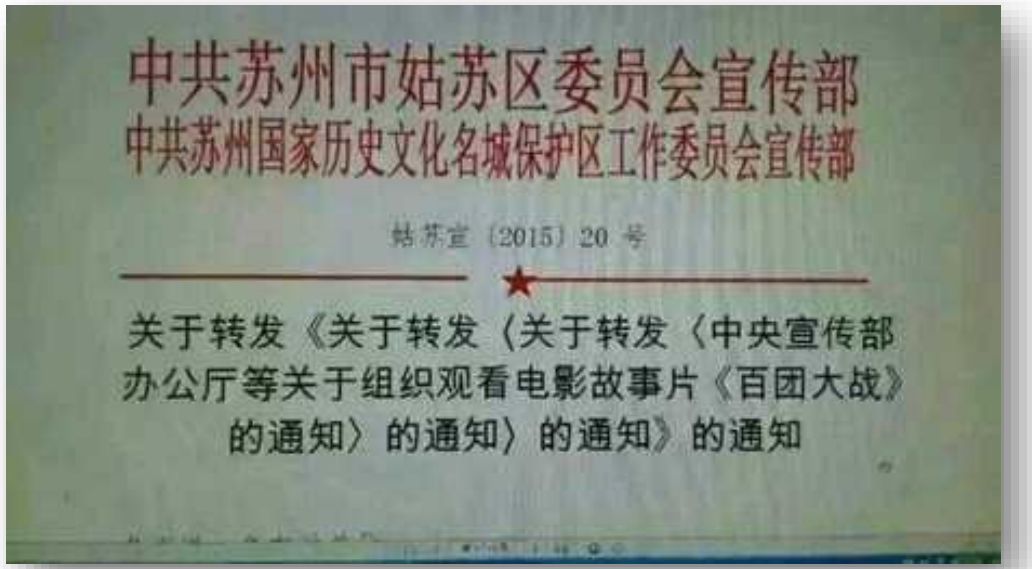
Mas

递归

要理解递归,就得先理解什么是递归。

递归,在数学和计算机科学中是指在函数的定义中使用函数自身的**方法**

在计算机科学中还额外指一种通过重复将问题分解为同类的子问题而解决问题的方法



#1767、弹簧板

题目描述

有一个小球掉落在一串连续的弹簧板上

小球落到某一个弹簧板后,会被弹到某一个地点,直到小球被弹到弹簧板以外的地方

假设有 n 个连续的弹簧板,每个弹簧板占一个单位距离

第 i 个弹簧板能选择把小球向前弹 a_i 个距离,或者向前弹 b_i 个距离

比如位置 1 的弹簧能让小球前进 2 个距离到达位置 3

现在小球掉到了 1 号弹簧板上面,请你帮忙计算小球最少会被弹起多少次,才会弹出弹簧板

> 1 *号弹簧板也算一次*

输入格式

第一个行输入一个 n 代表一共有 n 个弹簧板

第二行输入 n 个数字,中间用空格分开

第 i 个数字 a_i 代表第 i 个弹簧板可以让小球移动的距离

第三行输入 n 个数字,中间用空格分开

第 i 个数字 b_i 代表第 i 个弹簧板可以让小球移动的距离

输出格式

输出一个整数,表示小球被弹起的最小次数

数据规模

对于全部的数据 $1 \leq n \leq 200, 0 < a_i \leq 30, 0 < b_i \leq 30$

设 f_x 为 当前小球在 x 号格子,弹出弹簧板的最少步数

若选择 向前 弹 a_x 个格子 , 那么将到达 $x + a_x$

若选择 向前 弹 b_x 个格子 , 那么将到达 $x + b_x$

那么有

$$f_x = \begin{cases} 1 + \min(f_{x+a_x}, f_{x+b_x}) & x \leq n \\ 0 & x > n \end{cases}$$

递归求解即可

#2396、字符串弱等于

题目描述

如果两字符串 a , b 满足下面两个条件之一,我们称 a 弱等于 b ,或者 b 弱等于 a

- a 等于 b .
- a , b 的长度相同,并且长度都为偶数,把字符串 a 串从正中间拆开,拆成 a_1 , a_2 ,同样把 b 串拆成 b_1 , b_2 。 a_1 弱等于 b_1 并且 a_2 弱等于 b_2 ,或者 a_1 弱等于 b_2 并且 a_2 弱等于 b_1 .

