西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY

C程序设计
Programming in C

1011014

主讲：姜学锋，计算机学院

用空间换取时间编程

1、空间换取时间

2、用数学方法解决问题

- 1．空间换时间
- 在计算机问题求解的算法设计中，考虑算法性能优化时需要注意80-20原则：即20%的程序消耗了80%的运行时间，因而要改进效率，最主要是改进那20%的代码，而不要优化程序中开销不大的那80%。

6.5 数组应用程序举例

- 计算机程序中最主要的矛盾是空间和时间的矛盾，那么，从80-20原则出发逆向思维考虑程序的效率问题，就有了解决问题“以空间换时间”。

- 算法的执行总是需要计算机的时间和空间，由于现在计算机的内存趋向于大容量，空间复杂性相对于时间复杂性来说不那么重要。因而，可以消耗空间来换取时间。

【例6.16】

一个只能被素数2、3、5、7整除的数称为Humble Numbers（简称丑数），数列{1,2,3,4,5,6,7,8,9,10,12,14,15,16,18......}是前15个丑数（把1也算作丑数）。编程输入n（$1 \le n \le 5842$），输出这个数列的第n项。

例题分析

可以用枚举法来求解。

枚举一个自然数H，逐一检查H是否只能被2、3、5、7整除，方法是去掉H所有的2、3、5、7因子，如果结果为1，则H是丑数；例如16/2/2/2/2=1，18/2/3/3=1结果为1，所以16、18是丑数，而22/2=11结果不为1，所以22不是丑数，其余依此类推。

6.5 数组应用程序举例

例6.16

```c
#include <stdio.h>
int main()
{
  int cnt=0, n , H=0, t;
  scanf("%d",&n);
  while (cnt<n) { //数列第n项时结束
    H++; //下一个自然数
    t = H;
    while (t%2 ==0 ) t=t/2; //去除所有2的因子
    while (t%3 ==0 ) t=t/3; //去除所有3的因子
    while (t%5 ==0 ) t=t/5; //去除所有5的因子
    while (t%7 ==0 ) t=t/7; //去除所有7的因子
    if (t==1) cnt++; //得到新 Humble Numbers
  }
  printf("%d\n",H); //第n项 Humble Numbers
```

6.5 数组应用程序举例

例6.16

```
    return 0;
}
```

在CodeBlocks中运行情况如下：

```
5842↙
2000000000
Process returned 0 (0x0)   execution time : 189.156 s
```

上面程序可以将结果求解出来，但是当n是5842时，程序运行时间会很长，因为5842时H已经枚举到了20亿。

例题分析

下面换一种思路来求解。

根据丑数的定义，一个数与｛2,3,5,7｝的积一定也是一个丑数。例如｛1×2，1×3，1×5，1×7｝，｛2×2，2×3，2×5，2×7｝，｛3×2，3×3，3×5，3×7｝，……，均为丑数。

由于丑数是自然数顺序且是唯一的，需要对乘积进行优选，例如2×3和3×2结果都是6，需要排除一个；3×2结果大于1×5，所以应先选1×5。

- 因此，假设一个集合A用来存储丑数，最开始的元素为｛1｝，按下面的方法将得到的最小丑数插入到A中作为新的丑数：

```
A(n)=min(A(i)×2,A(j)×3,A(k)×5,A(m)×7)
n>i,j,k,m，且i,j,k,m只有在本项被选中才向后移动。
```

- 例如：

```
开始时 A为｛1｝i=1,j=1,k=1,m=1
min(1×2,1×3,1×5,1×7)为2插入到A为｛1,2｝i=2,j=1,k=1,m=1
min(2×2,1×3,1×5,1×7)为3插入到A为｛1,2,3｝i=2,j=2,k=1,m=1
min(2×2,2×3,1×5,1×7)为4插入到A为｛1,2,3,4｝i=3,j=2,k=1,m=1
min(3×2,2×3,1×5,1×7)为5插入到A为｛1,2,3,4,5｝ i=3,j=2,k=2,m=1
min(3×2,2×3,2×5,1×7)为6插入到A为｛1,2,3,4,5,6｝i=4,j=2,k=2,m=1
min(4×2,3×3,2×5,1×7)为7插入到A为｛1,2,3,4,5,6,7｝i=4,j=3,k=2,
m=2
依次类推。
```

