第1章 绪 论

C语言是当今世界上使用最广泛的程序设计语言之一。本章主要介绍程序设计语言、程序设计、算法的概念及程序设计的主要步骤，同时介绍C语言的产生和发展以及C程序的构成和书写格式等。

1.1 程序设计语言

人们为了用计算机来解决实际问题，一般总是要编写程序。所谓程序，是指以某种程序设计语言为工具对解决问题的动作序列的描述，它表达了人们解决问题的过程，用于指挥计算机进行一系列操作，从而实现问题的解决。程序设计语言就是用户用来编写程序的语言，它是人与计算机进行信息交流的工具。对于理想的程序设计语言来说，所提供的语法应该能够满足描述算法、数据结构等方面的需要。然而用任何一种高级语言编写的程序（称其为源程序）都要通过编译程序翻译成机器语言程序（称其为目标程序）后计算机才能执行，或者通过解释程序边解释边执行。

程序设计语言是计算机软件系统的重要组成部分，就其发展过程而言，一般可分为机器语言、汇编语言和高级语言。

1.1.1 机器语言

最初的程序是用机器语言编写的，它包括了很多的基本指令，这些机器指令可以由机器直接执行。机器语言编写的程序是由二进制代码组成的代码序列。使用机器指令进行程序设计要求程序设计者具有深入的计算机专业知识，对机器的硬件有充分的了解。这种程序的可读性差，而且由于不同机器的机器指令不同，因此程序的可移植性差，所编写的程序只能在相同的硬件环境下使用，大大地限制了程序的通用性。另外，用机器语言描述问题的处理方式也与人们习惯的思维方式有较大差距。

1.1.2 汇编语言

为了便于记忆，人们很自然地用助记符号来代表机器语言中的01代码，这种用助记符号描述的指令系统称为汇编语言。为了把汇编语言编写的程序转换为机器语言程序，人们开发出了称为“汇编程序”的翻译程序。汇编语言指令与机器语言指令是一一对应的，因此，汇编语言也是与具体计算机硬件相关的。汇编语言除了可读性比机器语言好外，同样也存在机器语言的缺点，尤其是描述问题的方式与人们习惯相差太远。

1.1.3 高级语言

为了提高程序开发的效率,针对机器语言和汇编语言的缺点,20 世纪 50 年代中期,IBM 公司的一个编程小组提出了一个设想:能不能使用数学公式来编写人们想要进行的计算,然后再让计算机将这些公式翻译(解释)为机器语言呢?最终该团队于 1955 年完成了 Fortran(Formulation Translation)的最初版本,这是第一个高级编程语言。从那以后,各种高级语言相继涌现,而能引起广泛关注和使用的主要有 BASIC、Algol、COBOL 、Pascal、C 等。在这些众多的高级语言当中,最为流行,也是最成功的当数 C 语言,它既可用来写应用软件,又可用来写系统软件(如 UNIX 操作系统等)。如今时兴的 C++ 和 Java 也都是基于 C 语言的。

与低级语言(机器语言、汇编语言)相比,高级语言的表达方式更接近人类自然语言的表述习惯,具有很高的可读性,并且它不依赖于计算机的具体型号,具有良好的可移植性。高级语言的一条语句通常对应于多条机器指令,所以对同一功能的描述,高级语言程序比机器指令程序紧凑得多。

1.2 程序设计的概念

程序设计是一门用计算机解决问题的科学,在掌握程序设计的错综复杂的内容之前,有必要对什么是程序设计有一个感性认识。

1.2.1 程序设计

程序设计的目的就是用计算机解决问题。用计算机解决问题大体上经过两个步骤,首先通过分析问题构造出一个解决问题的算法(即解决问题的方法和步骤)或在解决该问题的备用算法中选择其中之一,这个过程称为算法设计;其次是用一种程序设计语言(如 C 语言)将该算法表达为程序,这个过程称为编码。程序设计的两个步骤是相辅相成的,作为一个程序设计(或编程)新手,只能从简单的问题入手,而简单问题的解决方法也较简单,所以算法设计阶段不会有什么困难,倒是 C 语言对于初学者来说是全新的,因此初学程序设计时,编码常常会是两个步骤中较为困难的阶段。但可以确定的是,随着对 C 语句语法的了解和编程实践的不断增加,编码会变得越来越简单;而与此正好相反的是,随着遇到的问题愈来愈复杂,算法设计会变得更加困难和更具有挑战性。