输入格式

第一行输入一个正整数 T

接下来输入 T 组数据,每组数量两行字符串,字符串只包含小写字母

输出格式

对于每组数据输出一行,如果满足若等于输出 `YES` ,否则输出 `NO`

数据规模

对于全部的数据 $1 \leq T \leq 100, 1 \leq |a|, |b| \leq 1000$

样例输入

```
1
abbaabba
baababab
```

样例输出

```
YES
```

#2396、字符串弱等于

设计一个函数bool equal(string a,string b)

对于传入的字符串 a, b

- 若 a, b 相等,返回 $true$
- 若 a, b 长度不相等,返回 $false$
- 若 a 长度为奇数 或 b 长度为奇数 ,返回 $false$
- 否则

  将 a 分为 $a1, a2$,b 分为 $b1, b2$ 四个部分

  如果 equal(a1,b1) && equal(a2,b2) 为 $true$ 返回 $true$

  如果 equal(a1,b2) && equal(a2,b1) 为 $true$ 返回 $true$

#1744、谢尔宾斯基三角形

题目描述

定义谢尔宾斯基三角形如下:

- 只有 1 层的分形三角是由 `_` , `\` , `/` 构成一个单独的三角形
- n 层分形三角由三个 $n-1$ 层分形三角构成,它们以品字形排列

给定一个正整数 n ,请输出一个层数为 n 的分形三角

输入格式

单个正整数,表示 n

输出格式

若干行,表示一个 n 层分形三角图案

输入样例2

```
1
```

输出样例2

```
 /\
/__\
```

数据范围

对于全部数据 $1 \leq n \leq 8$

输入样例1

```
3
```

输出样例1

```
      /\
     /__\
    /\  /\
   /__\/__\
  /\      /\
 /__\    /__\
/\  /\  /\  /\
/__\/__\/__\/__\
```

#1744、谢尔宾斯基三角形

输出为从上至下从左往右输出,不好控制画笔位置,考虑使用二维数组模拟画布,最后将整个二维数组输出即可

观察发现第 n 阶图形有 2^n 行 2^{n+1} 列,第 n 阶图形最上方 / 的位置为$(1,2^n)$

若现在需要在 (x,y) 位置绘制一个 1 阶图形

- 在 (x,y) 位置和 $(x,y+1)$ 位置绘制一个 /
- 在 $(x+1,y-1)$和 $(x+1,y+2)$ 位置位置绘制一个 \
- 在 $(x+1,y)$和 $(x+1,y+1)$ 位置位置绘制一个 _

若现在需要在 (x,y) 位置绘制一个 n 阶图形,其中 $n>1$

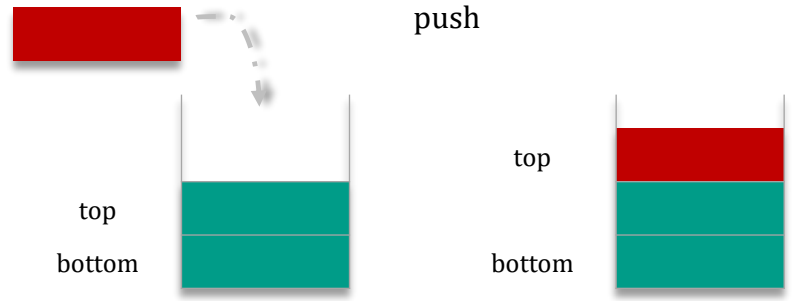
- 在 (x,y) 位置绘制一个 $n-1$ 阶图形
- 在 $(x+2^{n-1},y-2^{n-1})$ 位置绘制一个 $n-1$ 阶图形
- 在 $(x+2^{n-1},y+2^{n-1})$ 位置绘制一个 $n-1$ 阶图形

