C语言综合应用教程

刘华蓥 衣治安 主编

科学出版社

C语言综合应用教程

主　编　刘华鉴　衣治安
副主编　吴雅娟　时贵英
主　审　马瑞民

科学出版社
北京

内 容 简 介

本书是作者总结多年教学经验，根据全国计算机基础教学的改革方向并结合各专业学生在以后的学习和工作中的共性需求，精心组织编写的。本书介绍了计算机求解问题的过程和编程思想、常用的程序设计方法和程序调试方法，并通过学生成绩管理系统和几个常用的数值计算方法的实现来培养学生综合应用C语言知识解决实际问题的能力，最后讲解如何在VC环境中实现对数据库的操作。通过对本书的学习，将使学生具备程序设计思想，更好地掌握程序设计方法，并具备用C语言解决专业实际问题的能力。

本书适合作为高等学校C语言后续训练课程的教材，也适合作为广大C语言编程爱好者的自学读物。

图书在版编目(CIP)数据

C语言综合应用教程/刘华蓥，衣治安主编．—北京：科学出版社，2013

ISBN 978-7-03-038465-2

Ⅰ.①C… Ⅱ.①刘…②衣… Ⅲ.①C语言-程序设计-教材 Ⅳ.①TP312

中国版本图书馆CIP数据核字(2013)第202059号

责任编辑：胡云志 李淑丽 / 责任校对：胡小洁

责任印制：肖 兴 / 封面设计：华路天然工作室

科学出版社出版

北京东黄城根北街16号

邮政编码：100717

http://www.sciencep.com

北京科印技术咨询服务公司 印刷

科学出版社发行 各地新华书店经销

*

2013年8月第 一 版 开本：720×1000 B5

2013年8月第一次印刷 印张：10 1/4

字数：206 000

POD售价：35.00元

NO. 00000001 D2014

前　言

本书是我校开设的“计算机综合应用（C语言）”课程的教材，其先修课是“大学计算机基础”和“C程序设计”。C语言由于其功能丰富、应用灵活等优点，广受欢迎，是许多高校程序设计课程的首选语言之一，但也正是它的众多优点，使其学习很有难度。通过计算机综合应用这门课程的学习，将使学生具备程序设计思想，更好地掌握程序设计方法，并具备用C语言解决专业实际问题的能力。

本书编写组成员多年来一直从事计算机基础教学，常年担任高级语言类课程的主讲和实验任务，主持了二十多项省部级计算机基础教学改革课题的研究，出版了高等学校教材十余部。以本书编写组为骨干的计算机基础教学团队是省级教学团队，该团队中有1人获得省级教学名师奖，2人获得校级教学名师称号，建设有“C程序设计”等2门省级精品课，获得国家级教学成果奖1项，省部级教学成果奖13项。近几年来，该教学团队一直关注计算机综合应用课程的教学改革。根据全国计算机基础教学的改革方向和我校各专业学生在以后的学习和工作中的共性需求，编写了本书。

本书介绍了计算机求解问题的过程和编程思想、几种常用的程序设计方法和程序调试方法，以进一步培养学生的计算思维能力，并通过学生成绩管理系统和几个常用的数值计算方法的实现来训练学生综合应用C语言知识解决实际问题的能力。目前数据库已成为企业、部门乃至个人日常工作、生产和生活的基础设施，因此，本书最后讲解了如何在VC环境中实现对数据库的操作。

本书C例题均在Visual C++6.0环境中调试运行，大部分例题也在Turbo C 2.0环境中调试运行。

本书由刘华蓥、衣治安主编，吴雅娟和时贵英担任副主编，马瑞民主审。其中第1章由衣治安编写，第2章由吴雅娟编写，第3章由刘华蓥编写，第4章由时贵英编写，全书由刘华蓥统稿。本教材建议授课时间为40学时，其中理论占24学时，实验占16学时。可以根据授课对象和教学需要调整授课学时和教学内容。

本书在编写过程中得到了计算机基础教育系老师们的指导与帮助，他们的教学资料和经验对本书的完善起到了重要作用，在此致以诚挚的谢意。

由于水平有限，难免有不当之处，恳请批评指正。

作　者

2013年6月

目　录

第 1 章　程序设计方法

在计算机普及的信息时代，社会运转的各种沟通交流都建立在高度发展的信息技术平台之上，计算机学科所体现出来的思维方法已经超越了本学科领域，成为从事其他科学研究的重要支撑。本章介绍应用计算机解决问题的计算思维方法和程序设计中常用的算法。