例6.16

```c
#include <stdio.h>
int min(int a, int b, int c, int d) //求四个数的最小值
{
  a = a > b ? b : a;
  c = c > d ? d : c;
  return a > c ? c : a;
}
int main()
{
  int i=1, j=1, k=1, m=1, n=1;
  int A[6000];
  A[1] = 1; //数列第1项
  for (n=2 ; n<=5842 ; n++) {
    int t1,t2,t3,t4; //计算{2,3,5,7}的乘积
    t1=A[i]*2, t2=A[j]*3, t3=A[k]*5, t4=A[m]*7;
```

6.5 数组应用程序举例

例6.16

```
        A[n] = min(t1,t2,t3,t4); //取最小Humble Number
        if ( A[n]==t1 ) i++; //移动i
        if ( A[n]==t2 ) j++; //移动j
        if ( A[n]==t3 ) k++; //移动k
        if ( A[n]==t4 ) m++; //移动m
    }
    scanf("%d",&n);
    printf("%d\n",A[n]); //输出第n项Humble Number
    return 0;
}
```

- 在CodeBlocks中运行情况如下：

```
5842↙
2000000000
Process returned 0 (0x0)   execution time : 3.281 s
```

- 运行速度显著提高。实际上，这个程序运用了动态规划算法。

- 2．用数学方法解决问题
- 数学是计算机算法的基础，没有数学的依据和基础，就没有算法设计的发展。所以在编写程序的时候，可以应用一些数学方法，会对程序的执行效率有数量级的提高。

【数组应用程序举例】

已知n个人（以编号1，2，3...n分别表示）围坐在一张圆桌周围。从编号为k的人开始报数，数到m的那个人出列；他的下一个人又从1开始报数，数到m的那个人又出列；依此重复下去，直到圆桌周围的人全部出列，问最后出列的人是谁（即编号是多少）？

例题分析

约瑟夫环（Josephus）问题。

可以用模拟算法求解此问题：即通过模拟问题的实际过程，从中得到结果。为此，需要定义一个包含n个元素的数组，初始值均设置为1，表示在圆桌周围的队列中。当某人出列时，设置为0，表示已出列。程序从k个元素开始计数，若数组元素值为0（该人已出列）则跳过计数，直到数到m时，则元素的下标（加1后）即是出列人的编号。继续重复上述过程，直到第n个人出列，得到结果。

例6.58

```
#include <stdio.h>
#define N 1000 //定义足够大的数组
int main()
{
  int A[N], i, s, count; //A[i]=1表示在圈内，=0表示在圈外
  int n, k, m; //n个人，从k开始，数到m
  scanf("%d%d%d",&n,&k,&m);
  if (n<N && k>=1 && k<=n && m>0) { //检验n,k,m输入是否正确
    for (i=0; i<n; i++)
      A[i]=1; //首先设置所有人都在圈内，
    i=k-1; //转换为下标（因为下标从0开始）
    s=1; //累计出圈的人数，设初值为1，那最后一个就不用出圈了
    count=0; //报数计数器，数到m就归零
    while(s<n) {
      count+= A[i]; //若a[i]=0，直接跳过
```

6.5 数组应用程序举例

例6.58

```c
        if (count==m) { //报数为m的那个人出圈
          A[i]=0; //设此位置为0，表示此人已经出圈
          ++s; //出圈人数增加一个
          count=0; //报数计数器归零
        }
        i=(i+1)%n; //若有超过1个人在圈里，就不断遍历数组
      }
      for (i=0; i<n; i++) { //遍历数组，看看还有谁没有出圈
        if (A[i]==1) { //找的就是他
          printf("last=%d\n",i+1); //编号等于下标加1
          break;
        }
      }
    }
    return 0;
```