程序设计既是一门科学,又是一门艺术。就像练习写作一样,你必须不断地编程实践并且大量阅读他人程序,来积累经验,才能形成良好的程序设计风格,而不仅仅只是记住一些规则。只有经过长期编程实践训练,才能不断提高程序设计能力。

1.2.2 程序设计的步骤

对于初学编程的人来说,往往简单地把程序设计理解为编写一个程序,认为能根据实际问题直接用计算机语言编出一个程序就行了,这样理解是不全面的。事实上,程序是程序设

计的最终产品,需要经过中间每一步的细致加工才能得到。如果企图一开始就编写出程序,往往会适得其反,达不到预想的结果。通常应在充分论证数据结构和算法以后才能考虑编写程序,同时编写程序时还需要结合程序设计方法(面向过程的或是面向对象的)和程序设计语言(C 语言、C ++ ,Java 等)。因此,不提倡一开始就编写程序,特别是对于大型的程序。前面介绍程序设计过程由算法设计和编码两步骤组成只是大略说法,严格地说,程序设计一般应遵循以下步骤。

1. 分析问题

用计算机来解决问题,首先要通过对问题的分析,以便确定在解决这个问题过程中要做些什么? 若在没有把所要解决的问题分析清楚之前就贸然开始编制程序,只能起到事倍功半的效果,而且很难得到满意结果。因此,分析问题,弄清楚要解决的问题并给出问题的明确定义是解决问题的关键。

2. 系统设计

在弄清要解决的问题之后,就要考虑如何解决它,即如何做? 由于从本质上看,用计算机解决问题的方式就是对数据进行处理,因此,首先要对问题进行抽象,抽取出能够反映本质特征的数据并对其进行描述,即给出数据结构的设计; 然后考虑对设计好的数据如何进行操作以获得问题的结果,即进行算法设计。注意,不同的程序设计方式在处理数据结构和算法这两者的关系上是有所区别的。在面向过程程序设计中,把数据结构设计和算法设计分开考虑; 而面向对象程序设计是把两者结合成对象来考虑。

3. 用某种程序设计语言编程

在进行数据结构设计和算法设计时,往往采用某种与具体程序设计语言无关的语言(如伪代码或自然语言等)来描述算法。这样做的目的是为了避免一开始就陷入程序设计语言的具体细节中。因为过多地涉及实现细节,不利于从较高抽象层次对问题本质的东西进行考虑,并造成对设计过程难以把握和理解。当然,用伪代码(自然语言或框图)描述的算法是不能被计算机所执行的,必须用具体程序设计语言把它表示出来,即编程实现。因此,用某种程序设计语言编写的程序,本质上也是对问题处理方案的描述,并且是最终的描述。程序文本保存在一个文件或多个文件中。包含程序文本的文件称为源文件,即源程序文件。

4. 测试与调试

源程序文件要经过编译程序把它翻译成等价的目标程序文件,并将一系统列目标文件及库文件链接在一起生成一个可执行文件,程序才能被计算机执行。

程序编写好后,还需要进行调试和测试。只有经过调试的程序才能正式运行。所谓调试,是指找出程序中的错误的位置并改正错误。因此,调试又称查错。而所谓测试,是指利用精心设计的一批测试用例(包括输入数据和与之相应的预期输出结果)来运行程序,看程序的实际运行结果与预期输出结果是否一致,以尽可能多地发现程序中的错误。因此,测试的目的也是为了发现程序中的错误(一般是算法错误而非语法错误),而不是为了证明程序正确。调试和测试往往是交替进行的,通过测试发现程序中的错误,通过调试进一步找出错

