计算机基础课程系列教材

C语言程序设计

郝玉洁　袁　平
常　征　易　涛　编著

机械工业出版社
China Machine Press

计算机基础课程系列教材

C语言程序设计

郝玉洁 袁 平 编著
常 征 易 涛

机械工业出版社
China Machine Press

C语言是目前流行的通用程序设计语言，是许多计算机专业人员和计算机的爱好者学习程序设计语言的首选。

本书共10章，分别介绍了C语言的算法基础、基本概念和如何利用C语言进行程序设计。通过大量的实例介绍了C语言的精华，特别针对目前C语言的发展，精心策划了C语言的综合应用、图形与动画的设计、音乐的演奏等。

从结构化程序设计到面向对象的程序设计，其发展前景是不以人的意志为转移的。为了使学生尽快地掌握并顺利地过渡面向对象的程序设计方法，我们专门编写了介绍C++入门知识的一章，以使本教材更加完美。

本书的作者都是长期在高校从事计算机基础教育的一线教师，有丰富的教学经验和写作经验，书中文字流畅、通俗易懂、概念清楚、深入浅出、例题丰富，书后的附录专门配有每章的练习供读者选用。

此书可作为普通高校本、专科学生的教学用书，也可供一般工程技术人员自学使用。

图书在版编目（CIP）数据

C语言程序设计 / 郝玉洁等编著. - 北京：机械工业出版社，2000.8
（计算机基础课程系列教材）
ISBN 7-111-08160-9

I. C… II. 郝… III. C语言-程序设计-教材 IV.TP312

中国版本图书馆CIP数据核字（2000）第45450号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）
责任编辑：赵红燕
北京第二外国语学院印刷厂印刷 · 新华书店北京发行所发行
2000年9月第1版第1次印刷
787mm × 1092 mm 1/16 · 16.5印张
印数：0 001-8 000册
定价：26.00元

前　言

随着计算机技术的深入发展，特别是微机的普及，越来越多的人在学习计算机知识以满足社会发展和自身的需要。

C语言不仅为计算机专业人员所使用和喜爱，近年来，许多计算机应用人员也开始喜爱和使用C语言。实践证明，它是一种很好的程序设计语言。

但C语言对一般初学者来说，规则较多，使用太灵活，不易掌握，学习会有一定的困难；而且C语言的应用范围越来越大，所涉及的知识也在不断地增加。基于此，我们重新编写了C语言教程，一方面以满足初学者的需要，另一方面，对C语言所涉及的深入技术做了一定介绍。此书既可作为C语言的教程，也可作为工具书以备编程时参考。

本教材对C语言的精华部分作了较为细致的介绍。我们还针对目前高等院校和社会上举办的各种计算机的等级考试，精心组织了教材的内容。在教材的第1、2章，介绍了算法的基本思想和C语言的基础知识；第3、4章介绍了C语言的基本程序设计技术、C语言函数的特点、函数的相互调用及变量的特性；第5、6章介绍了C语言的各种不同类型的数组及指向变量的指针和指向各种不同类型的数组的指针的使用和程序设计，详细地介绍了指针的特点和灵活性；第7、8章介绍了链表技术和文件的操作方法；第9章介绍了C语言的综合应用，图形的制作和音乐的演奏方法；第10章介绍了C++的入门知识，以满足不同层次读者的需求。列举的例题都是本书作者的精心设计，并全部在Turbo C 环境下调试通过。

本教材的第1、3章由易涛编写，第2、4、5、9章由袁平编写，第6、7、8章由郝玉洁编写，第10章由常征编写。由于作者水平有限，书中难免会有错误，希望读者和专家提出宝贵意见以帮助我们将此教材进一步完善。

作　者

2000年1月15日

第1章　程序设计与算法

1.1　程序设计语言的发展

自1946年世界上第一台电子计算机问世以来，计算机科学及其应用的发展十分迅猛，计算机被广泛地应用于人类生产、生活的各个领域，推动了社会的进步与发展。特别是随着国际互联网（Internet）日益深入千家万户，传统的信息收集、传输及交换方式正被革命性地改变，我们已经难以摆脱对计算机的依赖，计算机已将人类带入了一个新的时代——信息时代。

新的时代对于我们的基本要求之一是：自觉地、主动地学习和掌握计算机的基本知识和基本技能，并把它作为自己应该具备的基本素质。要充分认识到，缺乏计算机知识，就是信息时代的“文盲”。

对于理工科的大学生而言，掌握一门高级语言及其基本的编程技能是必需的。大学学习，除了掌握本专业系统的基础知识外，科学精神的培养、思维方法的锻炼、严谨踏实的科研作风养成，以及分析问题、解决问题的能力的训练，都是日后工作的基础。学习计算机语言，正是一种十分有益的训练方式，而语言本身又是与计算机进行交互的有力的工具。

一台计算机是由硬件系统和软件系统两大部分构成的，硬件是物质基础，而软件可以说是计算机的灵魂，没有软件，计算机是一台“裸机”，是什么也不能干的，有了软件，才能灵动起来，成为一台真正的“电脑”。所有的软件，都是用计算机语言编写的。

