4.栈与队列

（c1）栈应用：进制转换

Hickory,Dickory,Dock

The mouse ran up the clock

\- Nursery Rhyme Medley

邓俊辉

deng@tsinghua.edu.cn

典型应用场合

逆序输出	• conversion • 输出次序与处理过程颠倒；递归深度和输出长度不易预知
递归嵌套	• stack permutation + parenthesis • 具有自相似性的问题可递归描述，但分支位置和嵌套深度不固定
延迟缓冲	• evaluation • 线性扫描算法模式中，在预读足够长之后，方能确定可处理的前缀
栈式计算	• RPN • 基于栈结构的特定计算模式

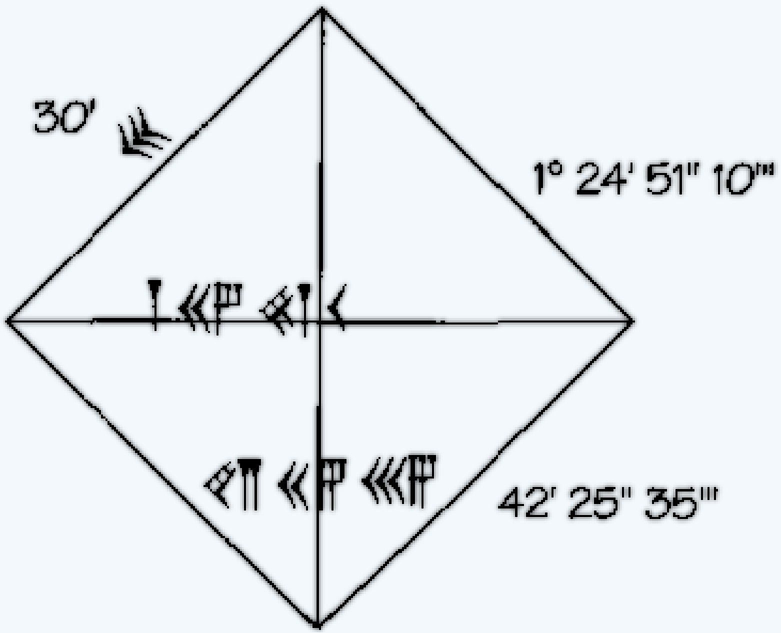
进制转换

❖ 描述：给定任一10进制非负整数，将其转换为λ进制表示形式

$12345_{(10)} = 30071_{(8)}$

```
printf("%d | %I64d | %b | %o | %x",n);
```

❖ 巴比伦楔形文字（Babylonic cuneiform）中的60进制...



❖ 正方形的对角线

1°24'51''10'''

= 1 + 24/60 + 51/60^2 + 10/60^3

= 1.41421296296296296...

