第3章 结构化程序设计

程序设计是一个复杂而精细的过程。早期的程序设计是自由的、毫无规律的、技巧性很强的个性化活动，当时编出来的软件晦涩难懂，不易维护，可靠性差，质量低下，由软件错误而引起的信息丢失、系统报废事件屡有发生，这种现象导致了20世纪60年代末爆发的软件危机。从那时开始，人们开始反思程序设计本身的规律和方法，并着手对软件开发方法和软件生产管理方面的研究，寻求以最少的时间和最小的代价获得较高质量的软件产品。模块化和结构化程序设计方法就是在这种背景下产生的。**结构化程序设计**方法是最早的程序设计方法之一，对于某个客观问题，在进行结构化程序设计时，人们首先是从整体的角度来考虑问题，将其分割成多个逻辑上相互独立的部分，分别一一实现，最后再按照某种方法将这些部分组装起来。用结构化程序设计方法得到的程序不仅结构良好，清晰易读，易写，而且易维护，易排错，易于验证正确性。它在一定程度上缓解了软件危机，改善了软件开发的状况。更重要的是，它向人们揭示了研究程序设计方法的重要性，并为后来的程序设计方法奠定了基础。初学者应及早学习和掌握这一方法。

本章主要介绍结构化程序设计的思想方法、三种基本结构、C语句的概念和输入输出函数的使用方法。

3.1 结构化程序设计方法

在用计算机处理问题时，编写程序只是其中一个步骤，而算法设计是整个程序设计的核心。如果算法不当，程序编写技艺再高也得不到正确的结果，而不同的算法对程序运行的效率和结果的精度会产生极大的影响，程序的质量主要是由算法决定的。

什么样的算法是好的呢？判断程序好坏的标准又是什么呢？

在计算机发展的初期，计算机的内存小，运算速度慢。程序设计追求代码短小、精练、运算速度快，即效率第一，采用的是手工方式，全凭编程者的个人技巧和爱好，使用了大量 goto 语句，而这往往导致程序晦涩难懂，也难以检查。随着计算机的飞速发展，机器的速度越来越快，内存容量越来越大，对效率的苛求有所缓解。更何况程序的效率主要是由算法来确定的，编程的技巧并不能对程序的效率产生决定性的影响。此外，程序的规模越来越大，软件开发采用的是团队化、大规模的生产方式。随之而来的是出错的可能性大了，出错所带来的后果也越来越严重，甚至是灾难性的。这时，判断程序好坏的标准就从效率第一变成要求程序有良好的可读性，以提高程序设计的质量和便于查错和维护，减少软件成本。即把程序的可靠性与可维护性摆在了首要位置。

为了从根本上保证程序的正确与可靠，要求软件人员改变按个人习惯与爱好编制程序的个体手工方式，1968 年，著名计算机科学家 E. W. Dijkstra 指出了程序设计中过去常用的 goto 语句的三大危害，反对滥用 goto 语句，代之以软件生产方式的科学化、规范化、工程化，并由此产生了结构化程序设计方法和“软件工程”概念。

结构化程序设计方法是一整套指导软件开发的方法，涵盖了系统分析、系统设计和程序设计三方面的内容。结构化的程序设计就是采用自顶向下、逐步细化、模块化的方法进行程序设计。它强调程序设计风格和程序结构的规范化，提倡清晰的程序结构，其基本思路是把一个复杂问题的求解过程分阶段进行，每个阶段处理的问题都控制在人们容易理解和处理的范围内。具体实现步骤如下：

(1) 按自顶向下逐步细化的方法对问题进行分析、设计；

(2) 系统的模块设计；

(3) 结构化编码。

3.1.1 自顶向下分析设计问题

由于人的思维能力有限，人们对问题规模的驾驭能力就受到限制，结构化程序设计方法是解决人脑思维能力的局限性与所处理问题的复杂性之间矛盾的一个有效办法。自顶向下、逐步细化的结构化的程序设计方法，可以把大的复杂问题分解成若干个小问题，然后各个击破。

自顶向下、逐步细化就是对一个复杂问题，首先进行上层(整体)的分析与设计，按其组织或功能将问题分解成若干个子问题，如果所有的子问题都得到了解决，整个问题就

解决了。而解决子问题无论在规模上还是在复杂性上都会大大低于原问题。如果子问题仍然十分复杂,再对它进一步分解,如此一层一层地分解下去,直到处理对象相对简单,容易处理为止。每一次分解都是对上一层进行细化,逐步求精,最终形成一种层次结构(类似树形),用于精确描述分析设计的结果。