计算机程序设计语言的发展，经历了从机器语言、汇编语言到高级语言的历程。

1. 机器语言

电子计算机所使用的是由“0”和“1”组成的二进制数，二进制是计算机的语言的基础。计算机发明之初，人们只能降贵纡尊，用计算机的语言去命令计算机干这干那，一句话，就是写出一串串由“0”和“1”组成的指令序列交由计算机执行，这种语言，就是机器语言。使用机器语言是十分痛苦的，特别是在程序有错需要修改时，更是如此。而且，由于每台计算机的指令系统往往各不相同，所以，在一台计算机上执行的程序，要想在另一台计算机上执行，必须另编程序，造成了重复工作。但由于使用的是针对特定型号计算机的语言，故而运算效率是所有语言中最高的。机器语言，是第一代计算机语言。

2. 汇编语言

为了减轻使用机器语言编程的痛苦，人们进行了一种有益的改进：用一些简洁的英文字母、符号串来替代一个特定的指令的二进制串，比如，用“ADD”代表加法，“MOV”代表数据传递等等，这样一来，人们很容易读懂并理解程序在干什么，纠错及维护都变得方便了，这种程序设计语言就称为汇编语言，即第二代计算机语言。然而计算机是不认识这些符号的，这就需要一个专门的程序，专门负责将这些符号翻译成二进制数的机器语言，这种翻译程序被称为汇编程序。

汇编语言同样十分依赖于机器硬件，移植性不好，但效率仍十分高，针对计算机特定硬件而编制的汇编语言程序，能准确发挥计算机硬件的功能和特长，程序精炼而质量高，所以

至今仍是一种常用而强有力的软件开发工具。

3. 高级语言

从最初与计算机交流的痛苦经历中，人们意识到，应该设计一种这样的语言，这种语言接近于数学语言或人的自然语言，同时又不依赖于计算机硬件，编出的程序能在所有机器上通用。经过努力，1954年，第一个完全脱离机器硬件的高级语言——FORTRAN问世了，40多年来，共有几百种高级语言出现，有重要意义的有几十种，影响较大、使用较普遍的有FORTRAN、ALGOL、COBOL、BASIC、LISP、SNOBOL、PL/1、Pascal、C、PROLOG、Ada、C++、VC、VB、Delphi、JAVA等。

高级语言的发展也经历了从早期语言到结构化程序设计语言，从面向过程到非过程化程序语言的过程。相应地，软件的开发也由最初的个体手工作坊式的封闭式生产，发展为产业化、流水线式的工业化生产。

60年代中后期，软件越来越多，规模越来越大，而软件的生产基本上是人自为战，缺乏科学规范的系统规划与测试、评估标准，其恶果是大批耗费巨资建立起来的软件系统，由于含有错误而无法使用，甚至带来巨大损失，软件给人的感觉是越来越不可靠，以致几乎没有不出错的软件。这一切，极大地震动了计算机界，史称“软件危机”。人们认识到：大型程序的编制不同于写小程序，它应该是一项新的技术，应该像处理工程一样处理软件研制的全过程。程序的设计应易于保证正确性，也便于验证正确性。1969年，提出了结构化程序设计方法，1970年，第一个结构化程序设计语言——Pascal语言出现，标志着结构化程序设计时期的开始。

80年代初开始，在软件设计思想上，又产生了一次革命，其成果就是面向对象的程序设计。在此之前的高级语言，几乎都是面向过程的，程序的执行是流水线似的，在一个模块被执行完成前，人们不能干别的事，也无法动态地改变程序的执行方向。这和人们日常处理事物的方式是不一致的，对人而言是希望发生一件事就处理一件事，也就是说，不能面向过程，而应是面向具体的应用功能，也就是对象（object）。其方法就是软件的集成化，如同硬件的集成电路一样，生产一些通用的、封装紧密的功能模块，称之为软件集成块，它与具体应用无关，但能相互组合，完成具体的应用功能，同时又能重复使用。对使用者来说，只关心它的接口（输入量、输出量）及能实现的功能，至于如何实现的，那是它内部的事，使用者完全不用关心，C++、VB、Delphi就是典型代表。

高级语言的下一个发展目标是面向应用，也就是说：只需要告诉程序你要干什么，程序就能自动生成算法，自动进行处理，这就是非过程化的程序语言。

1.2 C语言的特点

1.2.1 C语言是中级语言

C语言通常称为中级计算机语言。中级语言并没有贬义，不意味着它功能差、难以使用、或者比BASIC、Pascal那样的高级语言原始，也不意味着它与汇编语言相似，会给使用者带来类似的麻烦。C语言之所以被称为中级语言，是因为它把高级语言的成分同汇编语言的功能结合起来了。表1-1表明了C语言在计算机语言中所处的地位。

表1-1 C语言在计算机语言中的地位

高级	Ada、Modula-2、Pascal、COBOL、FORTRAN、BASIC
中级	C、FORTH、Macro-assembler
低级	Assembler