❖ 误差

|1°24'51''10''' - sqrt(2)| < 0.000,000,6 = 0.6×10^{-6}

即便边长为1km，误差亦不足1mm

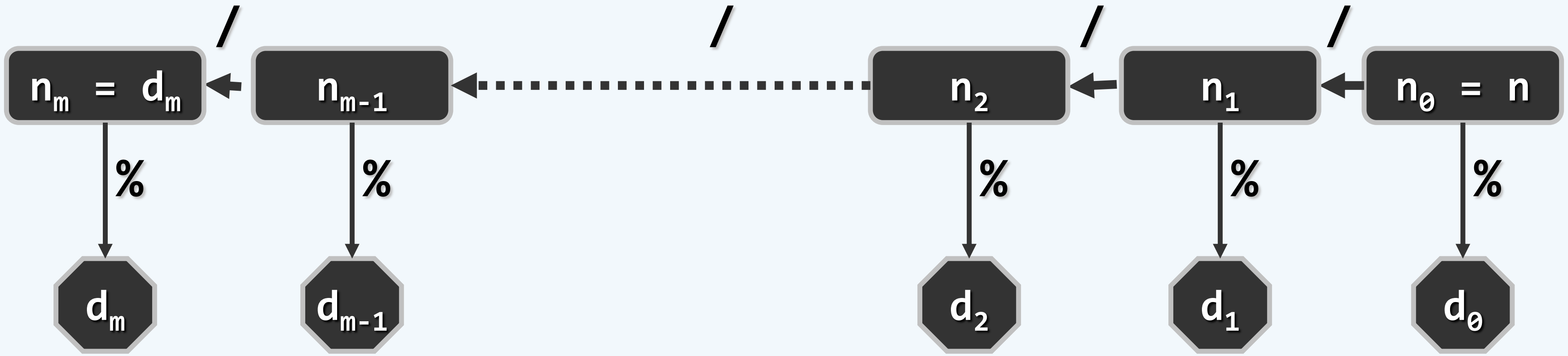
思路

- 设： $n = (d_m \ldots d_2\ d_1\ d_0)_\lambda = d_m \times \lambda^m + \ldots + d_1 \times \lambda^1 + d_0 \times \lambda^0$

  令： $n_i = (d_m \ldots d_{i+1}\ d_i)_\lambda$

- 则有：$n_{i+1} = n_i / \lambda$ 和 $d_i = n_i \% \lambda$ //初始取$n_0 = n$

- 构思：n对λ反复取模、整除，即可**自低到高**得出λ进制的各位



难点及解决方法

- 位数m并不确定，如何正确记录并输出转换结果？具体地

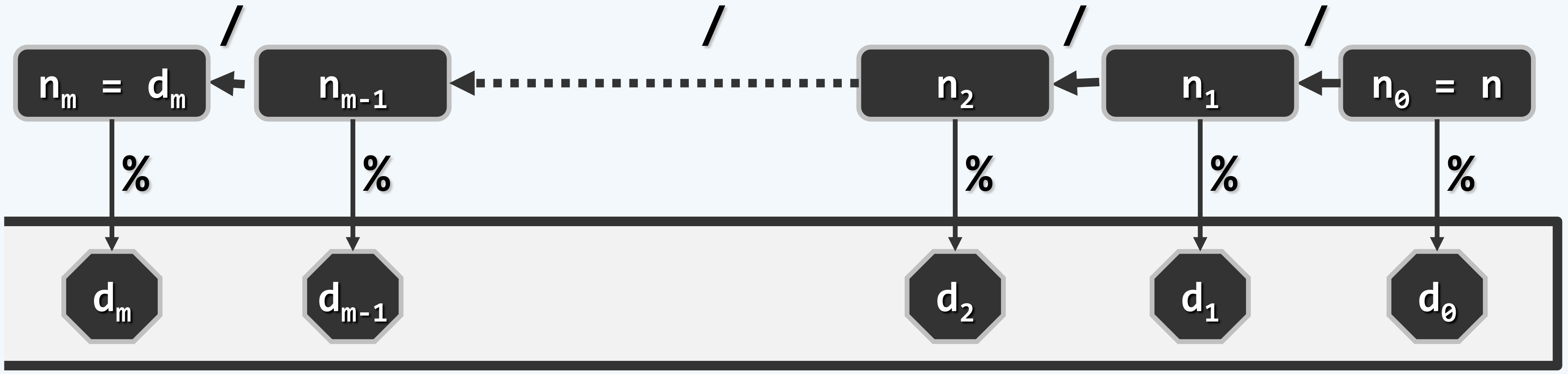
  如何支持足够大的m，同时空间也不浪费？

  自低到高得到的数位，如何**自高到低**输出？

- 若使用向量，则扩充策略必须得当

  若使用列表，则多数接口均被闲置

- 使用栈，即可满足以上要求，亦可有效控制计算成本



算法实现

```
❖ void convert( Stack<char> & S, __int64 n, int base ) {
     static char digit[] = //新进制下的数位符号，可视base取值范围适当扩充
        { '0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F' };
     while (n > 0) { //由低到高，逐一计算出新进制下的各数位
        S.push( digit[n % base] ); //余数（对应的数位）入栈
        n /= base; //n更新为其对base的除商
     }
  }
❖ main() {
    Stack<char> S; convert(S, n, base); //用栈记录转换得到的各数位
    while ( !S.empty() ) printf( "%c", S.pop() ); //逆序输出
  }
```