误的位置并改正错误。这个过程往往需要重复多次,因此这一步往往是程序设计中最困难的一步。

程序编写好之后,其中可能含有错误。程序的错误通常有3种:语法错误、逻辑错误和运行异常错误。语法错误是指源程序中存在有违反C语法规则的地方,如语句后遗漏了分号“;”或忘记了定义变量等,程序编译时编译器可以发现这类错误,并会给出“出错信息提示”和出错位置,用户可以根据编译器提供的提示信息修改源程序;逻辑错误是指程序没有完成预期的功能,如源程序中错将“c=a%b”写成“c=a/b”或将该用“==”的地方却误用了“=”等,编译程序时,编译器发现不了这类错误,但程序运行时,结果会不正确,甚至根本就无法显示结果。遇到此类错误,用户往往需要通过设置断点,跟踪程序的运行过程等测试手段,才能发现它们。因此程序设计中最致命的错误就是逻辑错误,因为它不太容易被发现,并会导致程序不能正确地解决问题。在程序中找出并修正逻辑错误的过程称为程序调试,它是软件开发中不可缺少的一个环节。俗话说,“三分编程七分调试”,说明程序调试的工作量要比编程的工作量大得多。而善于发现编程中的逻辑错误是通往成功的重要一环。优秀编程者的优秀之处就是他们能尽力将存在于完成程序中的逻辑错误数量减到最少,而不在于他们能够避免犯逻辑错误。运行异常错误是指对程序运行环境的非正常情况考虑不足而导致的程序运行异常终止。逻辑和运行异常错误可能是编程阶段导致的,也有可能是系统设计阶段或问题分析阶段的缺陷。这两类错误一般都要通过测试才能发现。常用的测试方法很多,如静态测试与动态测试。静态测试是不运行程序,而是通过对程序的静态分析,找出逻辑错误。动态测试是利用一些测试数据,通过运行程序观察运行结果是否与预期的结果相符。

需要注意的是,无论采用何种测试手段,都只能发现程序有错,而不能证明程序正确,因为几乎可以肯定,即使程序通过了良好的测试,总还是会有一些隐蔽的错误。一个编程者的任务就是不断找出并改正这些错误,并尽可能地对程序进行彻底测试。在大多情形时,彻底测试可以增加对解决方案正确性的把握。

5. 整理并写出所有的文档资料

有些人认为程序调试成功就万事大吉了,其实这种思想是错误的,因为对于程序设计人员来说,平时的归纳和总结是很重要的。程序员应将平时的源程序和各种文字资料进行归类保存,以便今后查找。最后还要编写使用和维护该程序的说明书,供他人参考。

6. 运行与维护

程序通过测试后就可交付使用了,但程序在使用中还需要不断得到维护。程序维护可分为3种:正确性维护、完善性维护和适应性维护。程序需要维护的原因主要有两个,首先,即使经过大量测试,源代码中依然可能存在逻辑错误,这些错误会在程序的使用过程中不断被暴露;其次,当出现一些不常见的情况或发生之前未预料到的情况时,之前隐藏的异常错误就会使程序运行失败。因此,测试与调试是程序维护的一个基本部分。但它不是最重要的,更为重要的是功能完善。编写程序是为了完成客户需要完成的任务,但程序不可能完成客户想做的每一件事。在一个程序使用了一段时间后,客户会期望程序能做些别的事情,或者用不同的方法做某些事情,或者用更有用的方式显示数据,或者运行得更快一些等。

对于用户提出的完善和扩充程序功能的要求，程序开发者应考虑响应这些要求。无论是要排除逻辑错误还是要完善程序功能，在任何情况下都需要有人查看程序，做出必要的修改，并确保这些修改能使程序更好地工作。