栈

栈（$stack$）,是一种操作受限制的线性的数据结构

其限制是只允许在栈的一端进行插入和删除运算

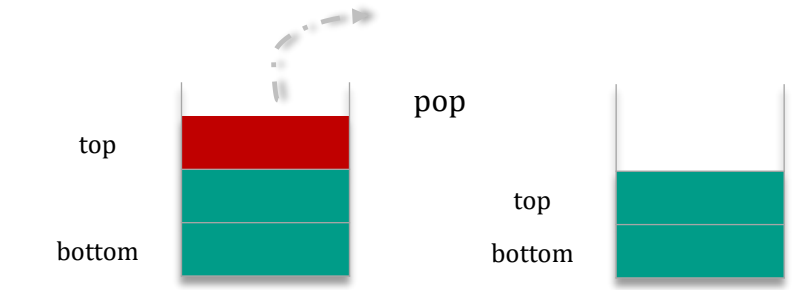
栈的修改按照后进先出的原则,栈被称做后进先出($last\ in\ first\ out$)表,简称 $LIFO$ 表



可以想象往子弹夹中装子弹的情形

正常情况下只能往子弹夹入口那段端入子弹,好比向栈中压入元素,称为$push$

射击的时,弹夹会从顶端弹出子弹,好比从栈顶弹出元素称为pop



栈的实现


```
int s[100005], pos;
void push(int val)
{
  s[++pos] = val;
}
int pop()
{
  if (pos <= 0)
    return -1;
  return s[pos--];
}
```

C++ 中使用STL构造一个stack的语句为： stack<T> s

更多用法可以翻阅文档: http://cplusplus.com/reference/stack/stack/?kw=stack

方法	功能
push	往栈中压入一个元素
pop	从栈中弹出一个元素
top	获取栈顶元素
size	获取栈内元素个数

#2273、栈计算器

题目描述

本题要求你为初学数据结构的小伙伴设计一款简单的利用堆栈执行的计算器。计算器由两个堆栈组成，一个堆栈 S_1 存放数字，另一个堆栈 S_2 存放运算符。计算器的最下方有一个等号键，每次按下这个键，计算器就执行以下操作：

- 从 S_1 中弹出两个数字，顺序为 n_1 和 n_2
- 从 S_2 中弹出一个运算符 `op`；
- 执行计算 $n_2\ op\ n_1$
- 将得到的结果压回 S_1

直到两个堆栈都为空时，计算结束，最后的结果将显示在屏幕上。

输入格式:

输入首先在第一行给出正整数 N（$1 \le N \le 10^3$），为 S_1 中数字的个数。

第二行给出 N 个绝对值不超过 100 的整数；第三行给出 $N-1$ 个运算符

> *这里仅考虑 ＋、－、＊、／这四种运算。一行中的数字和符号都以空格分隔。*

输出格式:

将输入的数字和运算符按给定顺序分别压入堆栈 $S1$ 和 S_2，将执行计算的最后结果输出。注意所有的计算都只取结果的整数部分。题目保证计算的中间和最后结果的绝对值都不超过 10^9。

如果执行除法时出现分母为零的非法操作，则在一行中输出：`ERROR: X/0`，其中 X 是当时的分子。然后结束程序。

输入样例 1:

```
5
40 5 8 3 2
/ * - +
```

输出样例 1:

```
2
```

输入样例 2:

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

输出样例 2:

```
ERROR: 5/0
```

#436、合法的出栈序列

描述

对于入栈顺序 1，2，3，4 序列，4，3，2，1 是合法的出栈序列，而 3，4，1，2 就不是合法的出栈序列。现在就请你编写一个程序，判断出栈序列是否合法。

输入

第一行 2 个整数 $n(1 \leq n \leq 20)$ 和 m，1，2，3.....n 是入栈顺序，接下来 m 行，每行有 n 个数，是需要判断的出栈序列。