1.1　问 题 求 解

现实社会中，涉及应用计算机解决的问题可以分为两大类，一类是应用单一算法就可以解决的问题，称为算法类问题。计算学科中有许多著名的算法类问题，例如哥尼斯堡七桥问题、梵天塔问题、背包问题、旅行商问题及项目调度优化问题等。另一类是不能由单一算法解决，而必须构建一个系统来解决的问题，称为系统类问题。系统类问题广泛存在于工程、科学、经济等领域，例如，卫星导航系统就是典型的系统类问题。

算法类问题和系统类问题的求解过程、思维和方法是有很大区别的。

(1) 解决算法类问题需要建立数学模型、设计数据结构、设计算法，利用某种程序设计语言编写程序等几个步骤。还要关注算法的正确性、时间复杂性和空间复杂性等。

(2) 系统类问题求解需要建立业务模型、建立软件模型、开发软件模块，将其组装为软件系统并将其部署到实际环境中运行。软件系统需要关注体系结构的合理性，并关注可靠性与安全性等。

在初次学习程序设计语言的时候，多数时间都花费在理解程序设计语言的语法细节上，极少有机会应用所学知识设计一个完整的系统解决某一问题。本章重点介绍算法类问题的求解方法。

计算机算法是一组有穷的规则，它规定了解决某一特定类型问题的一系列运算。算法是对特定问题求解步骤的一种描述，是由若干条指令组成的有穷集合。

历史上最早的算法是欧几里得算法，也称辗转相除法，是描述求两个整数的最大公约数的算法。三国时期的数学家刘徽给出的求圆周率的算法——割圆术，是中国古代算法的代表作。

计算学科中有许多经典的算法策略，例如穷举法、递归法、回溯法和分治法

等，本章重点介绍常用的算法策略，并根据C语言的面向过程程序设计的特点，介绍结构化程序设计中自顶向下的模块化方法与程序调试的简单方法，期望读者对程序设计方法有个整体的了解，并建立问题求解、系统设计以及理解人类行为的一系列思维方式，培养计算思维能力。

1.2 穷 举 法

穷举法的基本思想是对要解答问题的所有可能情况一一进行测试，从中找出所有满足求解条件的答案。穷举的意思就是列出所有可能的情况，找出其中符合要求的解。穷举法简单有效并且容易应用，适合求解可能的答案是有限个并可知的问题。经典的百钱买百鸡问题、密码学中的暴力破解方法、旅行商中逐条线路计算以及国王的婚姻中国王使用的方法都属于穷举法。

使用穷举法解答问题一般需要解决以下3个基本问题：

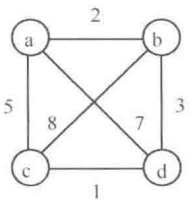
（1）所有可能情况的集合（简称情况集合）是什么。

（2）找出情况集合中元素（或子集）的一种排列方式（简称元素排列）。

（3）按元素排列中的次序对元素（或子集）一一进行测试（简称依次测试），寻找满足测试条件的解。

旅行商问题（Traveling Salesman Problem，TSP）是威廉·哈密尔顿爵士和英国数学家柯克曼于19世纪初提出的一个数学问题，是指旅行家要旅行n个城市然后回到出发城市，任何两个城市之间的距离都是确定的，要求各个城市经历且仅经历一次，并要求所走的路程最短。

【例1-1】如图1-1所示，设有4个城市分别用a，b，c，d表示，线旁的数字表示两个城市间的距离。求从a出发，每个城市只经过一次回到a的最短路径。



序号	路径	路径长度	是否最短
1	a→b→c→d→a	18	否
2	a→b→d→c→a	11	是
3	a→c→b→d→a	23	否
4	a→c→d→b→a	11	是
5	a→d→b→c→a	23	否
6	a→d→c→b→a	18	否

图1-1　旅行商问题

1. 建立数学模型

数学模型就是为了某种目的，根据对研究对象所观察到的现象及其实践经验，用字母、数字及其他数学符号建立起来的等式或不等式以及图表、图像、框图等归结成的一套反映对象某些主要数量关系的数学公式、逻辑准则和具体算法，用来描述客观事物的特征及其内在联系和运动规律。