例如,要想为图书馆开发一个图书馆管理系统,首先按其功能把整个管理分为 4 模块:图书登录、借书、还书和预约,一旦这 4 个功能都能实现,连接起来就形成了图书馆管理系统,但这 4 个模块还比较复杂,难于直接实现,可以进一步分解每一个模块。图 3.1.1 给出了图书馆管理系统设计的层次结构图。图中的每个方框都是程序设计中的模块,在 C 语言中用函数实现,相互之间的连接线就是函数之间的调用关系。

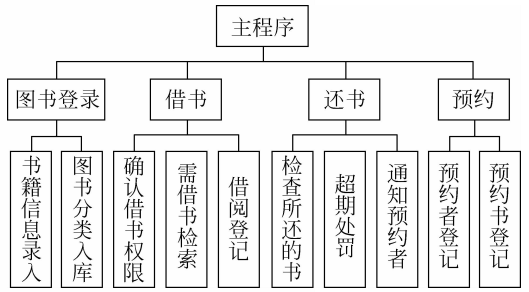

图 3.1.1　图书馆管理系统设计的层次结构

按照自顶向下方法设计系统,有助于后面模块的程序设计、各模块的调试验证以及系统最终按层次集成起来。

3.1.2　模块化程序设计

通过按自顶向下、逐步细化方法,可把复杂的问题分解成许多容易解决的小问题,如图 3.1.1 的每个小方框所示。每个小方框也就是系统中的一个模块。所以,每一个小问题可用一个程序模块实现,再把这些小程序模块像搭积木那样合成起来,形成解决整个复杂问题的大程序。因此,程序的模块化就是把程序划分成若干个模块,每个模块完成一个特定的功能。把这些模块综合起来组成一个整体,就可以完成指定问题的全部的功能要求。

此例表明,为了控制和简化复杂的软件结构,结构化程序设计将客观问题按照功能需求进行自顶向下的功能分解,并逐步求精,最后再将数据流图映射成模块化结构。这样构造的程序就由一组基本的相互独立的具有单入口单出口的程序模块(函数)组成,从而保证了程序功能评价的可靠性。

在设计某一个具体的程序模块时,程序模块中包含的语句一般不要超过 50 行,这样既便于程序员的思考与设计,也利于程序的阅读。一个模块应具有良好的独立性,使得程序模块的编写、调试都可以独自完成,尽量减少模块之间的相互影响,以免带来相互间的干扰。在 C 语言中,模块用函数来实现。一个模块对应一个函数,如果该模块功能复

杂，可以进一步调用低一层的模块函数，以实现结构化的程序设计思想。因此，在C语言中，应该把一个大程序编写为若干个函数集，每个函数的语句一般控制在20行左右，最多不宜超过50行语句。这不仅易于程序在屏幕上的显示，使人易于阅读理解。更重要的是功能简捷，易于编程和调试，因而有助于提高软件的可靠性。程序的模块化的另一个好处是：它也有助于软件开发工程的组织管理，一个复杂的大型程序可以由许多程序员分工编写不同的模块，以加快软件开发的速度，缩短开发周期，提高开发质量。

3.1.3 结构化程序编写

当一个软件经模块化设计后，每一个模块可以独立编程。业已证明，任何只含有一个入口和一个出口的程序均可由顺序结构、选择结构和循环结构组成。不仅程序本身是单入口单出口的，而且程序的每一局部结构也应是单入口单出口。从结构上讲，进入顺序结构、选择结构和循环结构是单入口的，当执行完结构离开时，也必定是单出口的。因此在结构化程序设计时，要严格采用3种基本程序结构，编程时尽量少用或不使用goto语句，因为它会破坏程序整体的结构性，使程序结构变得复杂，降低了程序的可读性。C语言提供了丰富的语句来实现这3种程序结构，对于编程者来说，应首先确定适合3种基本结构的程序流程图，然后使用若干相应语句组成函数。在C语言中，函数的重要性是不言而喻的。作为构成模块化结构的最小单位，函数可以当做独立的单元来处理。可以仅定义一次函数的实现，就可以多次使用它，并且多个函数还可以组织在一起构成所谓的函数库。第6章将会对函数作详细的讨论。

3.2 语句的概念

语句是C程序中的基本功能单元，程序中的任何一个语句都意味着为完成某一任务而进行的某些处理动作。通常简单的语句表示一种单一功能的操作，而复合语句、选择语句和循环语句等复杂语句可能代表多个操作组成的一种复杂操作或功能。语句的意义称为该语句的语义。虽然初学者阅读程序时可能会误解某些语句的语义，但一个正确运行的编译器只会以一种确定的方法解释语句。组成C程序的各种语句主要可分为4类：表达式语句、控制语句、复合语句和空语句等。书写简单C语句时通常每个编辑行写一条语句；但在相邻C语句都较短时，还可以在一行写多条语句，此时应注意不要遗漏语句的结束标志“；”，例如，x=6；y=8；另外，当语句比较长时，为了书写清晰和编辑方便也可以一条语句占多行。