输出

m 行，如果是合法的出栈序列，则输出 `YES`，否则输出 `NO`。

输入样例

```
4 2
3 4 2 1
3 4 1 2
```

输出样例

```
YES
NO
```

#436、合法的出栈序列

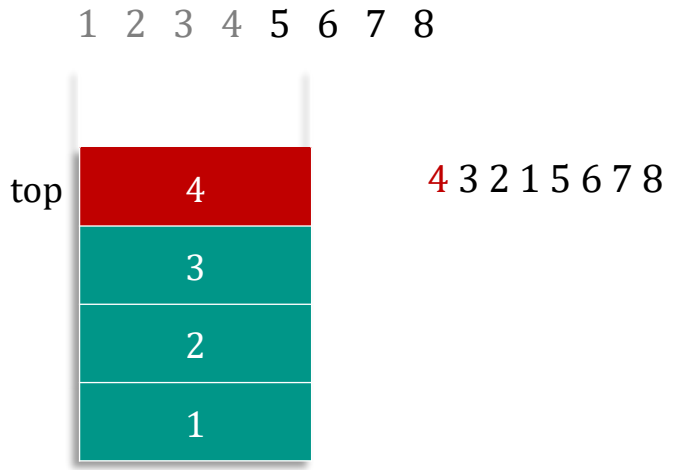
思路1

模拟数字1 ~ n入栈对于出栈序列s,每次判断第一个数值是否为栈顶

如果是那么从s序列中删去,并且弹出栈顶元素

最后栈若为空,说明序列为合法

每个元素仅会入/出栈一次,时间复杂度$O(n)$



思路2

根据栈$FILO$性质,本题栈内元素自底向上单调递增

对于元素x,当其入栈时,小于x元素要么已经出栈,要么还在栈中

检查 $s[i]$ 后 所有小于 $s[i]$ 的元素是否递减即可

#453、括号匹配(基础)

描述

输入一个字符串(长度不超过 10^5)，判断这是否是一个合法的括号序列

输入格式

> 一行，表示一个待判断的序列。

输出格式

> 如果合法输出 Yes ，否则输出 No

输入样例1

```
(())()
```

输出样例1

```
Yes
```

输入样例2

```
()()(()
```

输出样例2

```
No
```

维护一个栈,对于 $i = 1,2,\dots,|s|$依次考虑:

若s_i是右括号且栈非空且栈顶元素是 s_i 对应的左括号,就弹出栈顶元素

若不满足上述条件,则将 s_i压入栈中

在遍历整个 s 后,若栈是空的,那么 s 就是合法括号序列,否则就不是

时间复杂度 $O(|s|)$

#1765、网页浏览历史

题目描述

现代浏览器都有三个基本功能:

- **打开页面**

  > *在地址栏中输入网址，并跳转到网址对应的页面;*

- **回退**

  > *返回到上一次访问的页面;*

- **前进**

  > *返回到上次回退前的页面，如果上一次操作是打开页面，那么将无法前进;*

现在，Mas 打开浏览器，进行了一系列操作，你需要输出他每次操作后所在页面的网址。

输入格式

第一行输入一个整数 $n(1 \leq n \leq 50000)$，表示操作次数。

接下来一共 n 行，每行首先输入一个字符串

- 如果是 `GOTO`，后面接着输入一个不含有空格和换行的网址（网址长度小于 `256`），表示浏览器地址栏中输入的网址;
- 如果是 `BACK`，表示点击了回退按钮;
- 如果是 `FORWARD`，表示点击了前进按钮。

输出格式

对于每次操作，如果 Mas 能操作成功，输出操作之后的网址，否则输出 `Invalid`。保证输入的所有网址都是合法的。

样例输入

```
10
GOTO https://oj.shiyancang.cn/Problem/1765.html
GOTO http://www.hourofcode.cn
BACK
BACK
FORWARD
FORWARD
BACK
GOTO http://noi.ac/problem/31
FORWARD
BACK
```

样例输出

```
https://oj.shiyancang.cn/Problem/1765.html
http://www.hourofcode.cn
https://oj.shiyancang.cn/Problem/1765.html
Invalid
http://www.hourofcode.cn
Invalid
https://oj.shiyancang.cn/Problem/1765.html
http://noi.ac/problem/31
Invalid
https://oj.shiyancang.cn/Problem/1765.html
```

#1765、网页浏览历史

历史记录可以用栈来维护,使用数组实现栈

top 记录栈顶下标,cur记录当前位置

如果是 $GOTO$ 操作,将url入栈(令 $pos = cur + 1$,覆盖 cur 到原 top 之间的历史记录)