建立数学模型是求解算法类问题的第一步，它实现的是将实际问题抽象为数学模型。

设城市个数为 n，记为 $C=\{c_1,c_2,\cdots,c_n\}$，任意两个城市 c_i,c_j 之间的距离记为 $d_{c_ic_j}$，问题的解是寻找城市的访问顺序 $S=\{s_1,s_2,\cdots,s_n\}$，其中 $s_i\in C$，使得 $\sum_{i=1}^{n}d_{s_is_{i+1}}$ 达到最小，规定 $d_{n+1}=d_1$。

2. 建立数据结构

数据结构提供了问题求解/算法的数据操纵机制，数据结构是组织、记忆、改变和操作数据的集合，数据的逻辑结构表示了数据之间的关系。

存储结构：在反映数据逻辑关系的原则下，数据在存储器中的存储方式，有顺序存储结构和链式存储结构。

关于数据结构的基本运算包括建立数据结构、清除数据结构、插入数据元素、删除数据元素、更新数据元素、查找数据元素、按序重新排列、判定某个数据结构是否为空，或是否已达到最大允许的容量以及统计数据元素的个数等。

在本例中主要使用数组来表示各类数据，不涉及其他复杂的数据结构。使用二维数组 d 来表示城市间距离关系，d_{ij} 表示第 i 个城市和第 j 个城市之间的距离，见表 1-1，一维数组 sum 表示每条路径的距离之和。4 个城市所有行走路线共有 6 个可能路径选择。

表 1-1　城市间距离

	a	b	c	d
a	0	2	5	7
b	2	0	8	3
c	5	8	0	1
d	7	3	1	0

3. 问题求解中的算法策略

最自然的想法就是列出每一条可供选择的路线，即对给定的城市进行排列组合，计算每条路线的总里程，最后从中挑出一条最短的路线，这种求解方法是一种穷举的思想，也称遍历，是一种基本的问题求解方法。

（4）编写代码，上机计算结果

例 1-1 使用穷举法的 C 程序代码为：

```
# include "stdio.h"
int main()
{int smin,d[4][4]= {{0,2,5,7},{2,0,8,3},{5,8,0,1},{7,3,1,0}},sum[6],i,j;
  char pat[6][10]= {"abcda","abdca","acbda","acdba","adbca","adcba"};
```

```
    sum[0]= d[0][1]+ d[1][2]+ d[2][3]+ d[3][0];
    sum[1]= d[0][1]+ d[1][3]+ d[3][2]+ d[2][0];
    sum[2]= d[0][2]+ d[2][1]+ d[1][3]+ d[3][0];
    sum[3]= d[0][2]+ d[2][3]+ d[3][1]+ d[1][0];
    sum[4]= d[0][3]+ d[3][1]+ d[1][2]+ d[2][0];
    sum[5]= d[0][3]+ d[3][2]+ d[2][1]+ d[1][0];
    for(i= 0;i< 6;i+ + )
      printf("%5d",sum[i]);
    smin= 0;
    for(i= 1;i< 6;i+ + )
      if(sum[smin]> sum[i]) smin= i;
    printf("\nmin= %d,path= %s\n",sum[smin],pat[smin]);
}
```

运行结果为：

```
  18   11   23   11   23   18
min= 11,path= abdca
```

即求出了最短路径为11，经历的城市为a－b－d－c－a。

TSP是最有代表性的优化组合问题之一，在半导体制造、物流运输等行业有着广泛的应用，在大规模的生产活动中，采取最短路径可以有效地降低成本，因此，TSP问题仍是不少学者目前研究的内容。

穷举法对于小规模的TSP题是有效的，但是对于大规模的TSP问题，使用穷举法在时间上是不可行的。对于具有n个城市的TSP问题，可能的解有（n－1）！个，随着城市数目的增长，组合数呈指数级规律急剧增长，就是所谓的“组合爆炸问题”。10个城市的TSP问题有大约180000个可能解，20个城市的TSP问题的可能路线总数为1.216×10^{17}，假设计算机每秒检索1000万条路线，遍历完所有路线需要386年，这在现实中是不可行的。