作为中级语言，C允许对位、字节和地址这些计算机功能中的基本成分进行操作。C语言程序非常容易移植。可移植性表示为某种计算机写的软件可以用到另一种机器上去。举例来说，如果为苹果机写的一个程序能够方便地改为可以在IBM PC 上运行的程序，则称为是可移植的。

所有的高级语言都支持数据类型的概念。一个数据类型定义了一个变量的取值范围和可在其上操作的一组运算。常见的数据类型是整型、字符型和实数型。虽然C语言有五种基本数据类型，但与Pascal或Ada相比，它却不是强类型语言。C程序允许几乎所有的类型转换。例如，字符型和整型数据能够自由地混合在大多数表达式中进行运算。这在强类型高级语言中是不允许的。

C语言的另一个重要特点是它仅有32个关键字，这些关键字就是构成C语言的命令。和IBM PC的BASIC相比，后者包含的关键字达159个之多。

1.2.2 C语言是结构化语言

虽然从严格的学术观点上看，C语言是块结构（block-structured）语言，但是它还是常被称为结构化语言。这是因为它在结构上类似于ALGOL、Pascal和Modula-2（从技术上讲，块结构语言允许在过程和函数中定义过程或函数。用这种方法，全局和局部的概念可以通过“作用域”规则加以扩展，“作用域”管理变量和过程的“可见性”。因为C语言不允许在函数中定义函数，所以不能称之为通常意义上的块结构语言）。

结构化语言的显著特征是代码和数据的分离。这种语言能够把执行某个特殊任务的指令和数据从程序的其余部分分离出去、隐藏起来。获得隔离的一个方法是调用使用局部（临时）变量的子程序。通过使用局部变量，我们能够写出对程序其它部分没有副作用的子程序。这使得编写共享代码段的程序变得十分简单。如果开发了一些分离很好的函数，在引用时我们仅需要知道函数做什么，不必知道它如何做。切记：过度使用全局变量（可以被全部程序访问的变量）会由于意外的副作用而在程序中引入错误。

结构化语言比非结构化语言更易于程序设计，用结构化语言编写的程序的清晰性使得它们更易于维护。这已是人们普遍接受的观点了。C语言的主要结构成分是函数C的独立子程序。在C语言中，函数是一种构件（程序块），是完成程序功能的基本构件。函数允许一个程序的诸任务被分别定义和编码，使程序模块化。可以确信，一个好的函数不仅能正确工作且不会对程序的其它部分产生副作用。

1.2.3 C语言是程序员的语言

也许你会问“所有的计算机语言不都是程序员使用的吗？”，回答是断然的“否”。我们考虑典型的非程序员的语言COBOL和BASIC。COBOL的设计使程序员难以改变所编写代码的可靠性，甚至不能提高代码的编写速度。

然而COBOL设计者的本意却是打算使非程序员能读程序（这是不大可能的事）。注意，

这并不是攻击COBOL的优点，而是想指出，它没有被设计成为程序员的理想语言。BASIC 的主要目的是允许非专业程序员在计算机上编程解决比较简单的问题。与其形成鲜明对照的是C语言，由于程序生成、修改和现场测试自始至终均由真正的程序员进行，因而它实现了程序员的期望：很少限制、很少强求、块结构、独立的函数以及紧凑的关键字集合。用C语言编程，程序员可以获得高效机器代码，其效率几乎接近汇编语言代码。

C语言被程序员广泛使用的另一个原因是可以用它代替汇编语言。汇编语言使用的汇编指令，是能够在计算机上直接执行的二进制机器码的符号表示。汇编语言的每个操作都对应为计算机执行的单一指令。虽然汇编语言给予程序员达到最大灵活性和最高效率的潜力，但开发和调试汇编语言程序的困难是难以忍受的。非结构性使得汇编语言程序难于阅读、改进和维护。也许更重要的是，汇编语言程序不能在使用不同CPU的机器间移植。

最初，C语言被用于系统程序设计。一个“系统程序”是一大类程序的一部分，这一大类构成了计算机操作系统及实用程序。通常被称为系统程序的有：

- 操作系统。
- 翻译程序。
- 编辑程序。
- 汇编程序。
- 编译程序。
- 数据库管理程序。

随着C语言的普及，加之其可移植性和高效率，许多程序员用它设计各类程序。几乎所有的计算机上都有C语言编译程序，这使我们可以很少改动甚至不加改动地将为一种机器写的C语言源程序在另一种机器上编译执行。可移植性节省了时间和财力。

C语言不仅在速度和结构上有它的优势，而且每个C语言系统都提供了专门的函数库，程序员可以根据不同需要对其进行剪裁，以适应各种程序的设计。由于它允许（更准确地说是鼓励）分别编译，所以C语言可使程序员方便地管理大型项目，最大限度地减少重复劳动。

1.3 C语言的程序结构

1.3.1 基本程序结构

任何一种程序设计语言都具有特定的语法规则和规定的表达方法。一个程序只有严格按照语言规定的语法和表达方式编写，才能保证编写的程序在计算机中能正确地执行，同时也便于阅读和理解。