如果是 $BACK$ 操作,输出$cur - 1$位置的url, 并移动 cur

如果时 $FORWARD$ 操作,输出 $cur + 1$ 位置的 url , 并移动 cur

#1636、栈和最小值

题面描述

实现一个栈, 支持以下操作:

> $push\ x$ *将 x 压入栈*
> pop *将栈顶元素弹出,若栈内没有元素忽略该操作*
> min *如果栈为空返回 -1 ,否则返回栈中元素的最小值*

输入格式

第一行一个正整数 n

接下来 n 行命令

输出格式

对于每一个 min 操作，输出当前栈内的最小值

数据规模

对于 40% 的数据, $1 \leq n \leq 1000$
对于 50% 的数据, $1 \leq n \leq 200000, x \leq 10000$
对于 100% 的数据, $1 \leq n \leq 200000$
对于全部数据 $-1000000 \leq x \leq 1000000$

输入样例:

```
6
push 1
min
push 2
min
push 3
min
```

输出样例:

```
1
1
1
```

使用两个栈 s_1, s_2

s_1 维护 x

s_2 维护栈内最小值

对于每一个$push\ x$操作,s_1直接压入x

s_2 压入 $\min(s2[top], x)$

时间复杂度 $O(n)$

前中后缀表达式

中缀表达式就是常见的运算表达式

$$(3+4)\times 5-6$$

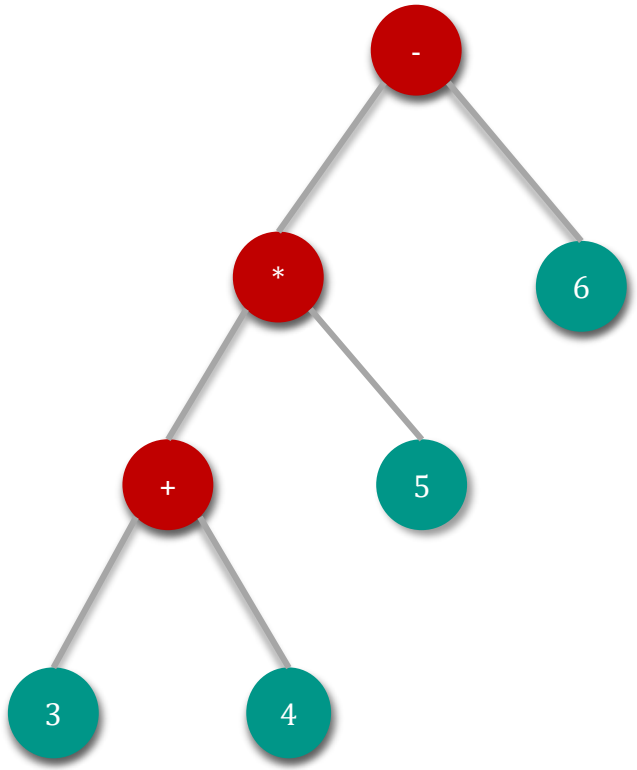
后缀表达式又称逆波兰表达式,运算符位于操作数之后

$$3\ 4\ +\ 5\ \times\ 6\ -$$

前缀表达式又称波兰式,前缀表达式的运算符位于操作数之前

$$-\ \times\ +\ 3\ 4\ 5\ 6$$

后缀表达式和前缀表达式都不带括号