6.5 数组应用程序举例

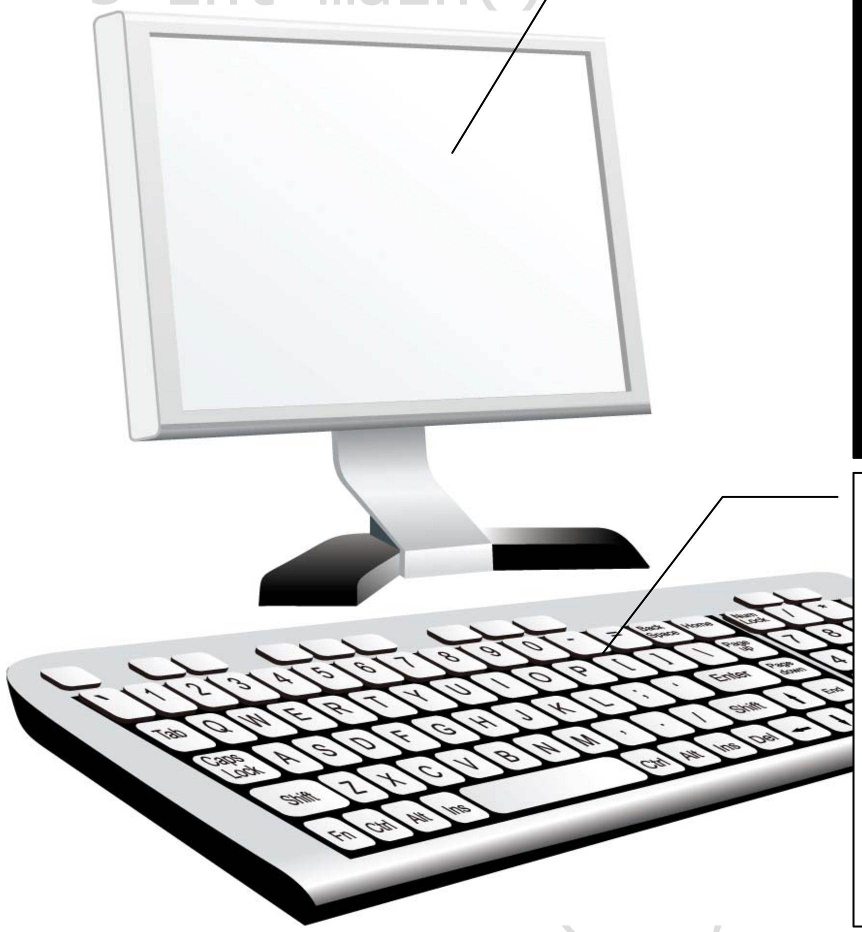
例6.58

```
}
```

6.5 数组应用程序举例

例6.58

```
#include <stdio.h>
#define N 1000 //定
int main()
```



程序运行屏幕

```
last=26
```

```
100 1 10↙
```

例题分析

前面的模拟算法，不仅程序写起来比较麻烦，而且时间复杂度高达O(nm），，当n，m非常大的时候，几乎是没有办法在短时间内出结果的。考虑到原问题仅仅是要求出最后出列的编号，而不是要模拟整个过程。因此如果要追求效率，就要打破常规，实施一点数学策略。

例题分析

约瑟夫环问题的数学方法。

为了方便讨论，先把原问题改变一下，并不影响原意：

问题描述：n个人（编号0~(n-1)），从0开始报数，报到m-1的退出，剩下的人继续从0开始报数。求最后出列人的编号。

例题分析

第1个人（编号一定是（m-1）%n) 出列之后，剩下的n-1个人组成了一个新的约瑟夫环（以编号为k=m%n的人开始）：

k，k+1，k+2，…，n-2，n-1，0，1，2，…，k-2

并且从k开始报0。

例题分析

现在把编号做一下转换：

k --> 0

k+1 --> 1

k+2 --> 2

…

k-3 --> n-3

k-2 --> n-2

例题分析

序列1：0,1,2,3 … n-2,n-1

序列2：0,1,2,3 … k-2,k，…，n-2,n-1

序列3：k,k+1,k+2,k+3，…，n-2,n-1,1,2,3，…，k-2,

序列4：0,1,2,3 …，5,6,7,8，…，n-3,n-2

变换后就成为了（n-1）个人报数的子问题，假如知道这个子问题的解：如x是最终的出列人，那么根据上面这个表把x变回去正好就是n个人时的解。

例题分析

变回去的公式很简单：

∵ k=m%n;