为了了解C语言的基本程序结构，我们先介绍几个简单的C程序。

[例1-1]

```
# include <stdio.h>
main()          /*主函数*/
{
   printf("This is a sample of c program. \n");     /*调用标准函数,
                                                      显示引号中的内容*/
}
```

这是一个最简单的C程序，其执行结果是在屏幕上显示一行信息：

RUN↵

```
This is a sample of c program.
```

[例1-2]

```
main()                  /*主函数*/
{
   void proc();         /* 函数声明 */
   int  a=3;            /*指定a为整数，初始值为3*/
   proc();              /*调用函数proc,无返回*/
   a=func();            /*调用函数func,结果返回给a*/
   printf("This is a sample of c program. \n");
}

void proc()             /*定义函数proc,void指定该函数不返回结果*/
{
   printf("Hello. \n");
}

int func()              /*定义函数func,int指定该函数返回一个整数*/
{
       return(2);       /*返回整数2*/
}
```

本程序的执行过程是：

- 程序从main()处开始。
- 变量a代表一个整数，并且初始值为3。
- 执行程序（函数）proc()；屏幕上显示Hello，\n为转义字符，代表换行的意思。
- 执行程序（函数）func()；并将结果赋予a，此时，a的值为2。
- 屏幕上显示“This is a sample of c program.”。

程序执行的结果是在屏幕显示两行信息：

RUN↵

```
Hello.
This is a sample of c program.
```

程序中/*.....*/表示对程序的说明（称为注释），不参与程序的运行。注释文字可以是任意字符，如汉字、拼音、英文等。

[例1-3]

```
/*输入长方体的长、宽、高，计算长方体体积*/
main()
{
   int x,y,z,v;                          /*定义整型变量*/
   scanf("%d,%d,%d",&x,&y,&z);           /*调用标准函数，从键盘输入x,y,z的值*/
   v = volume(x,y,z);                    /*调用volume函数，计算体积*/
   prinf("v = %d\n",v);
}
```

```
int volume(a,b,c)                    /*定义volume函数*/
    int a,b,c;                       /*对形参a,b,c作类型定义*/
    {
    int p;                      /*定义函数内部使用的变量p*/

    p = a*b*c;                  /*计算体积p的值*/
    return(p);                  /*将p值返回调用处*/
    }
```

本程序的功能是对从键盘输入的长方体的长、宽、高三个整型量求其体积的值。程序运行的情况如下：

```
RUN↵
5,8,6↵
v = 240
```

在本例中，main函数在调用volume函数时，将实际参数x、y、z的值分别传送给volume函数中的形式参数a、b、c。经过执行volume函数得到一个结果（即volume函数中变量p的值）并把这个值赋给变量v。

从上面程序例子，可以看出C程序的基本结构。

C程序为函数模块结构，所有的C程序都是由一个或多个函数构成，其中必须只能有一个主函数main()。程序从主函数开始执行，当执行到调用函数的语句时，程序将控制转移到调用函数中执行，执行结束后，再返回主函数中继续运行，直至程序执行结束。C程序的函数是由编译系统提供的标准函数（如printf、scanf等）和由用户自己定义的函数（如proc、func、volume等）。虽然从技术上讲，主函数不是C语言的一个成分，但它仍被看做是其中的一部分，因此，“main”不能用作变量名。

函数的基本形式是：

```
函数类型    函数名(形式参数)
形式参数说明;
{
        数据说明部分;
        语句部分;
}
```

其中：

函数头　包括函数说明、函数名和圆括号中的形式参数（如int volume(a,b,c)），如果函数调用无参数传递，圆括号中形式参数为空（如void proc()函数）。

形式参数说明　指定函数调用传递参数的数据类型（如例1.3中语句int a,b,c;）。

函数体　包括函数体内使用的数据说明和执行函数功能的语句，花括号{和}表示函数体的开始和结束。

1.3.2 函数库和链接

从技术上讲，纯粹由程序员自己编写的语句构成C语言程序是可能的，但这却是罕见的。因为所有的C编译程序都提供能完成各种常用任务的函数——函数库（如printf、scanf等）。

C编译程序的实现者已经编写了大部分常见的通用函数。当我们调用一个别人编写的函数

时编译程序“记忆”它的名字。随后，“链接程序”把我们编写的程序同标准函数库中找到的目标码结合起来，这个过程称为“链接”。

保存在函数库中的函数是可重定位的。这意味着其中机器码指令的内存地址并未绝对地确定，只有偏移量是确定的。当把程序与标准函数库中的函数相链接时，内存偏移量被用来产生实际地址。有关重定位的详细内容，请查阅其他技术书籍。

编写程序时用到的函数，许多都可以在标准函数库中找到。它们是可以简单地组合起来的程序构件。编写了一个经常要用的函数之后，可将其放入库中备用。

1.3.3 开发一个C程序

开发一个C程序，包括以下四步：

1) 程序设计 程序设计亦称程序编辑。程序员用任一编辑软件（编辑器）将编写好的C程序输入计算机，并以文本文件的形式保存在计算机的磁盘上。编辑的结果是建立C源程序文件。C程序习惯上使用小写英文字母，常量和其他用途的符号可用大写字母。C语言对大、小写字母是有区别的。关键字必须小写。