如果对程序的功能进行了较大的更改，应发布程序的新版本。

1.3　C 语言的发展和 C ++ 简介

1.3.1　C 语言发展简述

1972 年，为了编写 UNIX 操作系统，贝尔实验室的 D. M. Ritchie 设计并实现了 C 语言。

后来，C 语言多次做了改进，逐渐成熟，先后被移植到各种计算机上，现在 C 语言已成为世界上使用最广泛的程序设计语言之一。C 语言在其 30 多年的发展中，涌现了众多不同的版本，它们都普遍遵守两个重要的标准：一是 Brian W. Kernighan 和 Dennis M. Ritchie 于 1978 年合著的名著《The C Programming Language》，被称为标准 C。二是美国国家标准化协会（ANSI）于 1983 年开始制定并于 1988 年最终完成的 ANSI 标准，即“ANSI C”。ANSI C 比原来的标准 C 有了很大的发展。20 世纪 90 年代后，尽管 C ++ 、Visual C ++ 开发如火如荼，C 语言也并没有停滞不前（更没有被替代），版本不断更新、升级，但 C 的特征未变，C 仍然是 C!。

1.3.2　C ++ 简介

当 C 语言程序达到一定的规模（代码达到 25 000 行以上）后，维护和修改显得相当困难。为了满足管理程序复杂性的需要，贝尔实验室的 Bjarne Stroustrup 博士于 1979 年开始对 C 语言进行了改进和扩充，并引入了面向对象程序设计的内容，最初取名为“带类的”C，1983 年改名为 C ++ 。在经历了 3 次重大修订后，于 1994 年制定了标准 C ++ 草案，后又经不断完善，成为目前的 C ++ ，它具有以下特点：

（1）C ++ 是 C 语言的超集。C ++ 由两部分组成：一是过程性语言部分，这部分与 C 语言无本质区别，一般遵守 ANSI C 标准；二是类和对象部分，这是 C 语言所没有的，它是面向对象程序设计的主体。

（2）C ++ 充分保持了与 C 语言的兼容性，绝大多数 C 语言程序可以不经修改直接在 C ++ 环境中运行。

（3）C ++ 仍然支持面向过程的程序设计，是一种理想的结构化程序设计语言，又几乎全部包含了面向对象程序设计的特征。

（4）C ++ 继承了 C 语言的高效率、灵活性等优点。用 Bjarne Stroustrup 博士的话来说，C ++ 使程序“结构清晰、易于扩展、易于维护而不失效率”。

（5）C ++ 是一种标准化的、与硬件基本无关的、广泛使用的程序设计语言，具有很好的通用性和可移植性。C ++ 程序通常无需修改，或稍作修改，即可在其他计算机系统上运行。

(6) 具有丰富的数据类型和运算符,并提供了功能强大的库函数。

目前主要的C++开发环境有Borland公司的Borland C++、Microsoft公司的Visual C++等。由于Visual C++既能支持C语言程序,又能运行C++程序,同时考虑到C++对C语言的兼容性,因此本书所有例子的编程环境均为Visual C++ 6.0。

1.4 C语言程序的基本结构

C语言属于人造语言,因此C语言与自然语言之间有很多相似之处。自然语言(如英文)由句子、单词和字母组成,C语言也由基本符号(字符)构成一系列单词(语法元素),由多个单词构成句子(语句),再由多条语句构成程序。不同的是C语言具有较严格的语法规则,在语义上也不像自然语言那样具有多义性,程序文本所代表的语义是单一的、确定的。正像写文章时分章节、段落和层次一样,C程序也具有层次结构,但无论其大小如何,它都是由函数组成的,而函数又是由语句组成的。

为了使读者能从整体上对C语言程序的基本结构有一个感性认识,下面先来看一个简单的程序。

例 1.4.1 设圆柱体的半径为r、高为h,求其体积v。

解 程序如下:

```
/*
 * file: compute volume C   程序注释
 * 这程序是用于计算圆柱体体积
 */                                              } 程序注释
#include<stdio.h>                                } 包含库文件
#define PI  3.14159                              } 常量定义
float volume(float r,float h);                   } 函数原型声明
void main()
{float radius,height,vol;          /*声明 radius,height,vol 为 float 型变量*/
printf("input radius,height: ");   /*提示用户输入半径 radius 和高 height 的值*/
scanf("%f%f",&radius,&height);     /*从键盘输入两个实数分别存入变量 radius 和 height*/   } 主程序
vol = volume(radius,height );      /*调用函数 volume()计算圆柱体的体积*/
printf("vol = %f ",vol);           /*输出圆柱体的体积*/
}
 * 函数: volume
 * 用法: v = volume(r,h);
 * 该函数的功能是计算半径为 r、高为 h 的圆柱体的体积                                  } 函数注释
float volume(float r,float h)
{float v;              /*局部变量的定义*/                                            } 函数体
v = PI * r * r * h;                                                                  } 函数定义
return v;
}
```