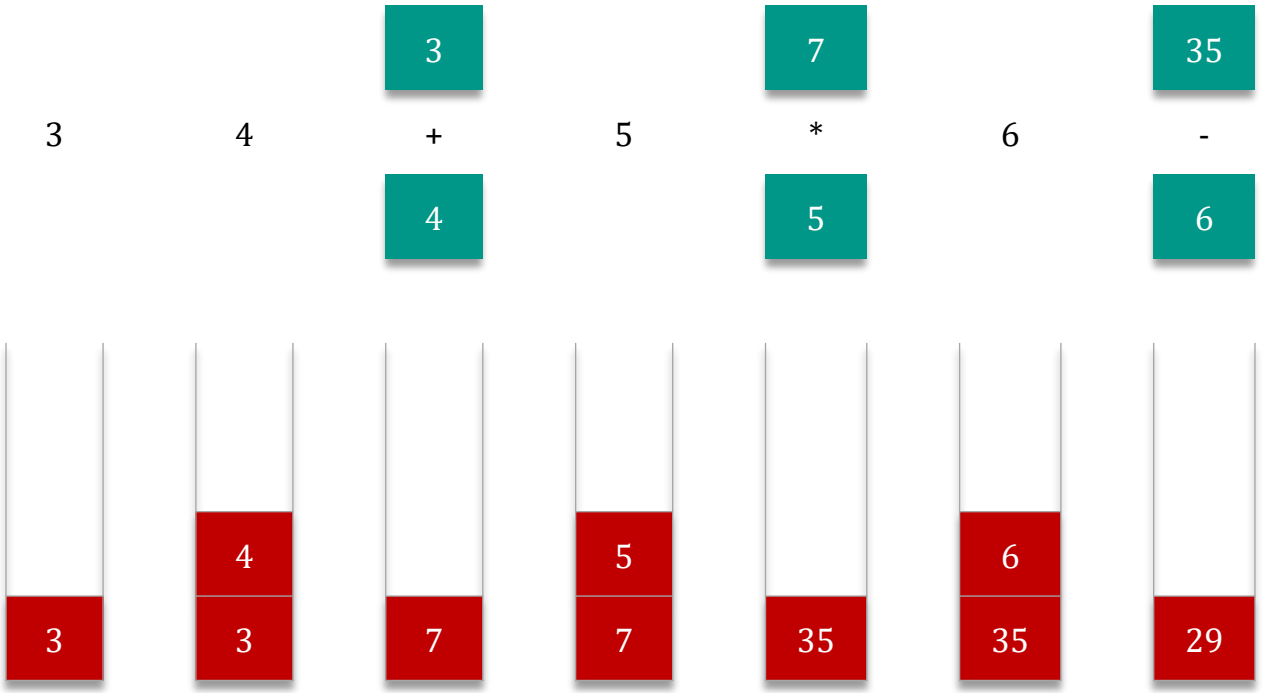


后缀表达式的计算

从左往右扫描表达式

遇到操作数时,将操作数压入栈中

遇到操作符时,弹出两个操作数进行运算（次顶元素 op 栈顶元素）,将结果压入栈中

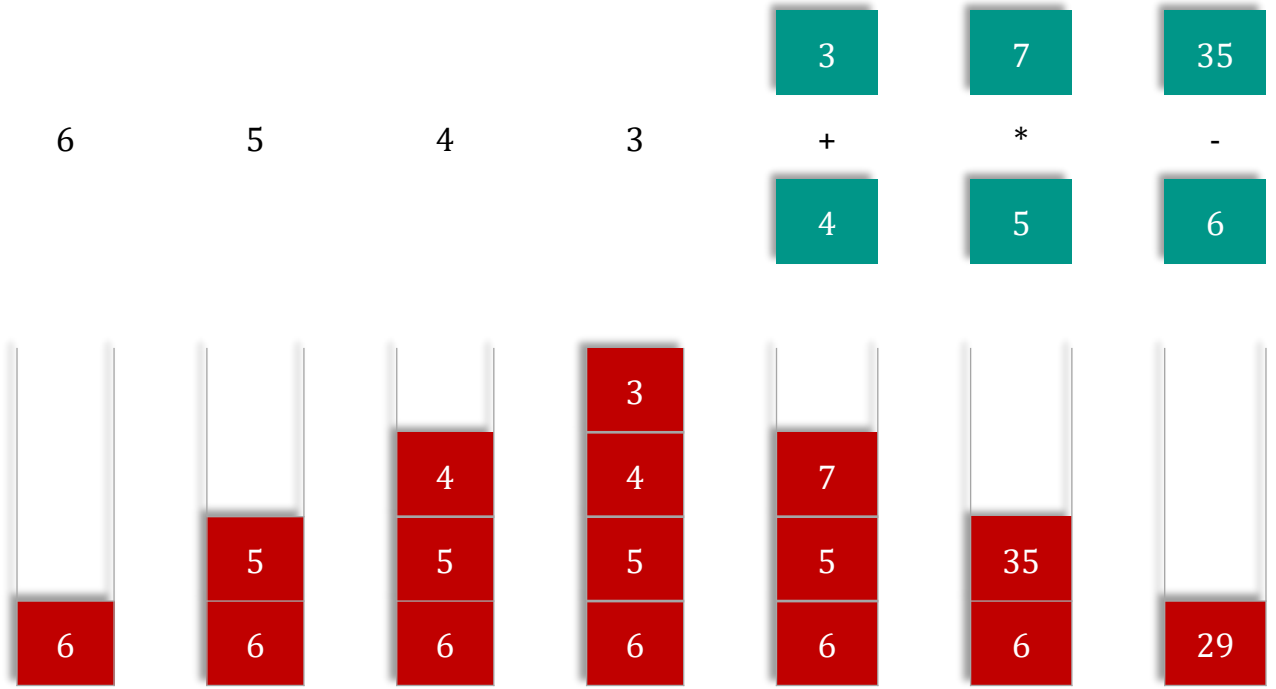


前缀表达式的计算

从右往左扫描表达式

遇到操作数时,将操作数压入栈中

遇到操作符时,弹出两个操作数进行运算（栈顶元素 *op* 次顶元素）,将结果压入栈中



#1117、后缀表达式求值

题目描述

从键盘读入一个后缀表达式（字符串），只含有 $0 - 9$ 组成的运算数（全部为整数）及加（ `+` ）、减（ `-` ）、乘（ `*` ）、除（ `/` ）四种运算符。每个运算数之间用一个空格隔开，不需要判断给你的表达式是否合法。以@作为结束标志。

输入

一行，一个后缀表达式，长度小于 255 ，以 `@` 作为结束标志。

输出

表达式的值。

样例输入

```
16 9 4 3 +*-@
```

样例输出

```
-47
```

```
while ((c = getchar()) != '@')
{
  if (isdigit(c))
    num = num * 10 + c - 48;
  else if (c == ' ')
  {
    s.push(num);
    num = 0;
  }
  else
  {
    int a = s.top();
    s.pop();
    int b = s.top();
    s.pop();
    if (c == '+') s.push(a + b);
    else if (c == '-') s.push(b - a);
    else if (c == '*') s.push(b * a);