2) 程序编译 编译是指将编辑好的源文件翻译成二进制目标代码的过程。编译过程是使用C语言提供的编译程序（编译器）完成的。不同操作系统下的各种编译器的使用命令不完全相同，使用时应注意计算机环境。编译时，编译器首先要对源程序中的每一个语句检查语法错误，当发现错误时，就在屏幕上显示错误的位置和错误类型的信息。此时，要再次调用编辑器进行查错修改。然后，再进行编译，直至排除所有语法和语义错误。正确的源程序文件经过编译后在磁盘上生成目标文件。

3) 链接程序 编译后产生的目标文件是可重定位的程序模块，不能直接运行。链接就是把目标文件和其他分别进行编译生成的目标程序模块（如果有的话）及系统提供的标准库函数链接在一起，生成可以运行的可执行文件的过程。链接过程使用C语言提供的链接程序（链接器）完成，生成的可执行文件存在磁盘中。

4) 程序运行 生成可执行文件后，就可以在操作系统控制下运行。若执行程序后达到预期目的，则C程序的开发工作到此完成。否则，要进一步检查修改源程序，重复编辑——编译——链接——运行的过程，直到取得预期结果为止。

大部分C语言都提供一个独立的开发集成环境，它可将上述四步连贯在一个程序之中。本书所涉及的程序全部在Turbo C环境中进行。

1.3.4 C语言的关键字

表1-2列举了32个关键字，它们与标准C句法结合，形成了程序设计语言C。

表1-2 关 键 字

auto	break	case	char	const	continue	default	do
double	else	enum	extern	float	for	goto	if
int	long	register	short	signed	sizeof	static	return
struct	switch	typedef	union	unsigned	void	volatile	while

C语言的关键字都用小写字母。C语言中区分大写与小写，else是关键字，“ELSE”则不是。在C程序中，关键字不能用于其它目的，即不允许将关键字作为变量名或函数名使用。

1.4 算法

什么是程序？程序 = 数据结构 + 算法。

对于面向对象程序设计，强调的是数据结构，而对于面向过程的程序设计语言如C、Pascal、FORTRAN等语言，主要关注的是算法。掌握算法，也是为面向对象程序设计打下一个扎实的基础。那么，什么是算法呢？

人们使用计算机，就是要利用计算机处理各种不同的问题，而要做到这一点，人们就必须事先对各类问题进行分析，确定解决问题的具体方法和步骤，再编制好一组让计算机执行的指令即程序，交给计算机，让计算机按人们指定的步骤有效地工作。这些具体的方法和步骤，其实就是解决一个问题的算法。根据算法，依据某种规则编写计算机执行的命令序列，就是编制程序，而书写时所应遵守的规则，即为某种语言的语法。

由此可见，程序设计的关键之一，是解题的方法与步骤，是算法。学习高级语言的重点，就是掌握分析问题、解决问题的方法，就是锻炼分析、分解，最终归纳整理出算法的能力。与之相对应，具体语言，如C语言的语法是工具，是算法的一个具体实现。所以在高级语言的学习中，一方面应熟练掌握该语言的语法，因为它是算法实现的基础，另一方面必须认识到算法的重要性，加强思维训练，以写出高质量的程序。

下面通过例子来介绍如何设计一个算法：

[例1-4] 输入三个数，然后输出其中最大的数。

首先，得先有个地方装这三个数，我们定义三个变量A、B、C，将三个数依次输入到A、B、C中，另外，再准备一个MAX装最大数。

由于计算机一次只能比较两个数，我们首先把A与B比，大的数放入MAX中，再把MAX与C比，又把大的数放入MAX中。

最后，把MAX输出，此时MAX中装的就是A、B、C三数中最大的一个数。算法可以表示如下：

1) 输入A、B、C。

2) A与B中大的一个放入MAX中。

3) 把C与MAX中大的一个放入MAX中。

4) 输出MAX，MAX即为最大数。

其中的2)、3)两步仍不明确，无法直接转化为程序语句，可以继续细化：

2) 把A与B中大的一个放入MAX中，若A>B，则MAX ← A；否则MAX ← B。

3) 把C与MAX中大的一个放入MAX中，若C>MAX，则MAX←C。

于是算法最后可以写成：

1) 输入A，B，C。

2) 若A>B，则MAX ← A；

否则MAX←B。

3) 若C>MAX，则MAX← C。

4) 输出MAX，MAX即为最大数。

这样的算法已经可以很方便地转化为相应的程序语句了。

[例1-5] 猴子吃桃问题：有一堆桃子不知数目，猴子第一天吃掉一半，觉得不过瘾，又多吃了一只，第二天照此办理，吃掉剩下桃子的一半另加一个，天天如此，到第十天早上，猴子发现只剩一只桃子了，问这堆桃子原来有多少个？