以上 C 程序由注释、预处理命令、主函数、函数定义等组成。尽管其结构简单，但它是本书所有程序的榜样，应将其作为 C 语言程序组织的范例。

1. 注释

在一个具有良好书写风格的程序开头，往往是该程序的注释部分，因为它有助于人们阅读和理解程序。在 C 语言中，注释是程序中位于符号/ * 和符号 * /之间的所有文字，可以占连续的几行。注释是写给人看的，而不是写给计算机的。当 C 编译器将程序转换成可由机器直接执行的形式时，注释被完全忽略。C ++ 除了兼容 C 注释方法外，还提供一种新的注释，即某行语句后从符号//开始至本行结束的所有字符均为注释内容。

编程时在程序中添加适当的注释是一个良好的编程习惯。在每个规模较大或所处地位较重要的函数开头最好添加关于该函数的注释，包括函数所采用的主要算法、参数含义、所用到的全局变量、编写的时间等。这些注释有助于阅读和理解该函数，对程序维护和重用非常必要。一般来说，注释可以用英文也可以用中文。

2. 预处理命令

一般来说，每个 C 程序的开始部分都会有几行是编译预处理命令。例如程序注释的下一行为

```
#include<stdio.h>
```

是 C/C ++ 中常用的一种编译预处理命令，它说明该程序使用了由 ANSI C 提供的标准 I/O（输入/输出）库文件 stdio.h。每个 C 程序一般都要用到这个库。库是一些工具的集合，这些工具是其他程序员编写，用于执行特定的功能。熟悉各种库对编程者来说是十分重要的，因为当你开始编写一些更为复杂的程序时，还会需要用到其他一些重要的库，这时需要分别使用相应的 #include 命令将它们包含进来。只有这样，你才可以在程序中使用这些库所提供的工具，从而省去自己编写这些工具的麻烦。

预处理命令并非 C 语句，是在 C 源程序编译之前由预处理程序处理的命令，预处理命令以字符 # 开头，有关编译预处理命令的用法请参见第 7 章。

3. 程序级定义

在 #include 命令之后，大多数的程序都会包含对整个程序有效的一些定义，如符号常量定义、新的数据类型定义、全局变量定义等。在本程序中只包含了下面一行：

```
#define PI 3.14159
```

它虽然也属编译预处理命令，但 C 可用它来定义符号常量，经过定义，PI 就代表 3.14159。

4. 函数原型声明

在 C 程序中，所有计算是由相关的函数来执行的。函数是能够完成一定操作、具有具体名称的一组语句。在本程序中包含了 main 和 volume 两个函数。下面一行代码

```
float volume(float r,float h);
```

是函数原型的一个示例,它使得 C 编译器能够检查函数调用是否和相应的函数定义一致,从而帮助在代码中发现错误,因此在程序中包含每个要调用函数的函数原型,是一个良好的编程习惯,尽管 ANSI C 允许在某些特定的情况下省略函数原型声明。

5. main 函数

每个完整的 C 程序,无论其功能大小,都是由一个或多个函数组成的,而这些函数可以是系统提供的也可以是用户自定义的,但其中总有一个且仅有一个被称作主函数的特殊函数,即 main()。该函数为程序指明了起点,即每个程序总是从 main()作为起点开始执行的,而不论 main 函数在整个程序中的位置如何(main 函数可以放在程序前头,也可以放在程序最后,或在一些函数之前,在另一些函数之后)。在主函数 main()中,通常会调用若干函数来帮助完成某些工作,被调用的函数可以是由编程员自己编写的,如本例中的函数 volume;也可以是来自于函数库,如本例中的 scanf 和 printf。一般较大的程序都由多个函数组成,为程序结构清晰和管理方便,通常将一些关系密切的函数组织在一起放在同一个文件中,因此,C 程序通常由多个源文件组成,并且每个源文件可以单独编译。