3.2.1 表达式语句

在C语言中，只要在任何表达式的结尾加上分号“；”就构成了**表达式语句**，它是程

序中要求计算机对数据执行一些基本操作和处理的语句,最典型、最常用的表达式语句是赋值语句和函数调用语句等,例如

```
i = 1,j = 2,k = 3;
j + k;
y = 5 * sqrt(27.0) + 3;
i ++ ;
fun(j,j + k,6);
```

注意:尽管在C中任何表达式后跟分号都是一个合法语句,但并不是这样组合肯定会产生有效的语句。例如,上面的语句:

```
j + k;
```

是合法的语句,但它是一个无效语句,因为它对计算结果没有进行任何操作。该语句将变量j和k的值相加,然后将结果丢弃。

在C程序中,大多数语句是表达式语句,所以有人把C语言称作"表达式语言"。如下面程序行:

```
printf("This is a C program.\n");
```

就是一个合法的表达式语句。一般情况下,顺序书写的表达式语句将按它们的书写次序依次执行。

3.2.2 控制语句

如果没有任何转向指示,C程序的语句将按照出现顺序执行一次。然而,对于大多数应用来说,只有自上而下的程序顺序显然是不够的。现实问题的解决策略往往要求在多个动作中选择一个或能重复执行一系列步骤。**控制语句**就是程序中用来控制语句执行次序的语句,因此,它能够影响其他语句的执行的次序。总体上来说,C程序是按照主函数中语句的书写顺序执行,而控制语句可以改变这种执行顺序,使程序中语句的执行顺序与书写顺序不一致。C语言提供了9种控制语句,它们又可分为3类。

(1) 选择语句:如if(条件)~else~(双分支选择语句)、switch(多分支选择语句)等。if语句依靠条件测试的结果选择执行不同的语句。

(2) 循环语句:如while(条件)~、do~while(条件)、for()~等。do…while语句允许重复执行多条语句直到满足特定的条件为止,而for语句能根据指定的次数重复执行某语句。

(3) 转向语句:break语句(中止执行switch或循环语句)、continue语句(结束本次循环语句)、goto语句(无条件转向语句)和return语句(从函数返回语句)等。

为了充分发挥这些语句的作用,在今后的学习中应着重理解它们工作机制的更多细节问题,以及它们作为问题求解过程的一部分来应用的方法。

3.2.3 复合语句

复合语句(也称为“程序块”或块语句)是由大括号{}把一些语句括起来组成。例如,{temp=x; x=y; y=temp;}就是一个复合语句。

复合语句在语法上相当于一个语句,只是在大括号外,不再写分号。因此,在C程序中,当需要把若干条语句作为一条语句使用时,可以使用复合语句。并且,复合语句作为一个语句又可以出现在其他复合语句的内部,即复合语句是可以嵌套的。复合语句在编程中主要用于以下两种情形:

(1) 语法要求用一条语句,但又难以用一条简单语句表达清楚的情形,如它经常作为for语句、while语句中的循环体以及if语句的内嵌语句。

(2) 形成局部化的封装体。在C程序中,不仅仅是函数,只要是块语句形式,都是局部化的封装体。如块语句中定义的变量只在本块范围内可用,在第6章将介绍这一点。

【例3.2.1】 复合语句的嵌套。

```
void main()
{
   int x = 1, y = 10;
  {         int x;
             x = 5;
             {     int a;
                    a = y; y = 20;
                    printf("a = %d,x = %d\n",a,x);
             }
  }
  printf("x = %d,y = %d\n",x,y);
}
```

注意:复合语句内的各条语句都必须以分号“;”结尾,但在复合语句结束标志右大括号}后面则不能加分号。C编译器在遇到复合语句时,会将它看做一个简单语句。

3.2.4 空语句

没有表达式的语句称为**空语句**,即它是只有一个分号“;”的语句。事实上,它也可看成是一个特殊的表达式语句,但不做任何事情。其作用是用于语法上需要一条语句的地方,而该地方又不需做任何事情。空语句有时用来作为循环语句中的循环体,例如,

```
for(i = 1; i <= 100000;i ++ );
```