此题粗看起来有些无从着手的感觉，那么怎样开始呢？假设第一天开始时有a_1只桃子，第二天有a_2只，...，第9天有a_9只，第10天是a_{10}只，在$a_1,a_2,...,a_{10}$中，只有$a_{10}=1$是知道的，现要求a_1，而我们可以看出，$a_1,a_2,..,a_{10}$之间存在一个简单的关系：

$a_9=2*(a_{10}+1)$

$a_8=2*(a_9+1)$

⋮

$a_1=2*(a_2+1)$

也就是：$a_i=2*(a_{i+1}+1)$ i=9,8,7,6,...,1

这就是此题的数学模型。

再考察上面从a_9，a_8直至a_1的计算过程，这其实是一个递推过程，这种递推的方法在计算机解题中经常用到。另一方面，这九步运算从形式上完全一样，不同的只是a_i的下标而已。由此，我们引入循环的处理方法，并统一用a_0表示前一天的桃子数，a_1表示后一天的桃子数，将算法改写如下：

1) a_1=1; {第10天的桃子数，a_1的初值}

i=9。 {计数器初值为9}

2) $a_0=2*(a_1+1)$。 {计算当天的桃子数}

3) $a_1=a_0$。 {将当天的桃子数作为下一次计算的初值}

4) i=i-1。

5) 若i>=1，转2)。

6) 输出a_0的值。

其中2) ~ 5)步为循环。

这就是一个从具体到抽象的过程，具体方法是：

1) 弄清如果由人来做，应该采取哪些步骤。

2) 对这些步骤进行归纳整理，抽象出数学模型。

3) 对其中的重复步骤，通过使用相同变量等方式求得形式的统一，然后简练地用循环解决。

算法的描述方法有自然语言描述、伪代码、流程图、N-S图、PAD图等。

1.4.1 流程图与算法的结构化描述

1. 流程图

流程图是一种传统的算法表示法，它利用几何图形的框来代表各种不同性质的操作，用流程线来指示算法的执行方向。由于它简单直观，所以应用广泛，特别是在早期语言阶段，只有通过流程图才能简明地表述算法，流程图成为程序员们交流的重要手段，直到结构化的程序设计语言出现，对流程图的依赖才有所降低。

下面介绍常见的流程图符号及流程图的例子。

本章例1-1的算法的流程图如图1-2所示。本章例1-2的算法的流程图如图1-3所示。

在流程图中，判断框左边的流程线表示判断条件为真时的流程，右边的流程线表示条件为假时的流程，有时就在其左、右流程线的上方分别标注“真”、“假”或“T”、“F”或“Y”、“N”。