TSP难于求解。2001年解决了德国15112个城市的TSP问题，使用了美国Rice大学和普林斯顿大学之间互连的、速度为500MHz的Compaq EV6 Alpha处理器组成的110台计算机，所有计算机花费的时间之和为22.6年。

遍历能够获得最优解，然而，由于组合爆炸，对于较大规模的某些问题，无法在可接受的时间内获得最优解，退而求其次，在可接受的时间内获得足够好的可行解。

了解一下可行解和最优解的概念。本例中的“a－b－d－c－a”和“a－c－d－b－a”两条路线的总距离都是11，是问题的最优解，因为没有其他更短的路径。而abcda可以看成是一个可行解，虽然不是“最短”的距离，但在某些情况下已经“足够短”了，对于一些类似TSP的复杂问题，在可接受的时间范围内获得足够好的可行解更具有现实意义。

解决这类问题就需要寻找切实可行的简化算法，例如可以寻找启发式算法、近似算法、概率算法等。下面采用贪心算法来求解 TSP 问题。

1.3　贪　心　法

贪心法的策略就是一定要做当前情况下的最好选择，否则将来可能会后悔，故名“贪心”。每次在选择下一个城市的时候，只考虑当前情况，保证迄今为止经过的路径总距离最短。

【例 1-2】利用贪心法解决 TSP 问题。

TSP 问题贪心算法的流程图如图 1-2 所示。程序代码如下：

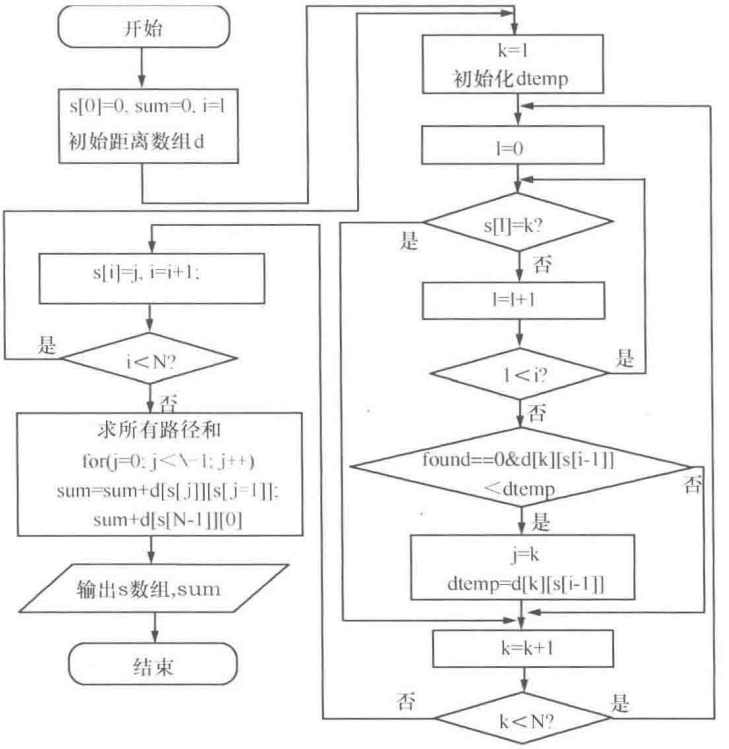


图 1-2　N 个城市的贪心法流程图

```
# include "stdio.h"
# define N 4
int main()
{
  int d[N][N]= {{0,2,5,7},{2,0,8,3},{5,8,0,1},{7,3,1,0}};
  int s[N],k,l,dtemp,found,i,j,sum;
```

```
    s[0]= 0;i= 1;sum= 0;
    do{
        k= 1;

        dtemp= 9999;
        do{
           l= 0;
           found= 0;
           do{
             if (s[l]= = k)
             {
               found= 1;
               break;
             }
             else
               l+ + ;
             }while (l< i);
           if(found= = 0 && d[k][s[i- 1]]< dtemp)
            {j= k;
               dtemp= d[k][s[i- 1]];
            }
            k+ + ;
        }while(k< N);
        s[i]= j;
        i+ + ;
      }while(i< = N);
    for(j= 0;j< N;j+ + )
        printf("%d,",s[j]);
    printf("%d\n",s[0]);
    sum= 0;
    for(j= 0;j< N- 1;j+ + )
    sum= sum+ d[s[j]][s[j+ 1]];
    printf("dis= %d\n",sum+ d[s[N- 1]][0]);
}
```