这样的循环语句用于程序中需要“耗时”的地方。

【例3.2.2】 表达式语句应用的示例程序。

```
#include <stdio.h>
main()
```

```
{  int  a,b,i=5;  //变量的定义
   a=i+3;         //赋值表达式 a=i+3 后面加一个分号
   i--;           //由自减运算符组成的表达式 i-- 后面加一个分号
   b=2,++b;       //逗号表达式 b=2,++b 后面加一个分号
   i+1;           //算术表达式 i+1 后面加一个分号
   printf("a=%d,b=%d,i=%d\n",a,b,i);
}
```

运行结果：

```
a=8,b=3,i=4
```

本程序中的所有语句都是在表达式后面加上分号而成，即它们都是表达式语句。但不是任何表达式语句在C语言中都有作用，例如，语句i+1；是合法的，但是它并不把i+1的值赋给另一变量，所以它在程序中毫无实际意义。

3.3 程序的3种基本结构

用计算机解题，要按确定的步骤进行，这些步骤的执行顺序则要通过程序的控制结构来实现。1966年计算机科学家Bohn和Jacopini研究证实，所有的程序(不论是简单还是复杂程序)都能够由顺序结构、选择结构和循环结构这三种控制结构组合而成来实现。1972年Mills又进一步提出了每个控制结构只应该有一个入口、一个出口的原则；无论一个程序包含多少个模块，也不论一个模块包含多少个基本控制结构，整个程序仍能保持一条清晰的线索。因此，这三种结构就被称为结构化程序设计的三种基本结构。也是结构化程序设计必须采用的结构，因为使用这三种结构编写出来的程序清晰可读又便于理解。而C语言是一门非常优秀的结构化程序设计语言。它提供了实现这三种基本结构的手段，如复合语句、选择语句、循环语句等。

3.3.1 顺序结构

物理上相邻的多条语句在执行时，按照其物理排列的前后顺序一条一条地顺序执行，这样聚集在一起的语句形成了一种**顺序结构**。由于这些语句在物理上顺序排列并紧邻，必要时可以使用复合语句将它们作为程序中的一个程序块。顺序结构中的每一条语句都将执行一次，而且只能执行一次。顺序结构是C语言的基本结构，也是在程序设计中最简单的一种结构。在C语言的4类语句中，除了控制语句外，其余3类属于顺序执行语句。

【例3.3.1】 顺序结构程序示例：从键盘输入四个浮点型数并分别存入变量a、b、c和d中，求它们的平均值。

解 程序如下：

```
#include<stdio.h>
main()
{ float a,b,c,d,ave;
```

```
    printf("Input a,b,c,d:\n");
    scanf("%f%f%f%f",&a,&b,&c,&d);
    ave=(a+b+c+d)/4;
    printf("ave=%f\n", ave);
}
```

运行结果：

```
input a,b,c,d:
1.3 2.6 3.8 6.6〈回车〉
ave=3.575000
```

在本例中，程序的执行过程是从主函数体内的第一条语句开始，由上到下按顺序逐条执行，其执行过程可用如图 3.3.1 的 N-S 流程图表示。其中语句“scanf("%f%f%f%f", &a, &b, &c, &d);”的作用是从键盘输入 4 个数分别赋值给 a、b、c、d，输入数据的具体方法将在 3.5 节中介绍。

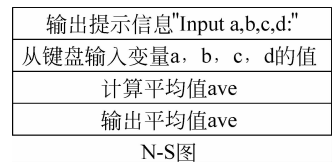

图 3.3.1　例 3.3.1 程序顺序执行过程

3.3.2　选择结构（分支结构）

流程不按照语句在程序中出现的先后顺序逐条执行，而是根据判断项的值有条件地选择部分语句执行，称这样的程序结构为**选择结构**（或分支结构）。C 语言中支持选择这种控制机制的语句称为选择语句，主要有两种：if 语句（条件语句）和 switch 语句（开关语句）。选择语句将若干语句序列联系在一起形成选择结构，每条语句作为该结构的一个运行分支，程序运行至该结构入口时将根据所满足的条件，在若干分支中选择一个分支加以执行。选择结构程序设计将在第 4 章中详细介绍。

3.3.3　循环结构

根据需要反复执行程序中的某些语句，这样的程序结构称为**循环结构**。C 语言专为循环结构提供了 3 种形式的循环语句：while 语句、do…while 语句、for 语句。循环语句将在第 5 章中有详细的介绍。

3.4　赋值语句

在赋值表达式的尾部加上一个分号可构成**赋值语句**，即

```
变量=表达式;
```

其中，赋值号“=”左边可以是任何变量名，“=”右边可以是任何表达式。例如，x=x+5 是赋值表达式，而 x=x+5; 则是赋值语句。因此，要写一个赋值语句，必须以一个变量