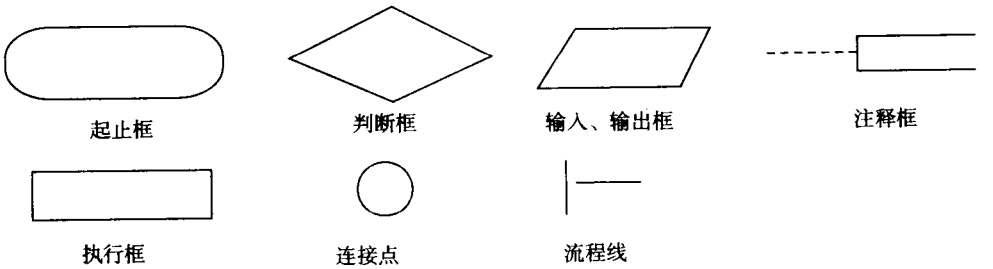

图1-1　常见的流程图符号

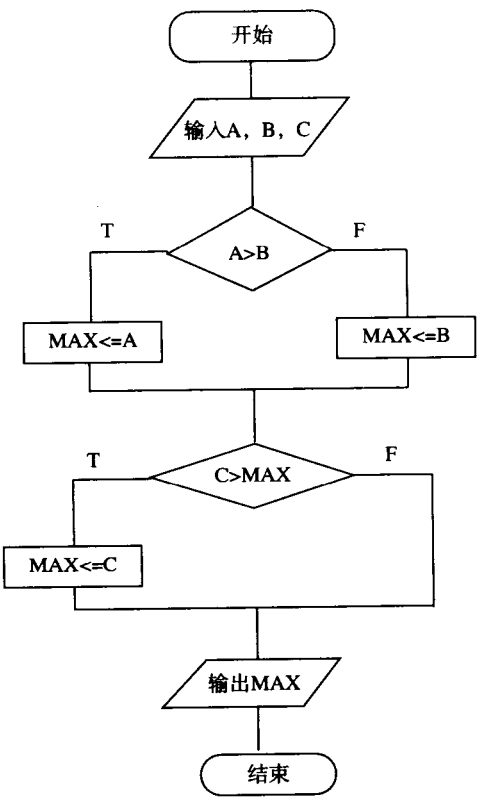

图1-2　例1-1的算法流程图

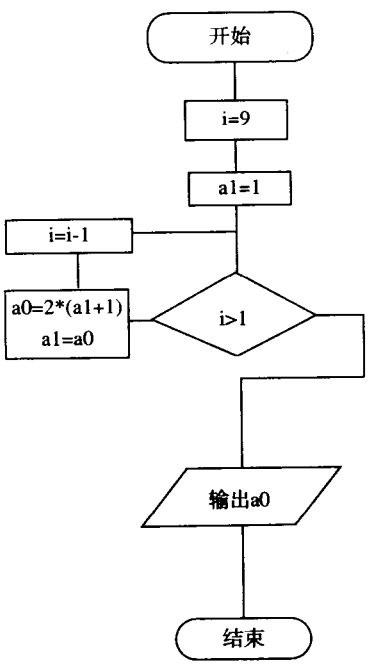

图1-3　例1-2的算法流程图

另外还规定，流程线是从下往上或从右向左时，必须带箭头，除此以外，都不画箭头，流程线的走向总是从上向下或从左向右。

2. 算法的结构化描述

早期的非结构化语言中都有goto语句，它允许程序从一个地方直接跳转到另一个地方去。执行这样做的好处是程序设计十分方便灵活，减少了人工复杂度，但其缺点也是十分突出的，

一大堆跳转语句使得程序的流程十分复杂紊乱，难以看懂也难以验证程序的正确性，如果有错，排起错来更是十分困难。这种转来转去的流程图所表达的混乱与复杂，正是软件危机中程序人员处境的一个生动写照。而结构化程序设计，就是要把这团乱麻理清。

经过研究，人们发现，任何复杂的算法，都可以由顺序结构、选择（分支）结构和循环结构这三种基本结构组成，因此，我们构造一个算法的时候，也仅以这三种基本结构作为“建筑单元”，遵守三种基本结构的规范，基本结构之间可以并列、可以相互包含，但不允许交叉，不允许从一个结构直接转到另一个结构的内部去。正因为整个算法都是由三种基本结构组成的，就像用模块构建的一样，所以结构清晰，易于正确性验证，易于纠错，这种方法，就是结构化方法。遵循这种方法的程序设计，就是结构化程序设计。

相应地，只要规定好三种基本结构的流程图的画法，就可以画出任何算法的流程图。

(1) 顺序结构

顺序结构是简单的线性结构，各框按顺序执行。其流程图的基本形态如图1-4所示，语句的执行顺序为：A→B→C。

(2) 选择（分支）结构

这种结构是对某个给定条件进行判断，条件为真或假时分别执行不同的框的内容。其基本形状有两种，如图1-5 a）、b）所示。图1-5 a）的执行序列为：当条件为真时执行A，否则执行B；图1-5b）的执行序列为：当条件为真时执行A，否则什么也不做。

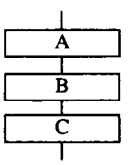

图1-4 顺序结构的流程图

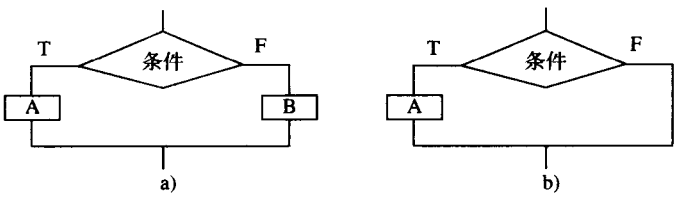

图1-5 选择（分支）结构的流程图

(3) 循环结构

循环结构有两种基本形态：while型循环和do-while型循环。

a. while 型循环

如图1-6所示。

其执行序列为：当条件为真时，反复执行A，一旦条件为假，跳出循环，执行循环紧后的语句。

b. do-while型循环

如图1-7所示。

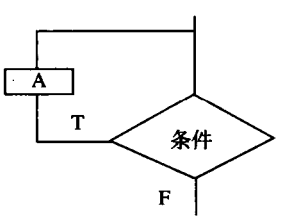

图1-6 while 型循环流程图

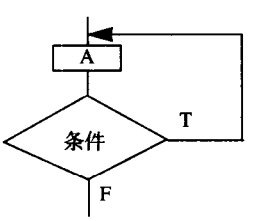

图1-7 do-while型循环流程图

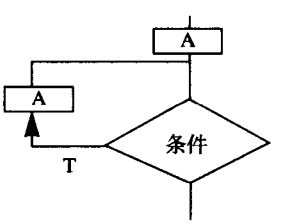

图1-8 do-while 型循环转换为while型循环

执行序列为：首先执行A，再判断条件，条件为真时，一直循环执行A，一旦条件为假，结束循环，执行循环紧后的下一条语句。

在图1-6、图1-7中，A被称为循环体，条件被称为循环控制条件。要注意的是：

1) 在循环体中，必然对条件要判断的值进行修改，使得经过有限次循环后，循环一定能结束，如图1-3中的i=i-1。

2) 当型循环中循环体可能一次都不执行，而直到型循环则至少执行一次循环体。

3) 直到型循环可以很方便地转化为当型循环，而当型循环不一定能转化为直到型循环。例如，图1-7可以转化为图1-8。

1.4.2 用N-S图描述算法

N-S图是另一种算法表示法，是由美国人I.Nassi和B.Shneiderman共同提出的，其根据是：既然任何算法都是由前面介绍的三种结构组成，所以各基本结构之间的流程线就是多余的，因此，N-S图也是算法的一种结构化描述方法。