6. 用户自定义函数

由于较大的程序理解和修改都不方便,因此大部分的程序都被分成若干个较小的函数。每个函数通常用于完成相对独立的功能,作为程序的一个模块,由一组相关的语句组成。函数具有严格的格式,函数可以有参数和返回值。调用函数时必须遵守函数所定义的格式。一个函数在被调用过程中可以再调用其他函数,也可以调用自己(称为递归,详见6.5 节)。在本例中,函数 volume()是一个用户自定义函数,其功能是计算一个半径为 r,高为 h 的圆柱体的体积。主函数通过函数调用语句"vol=volume(radius,height);"调用它。调用 volume()时,先将实际参数 radius,height 的值分别传递给对应的形式参数 r 和 h,然后依次执行 volume()函数中的各个语句,遇到"return v;"语句后将 v 的值作为函数的返回值返回给主函数,并结束 volume()函数的执行,变量 vol 将获得调用 volume()函数所返回的 v 值。

为了使读者对 C 语言程序的工作过程有更好的了解,再举一个例子,它要求用户输入两个整数,将两个数相加并显示结果。

例 1.4.2 求两个整数的和。

解 程序如下:

```
#include<stdio.h>
void main( )        /*求两数之和*/
{   int a,b,sum;   /*变量的定义*/
    printf("Input the two numbers: ");
    scanf("%d%d",&a,&b);
    sum = a + b;
    printf("sum is %d\n",sum);
}
```

本程序的作用是求两个整数之和。/*……*/表示注释部分,为便于理解,用汉字作注释,当然也可以用英语或汉字拼音作注释。注释只是给人看的,对编译和运行不起作用。注

释可以加在程序中任何位置。

第 3 行是对变量 a、b、sum 的定义，它告诉编译器每个变量中可以存放一个整型值(即整数)。关键字 int 表示整型。在程序中，变量相当于容器，变量名相当于容器上的标签，变量名在函数运行过程中始终不变，而变量中存放你要处理的数据或计算的结果(称变量的值)。当有新值存入变量时，其原值才会随之改变。例如，在编写该程序时，你不知道用户要将哪两个数相加。当程序运行时，用户才会通过键盘输入两个数。为了在程序中引用这些目前尚未确定的两个数以及它们之和，可创建 3 个变量来保存这些需要记住的值(整数)，并给这些变量命名。一旦要用到变量中存放的数据(即变量的值)时，可使用其变量名。

和任何 C 程序一样，本程序的运行也是通过执行主函数 main()中的每一条语句来实现问题求解的。当执行第 4 行的语句时，屏幕上会显示“Input the two numbers：”，该显示信息用来提示用户应输入几个什么样的值，光标停留在该行的末尾。程序接下来执行第 5 行的语句“scanf("%d%d",&a,&b)；”，函数 scanf()是 stdio.h 库中的一个函数，用于从用户键盘处输入各种类型的数据并存入相应的变量中。本程序在执行语句“scanf("%d,%d",&a,&b)；”时，程序等待用户从键盘键入两个将要相加的整数，如此时用户输入 3,9 并按回车后，它可将 3 和 9 分别保存在变量 a 和 b 中。

第 6 行是一个赋值语句，计算机首先计算赋值号“=”右边表达式的值，并将结果存储到“=”左边的变量中。这里的赋值语句的作用是将变量 a 和 b 中存储的两个值相加，并将结果存储到变量 sum 中。

第 7 行是程序计算结果的输出，在 C 语言中，输出往往是使用 printf 函数来完成。其中"%d"是输入输出的“格式字符串”，用来指定输入输出时的数据类型和格式(详见 3.5 节)，"%d"表示“以十进制整数形式输出”。在执行程序的最后一个语句“printf("sum is %d\n",sum)；”时，printf 函数在屏幕上显示如下信息：

```
sum is 12
```