    else if (c == '/') s.push(b / a);
  }
}
cout << s.top();
```

中缀表达式转后缀表达式

从左往右遍历表达式

遇到操作数

加入输出序列

遇左括号

压入栈中

遇右括号

不断的弹栈,将弹出的操作符弹出并加入输出序列,直到遇到左括号

遇到操作符

若栈为空或运算符优先级高于栈顶元素,压入栈中

否则,弹栈并将元素加入输出序列,直到遇到一个优先级小于它的栈顶元素

中缀表达式转前缀表达式

从右往左遍历表达式

遇到操作数

加入输出序列

遇右括号

压入栈中

$(1+2)*(3+4)+5-6$

遇左括号

不断的弹栈,将弹出的操作符弹出并且输出,直到遇到右括号

遇到操作符

如果 栈为空 或 运算符优先级高于或等于栈顶元素,压入栈中

否则,弹栈并将元素加入输出序列,直到遇到一个优先级小于它的栈顶元素

输出序列翻转即为前缀表达式

#1728、中缀式转换

题目描述

我们熟悉的表达式如 `a+b` 、 `a+b*(c+d)` 等都属于中缀表达式。

中缀表达式就是（对于双目运算符来说）操作符在两个操作数中间： `num1 operand num2` 。

同理，后缀表达式就是操作符在两个操作数之后： `num1 num2 operand` 。

前缀表达式则是操作符在两个操作数之前： `operand num1 num2` 。

现在试图输入一个中缀表达式分别转换为后缀表达式和前缀表达式。现在请你设计一个程序完成题目要求。

为简化问题，操作数均为个位数，操作符只有 `+-*/()`

输入

输入一个长度不超过 1000 的字符串，表示这个表达式。这个表达式里只包含 `+-*/()` 这几种符号。其中小括号可以嵌套使用。数据保证输入的操作数中不会出现负数。并且输入数据不会出现不匹配现象。

输出

输出转换的后缀表达式和前缀表达式。

谢谢观看