N-S图中，一个算法就是一个大矩形框，框内又包含若干基本的框，三种基本结构的N-S图描述如下所示：

1. 顺序结构

如图1-9所示，执行顺序先A后B。

2. 选择结构

对应于图1-5的N-S图为图1-10。图1-10 a)条件为真时执行A，条件为假时执行B。图1-10 b)条件为真时执行A，为假时什么都不做。

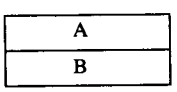

图1-9　顺序结构的N-S图

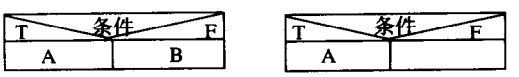

图1-10　选择结构的N-S图

3. 循环结构

1) while型循环的N-S图如图1-11所示，条件为真时一直循环执行循环体A，直到条件为假时才跳出循环。

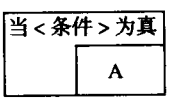

图1-11　while型循环的N-S图

2) do-while型循环的N-S图如图1-12，一直循环执行循环体A，直到条件为假时才跳出循环。

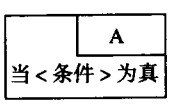

图1-12　do-while型循环的N-S图

本章例1-1的N-S图如图1-13，例1-2的N-S图如图1-14。应该说，N-S图比流程图更直观易懂，而且相对简练一些。

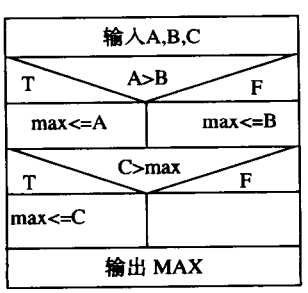

图1-13　例1-1的N-S图

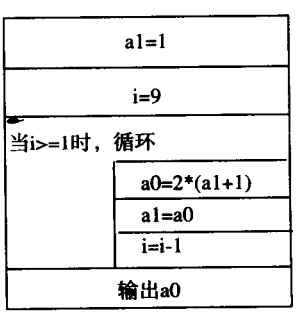

图1-14　例1-2的N-S图

1.4.3 用PAD图描述算法

PAD（Problem Analysis Diagram），是近年来在软件开发中被广泛使用的一种算法的图形表示法，与前述的流程图、N-S图相比，流程图、N-S图都是自上而下的顺序描述，而PAD图除了自上而下以外，还有自左向右的展开，所以，如果说流程图、N-S图是一维的算法描述的话，则PAD图就是二维的，它能展现算法的层次结构，更直观易懂。

下面是PAD图的几种基本形态：

1. 顺序结构：

如图1-15所示。

2. 选择结构

(1) 单分支选择，条件为真执行A，如图1-16 a)。

(2) 两分支选择，如图1-16 b)，条件为真执行A，为假执行B。

(3) 多分支选择，如图1-16 c)，当I=I_1时执行A，I =I_2时执行B，I=I_3时执行C，I=I_4时执行D。

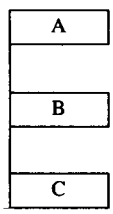

图1-15 顺序结构的PAD

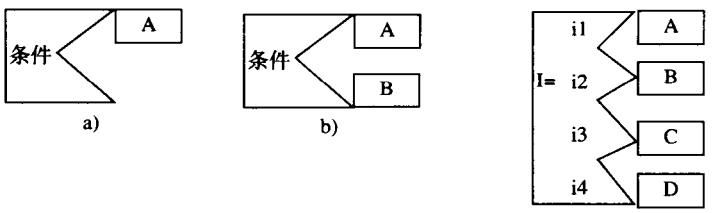

图1-16 选择结构的PAD

3. 循环结构

如图1-17所示。图1-17 a)为while型循环，图1-17 b)为do-while型循环。

图1-17 循环结构的PAD

本章例1.1的PAD图如图1-18，例1-2的PAD图如图1-19。

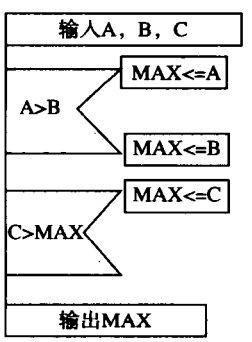

图1-18 例1-1的PAD

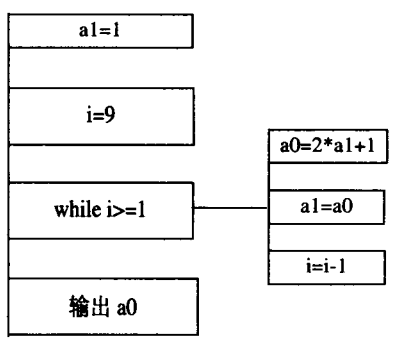

图1-19 例1-2的PAD