运行结果为：

```
0,1,3,2,0
dis= 11
```

【说明】0、1、3、2为4个城市a、b、d、c的编号，对应于a—b—d—c—a的顺序。

1.4　迭代和递推

迭代是指重复执行一组操作。每次执行这些操作时，都会有一个或一组变量的值发生变化，而且这种变化一直朝着问题的答案方向迈进。当得到答案时，迭代过程结束。

最常用、最基本的程序设计方法累加和累积都属于迭代，例 1-1 和例 1-2 中都有迭代过程。例如，例 1-2 中求路径和的过程，就是对每一个穷举出来的路径利用迭代法进行累加。在程序中迭代一般用循环结构实现。在求代数方程、微分方程的数值解等计算数学问题时往往使用迭代。

【例 1-3】用迭代法求方程 $x^3-x^2-1=0$ 在 $x=1.5$ 附近的根，其迭代精度为 10^{-5}。

从程序设计的角度来看，迭代法一般需要解决以下问题：

(1) 设置函数 f(x)。本例中设 $f(x)=x^3-x^2-1$。

(2) 分析问题，写出相应的迭代公式。将 $f(x)=0$ 改写为 $x=\varphi(x)$，本例改写 $x=\sqrt[3]{1+x^2}$，为并得到迭代公式(x 称作迭代变量)：

$$x_{n+1}=\sqrt[3]{1+x_n^2}$$

(3) 将迭代变量 x 代入迭代公式进行迭代。将初值 x_0 代入迭代公式求出 x_1，将 x_1 代入迭代公式求出 x_2，依次求出 x_3，x_4，…，x_n，x_{n+1}，…当 $|x_{n+1}-x_n|<\varepsilon$ (本例中为 10^{-5}) 时迭代结束，x_{n+1} 就是满足条件的答案。

(4) 收敛性问题判断。如果迭代变量 x 得出的迭代序列 x_1，x_2，…，x_n，x_{n+1}，…一直朝向方程的根的方向发展，其极限值为一确定常数，则数列 $\{x_k\}$ 是收敛的，否则发散。显然，该数列的收敛性与迭代公式有关。由于迭代公式收敛性的判断不是一个简单的问题，需要计算数学等方面的知识，程序中一般采用控制迭代次数的方法。如果某迭代公式在 M 次迭代之内使 $|x_{n+1}-x_n|<\varepsilon$ 成立，则认为该迭代收敛，否则认为发散。

C 程序如下：

```
# include "stdio.h"
# include "math.h"
# define M 30
float f(float x)
{    return pow(x * x+ 1,1.0/3);}
int main ()
{     float x1,x0,eps;
      int k= 0;
      scanf("%f",&x1);
```

```
    scanf("%f",&eps);
    do
   {    x0= x1;
        x1= f(x0);
        k+ + ;
   }
    while(fabs(x0- x1)> = eps&&k< M);
    if(k< M)
      printf ("迭代次数= %d,x= %f\n",k,x1);
    else
      printf("迭代发散\n");
}
```

运行结果为：

```
1.5 0.00001↙
迭代次数= 11, x= 1.465577
```

为了区分本例中用的求方程解的迭代法和程序设计中迭代所指的迭代法，我们不妨称本迭代法为解方程的迭代法。

【例 1-4】用牛顿迭代法求方程 $x^3+9.2x^2+16.7x+4=0$ 在 $x=0$ 附近的根，迭代精度为 10^{-6}。

按迭代问题解法做：

(1) 设置函数 $f(x)=x^3+9.2x^2+16.7x+4$ 数

(2) 牛顿迭代公式为

$$x_{n+1}=x_n-\frac{f(x_n)}{f'(x_n)}$$

其中 $f(x)=x^3+9.2x^2+16.7x+4$，$f'(x)=3x^2+18.4x+16.7$。

(3) 对迭代变量 x 进行迭代计算。如果根据迭代收敛性判断得知该迭代公式一定收敛，迭代结束的条件只需要 $|x_{n+1}-x_n|<10^{-6}$。

```
# include "stdio.h"
# include "math.h"
# define M 30
float f(float x)
{   return x * x * x+ 9.2 * x * x+ 16.7 * x+ 4;}
float f1(float x)
{   return 3 * x * x+ 18.4 * x+ 16.7;}
int main ()
{    float x1,x0,eps;
     int k= 0;
     scanf("%f",&x1);
```

```
    scanf("%f",&eps);
  do
  {     x0= x1;
        x1= x0- f(x0)/f1(x0);
        k+ + ;
    }
    while(fabs(x0- x1)> = eps&&k< M);
    if(k< M)
      printf ("%f\n",x1);
    else
      printf("迭代发散\n");
}
```