以上详尽分析了例 1.4.2 程序中每一个语句的功能。这种逐句分析是想从程序的每个独立部分的层面上理解程序，这种方法哲学上称为归约论，它认为只有理解一个事物的每个组成部分后才能很好地理解该事物。要成为一个优秀的程序员，就要掌握 C 语言中不同语句的作用和用法，因为调试程序时，经常要逐句地检查程序，尤其是查找那些使程序不能正常运行的逻辑错误。但这种逐一语句仔细观察分析的方式并不是检查或阅读程序的唯一方法。有时回过头从整体上检查或阅读程序会更有帮助。事实上例 1.4.2 程序的功能可以用一句话来归纳，即“它将两个数相加并显示结果”。这种从更全局性的角度分析一个程序，主要考虑它作为一个整体是如何工作的方法，哲学上称为整体论，它认为整体并非每一部分的简单的相加。这种从整体的角度观察程序可以从另一个角度分析程序，这对成功的程序设计来说也是至关重要的。在学习编程时，必须学会从这两种角度出发去分析程序。如果过分关注细节，则只见树木不见森林。相反，如果只注意大的方面，则不能理解解决问题需要的工具。

在学习程序设计时，最好的方法是交替使用这两种视角。整体论有助于从整体上把握程序的作用，使你对程序设计过程的直觉更加敏锐，并能够从较高的层面去研究程序。另一方面，在实际编程时，则需要采用归约法，以理解程序是怎样结合在一起的。

更进一步,可以对它做一些修改以完成一些不同的功能。例如,可以修改程序使其能将3个数相加。认识现有模型的大体模式并利用它来构建新的程序是程序设计的基本策略。

1.5 算法

1.5.1 算法的概念

算法泛指解决某一个问题的方法和步骤。事实上,做任何事情都有一定的方法和步骤,例如洗手,你要首先打开水龙头,把手淋湿,然后用肥皂抹手,最后再伸手冲水把肥皂洗净,并关上水龙头。这些步骤都是按一定的顺序进行的,缺一不可,次序错了也不行。因此,从事各种工作和活动,都必须事先想好进行的步骤,然后按部就班地进行,才能避免产生错乱。简单地说,算法是一种解决问题的策略。是人们对问题进行分析和抽象的结果。算法是解决"做什么"和"怎么做"的问题。无论是数学公式,还是计算机程序,都属于算法的具体表现。但算法不等于程序,其含义和范围更广。算法是程序设计的灵魂。不了解算法就谈不上程序设计,因此对于程序设计人员来说,必须会设计算法,并且能根据算法编写程序。需要说明的是,不要认为只有"计算"的问题才有算法。下面通过两个简单的问题说明设计算法的思维方法。

例 1.5.1 求6!

解 可以用小学生的方法进行。

步骤1:先求1×2,得到结果2。

步骤2:将步骤1得到的乘积2再乘以3,得到结果6。

步骤3:将6再乘以4,得24。

步骤4:将24再乘以5,得120。

步骤5:将120再乘以6,得720。这就是最后的结果。

这一算法虽然正确,但如果要求用它求100!,则要写99个步骤,太繁了,因此不可取。而且每次都直接使用上一步骤的数值结果(如2,6,24等)也不方便,应当找到一种通用的表示方法。事实上,可以设两个变量,一个变量代表被乘数,一个变量代表乘数。而乘积结果不再设定新变量存放,而是直接将每一步骤的乘积放在被乘数变量中。如设p为被乘数,i为乘数。用循环算法来求6!的结果,可以将算法改写如下:

S1:使 p=1;

S2:使 i=2;

S3:使 p×i,乘积仍放在变量p中,可表示为 p×i=>p;

S4:使i的值加1,即 i+1 => i;

S5:如果i不大于6,返回重新执行步骤S3以及其后的步骤S4和S5;否则,转到步骤S6;

S6:输出p的值,算法结束。

最后得到p的值就是6!的值。上面的S1,S2…代表步骤1,步骤2……S是step(步)