∴ x' = x+k = x+ m%n ; 而 x+ m%n 可能大于n

∴ x'= (x+ m%n)%n = (x+m)%n

得到 x'=(x+m)%n

例题分析

显然，欲求n-1个人报数问题的解，就要n-2个人的解。n-2个人的解要先求n-3的情况，于是有下面的递推公式。令f表示第i个人报m最后出列的编号，则最后的结果自然是f[n]。

递推公式：

f[1]=0;

f[i]=(f[i-1]+m)%i; (i>1）

有了这个公式，从1～n顺序算出f的值，最后结果必然是f[n]。因为编号是从1开始，因此输出f[n]+1。

6.5 数组应用程序举例

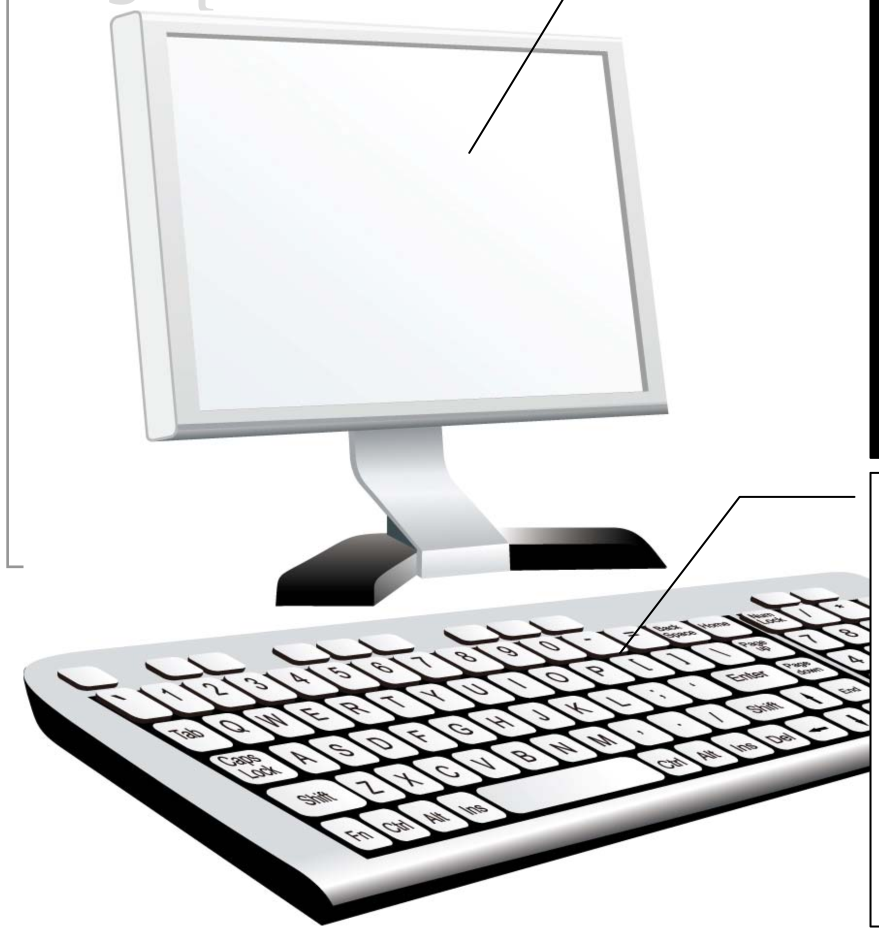
例6.59

```c
#include <stdio.h>
int main(void)
{
  int n, k, m, i; //n个人，从k开始，数到m
  scanf("%d%d%d",&n,&k,&m);
  k=k-1; //转换为下标（因为下标从0开始）
  for (i=2; i<=n; i++) k=(k+m)%i;
  printf("last=%d\n",k+1); //编号等于下标加1
  return 0 ;
}
```

6.5 数组应用程序举例

例6.59

```
#include <stdio.h>
int main(void)
{
```



程序运行屏幕

```
last=26
```

```
100 1 10↙
```

CP程序设计