如果无法判定给定初值时迭代的收敛性问题，需要事先给出迭代的最大次数，一旦迭代达到该最大次数还未得到答案，迭代也停止。当然，程序必须有一些相应的变化。此外，以上两种迭代也能解决其他函数的求根问题。以牛顿迭代程序为例，只需要修改求 f(x)和 f(x)导数的那两行程序即可。

【例 1-5】已知方程在 $x^4-5x^2+x+2=0$ 区间［1.5，2.5］内只有一个根，用二分法求出这个根。要求精度满足 10^{-5}。

解方程迭代法和牛顿迭代法都是通过迭代公式迭代求解，二分法是另一种方式的迭代。

（1）设函数 f(x)＝0。在已知区间［a，b］上有唯一的根的前提下才能使用二分法，这些有唯一根的区间称作隔根区间，隔根区间满足 f(a) * f(b)＜0。确定隔根区间可使用试值法（猜测一些小区间，使 f(a) * f(b)＜0），作图法（画草图找出每个根大致的范围），扫描法（编程序用相等的较小区间分割 x 轴可能具有根的大区间，从中找出隔根区间）等。本例中已经给出了一个隔根区间，故只需验证 f(a) * f(b)是否小于 0 即可。

（2）进行迭代的操作很简单：取［a，b］的中点 c，如果｜a－c｜＜ε（本例中为 10^{-5}），c 就是所求的根。否则再判 f(a) * f(c)，若该值大于或等于 0，其根在［c，b］之间；否则在［a，c］之间。重复这些操作，一定能找到满足条件的解，不必像前边介绍的迭代方法那样规定一个控制发散的迭代次数。

```
# include "stdio.h"
# include "math.h"
  float f(float x)
  {  return x * x * x * x- 5 * x * x+ x+ 2;}
  int main ()
  {float duifen(float a,float b,float eps);
    float a,b,r,eps;
```

```
    scanf("%f%f",&a,&b);
    scanf("%f",&eps);
    r= duifen(a,b,eps);
    printf("%f\n",r);
    }
float duifen(float a,float b,float eps)
{    float c,z;
    while (b- a> = eps)
    { c= (a+ b)/2;
      if(f(c)= = 0)
         return c;
      else if (f(a) * f(c)< 0)
         b= c;
      else
         a= c;
    }
    return c;
}
```

运行结果为：

```
1.5 2.5 0.00001↙
2.000000
```

把求函数值 f(x)的运算用函数子程序实现的好处是通用性强，若想用二分法求另一个方程的根时，只需要修改该函数子程序即可。

迭代不一定非得与精度控制相联系，斐波纳契数列就是这种迭代的典型代表。

【例 1-6】意大利数学家斐波纳契（Fibonacci）在他的书中提出了一个关于兔子繁殖的问题：如果一对兔子每月能生一对小兔（一雄一雌），而每对小兔在它们出生后的第三个月里，又能生一对小兔，假定在不发生死亡的情况下，由一对出生的小兔开始，问前 n 个月每个月有多少对兔子？

对该问题进行分析：第 1、第 2 两个月只有 1 对兔子，第 3 个月时小兔子已经长大，繁殖了 1 对小兔子，故第 3 个月有 2 对兔子（1 对大兔子，1 对小兔子）。第 4 个月时，1 对大兔子又繁殖了 1 对小兔子，此时有 3 对兔子（1 对大兔子，2 对小兔子）。第 5 个月时，第 3 个月出生的小兔子已经长大，这时共有 2 对大兔子，可繁殖 2 对小兔子，故第 5 个月共有 5 对兔子（2 对大兔子，3 对小兔子，其中 1 对小兔子是上个月出生的）。按这种规律繁殖下去，得兔子对序列：1，1，2，3，5，8，13，21，…即从第 3 个月开始，每个月拥有的兔子对数是前两个月兔子对数之和。因此得出斐波纳契数列的定义如下：