名开始,然后依次是赋值号“=”、表达式、分号。赋值语句与赋值表达式是C语言程序设计中最基本的、最常用的一种语句与表达式,其中赋值语句兼有表达式计算和赋值的双重功能。即将赋值运算符“=”右边表达式的值赋给“=”左边的目标变量。

【例 3.4.1】 交换变量x和y的值。

解 分析:两个人交换座位,只要各自去坐对方的位置即可,这种交换是**直接交换**。而要想将一杯水和一杯可乐互换,就不能直接从一个杯子倒入另一个杯子,必须借助于一个空杯子,先把水倒入空杯,再将可乐倒入已倒空的水杯,最后把水倒入已倒空的可乐杯,这样才能实现水和可乐的交换,这是**间接交换**。由于计算机内存有“取之不尽、一冲就走”的特点,因此程序设计中交换两个变量的值只能采用借助于第三个变量间接交换的方法,如图3.4.1所示。

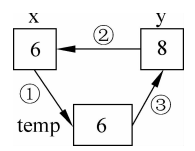

图3.4.1 相互交换两个变量值的过程

据上述间接交换原理,交换变量x和y的值的程序如下:

```
#include <stdio.h>
main()
{ int  x=6,y=8,temp;
  printf("x=%d,y=%d\n",x,y);
  temp=x;          //将x的初值赋予变量temp

  x=y;             //仅改变变量x的值,y的值不变
  y=temp;          //变量y被赋予新的值,原值被覆盖
  printf("x=%d,y=%d\n", x,y);
}
```

运行结果:

```
x=6,y=8
x=8,y=6
```

请思考:起交换作用的三条语句“temp=x; x=y; y=temp;”能否用“x=y; y=x;”两条语句代替?

此例告诉我们:①每个变量只能存放一个值。一旦对变量进行赋值,则变量将保存该值,直到该变量被赋予新的变量值为止。②若将一个变量的值赋予另一个变量,该变量的值不会消失。

【例 3.4.2】 输入一个4位整数,然后打印出它的4位数字的和。

```
#include <stdio.h>
main ()
{  int  n,a,b,c,d,sum;
   printf("Input n:\n");
   scanf ( "%d", &n );
   a=n%10;          //求个位数
   b=n/10%10;       //求十位数
```

```
    c = n/100 % 10;    //求百位数
    d = n/1000;        //求千位数
    sum = a + b + c + d;
    printf("n = %d,sum = %d\n",n,sum);
}
```

运行结果：

```
Input n:2568 <回车>
n = 2568,sum = 21
```

请思考：是否还有其他方法求得任一整数的各位数字？

以上两个例子涉及了交换两个数据和求某整数各位数字的算法，这些算法在程序设计中经常使用。需要注意的是，解决某一问题的算法并不唯一，但有些算法效率高，易于理解，而有些算法则不然。所以拿到一项较大的任务后，不要急于编写程序，应遵循“分析问题——设计算法——画流程图”的顺序做好准备工作，然后开始编写程序。程序设计的一般过程包括问题的分析、算法的设计、流程的描述、程序的编写、调试与运行、文档编制等步骤。由于本书重点介绍用 C 语言设计程序的方法，所举的例子程序较短，算法也简单，因此没有严格按照上述步骤进行。

3.5 输入输出函数

在用计算机解决实际问题时，一般都伴随有一些初始数据的输入以及处理结果的输出，因此，每个程序一般总可以由 3 部分组成：输入初始数据、计算处理和输出结果。数据的输入与输出是程序与用户之间的交互界面。为了实现输入输出功能，在 C 的库函数中提供了数据的输入输出函数，而其中函数 printf 和 scanf 是最能体现 C 语言的特征。

3.5.1 格式输出函数 printf

要想正确地输出 C 语言的各种类型的数据，必须精通函数 printf 提供的功能。printf 函数的调用形式如下：

```
printf("格式控制字符串",表达式 1,表达式 2,…,表达式 n);
```

作为参数传递的表达式的个数 n 取决于需要输出的数据的个数，特别当 n＝0 时，调用形式如下：

```
printf("格式控制字符串");
```

即调用 printf 时可以没有输出表达式。

功能：按格式控制字符串中的格式依次输出表达式 1、表达式 2、…、表达式 n 的值。

说明：格式控制字符串是用双引号括起来的字符串，用于说明输出数据所采用的格式，它由两种成分组成：普通字符和格式控制符。printf 将“格式控制字符串”中的普通