第3章

简单C程序设计

C语言是结构化的程序设计语言，非常适合编写结构化的程序。

结构化的程序通常包括数据的描述和操作的描述两方面的内容。数据的描述是指程序中数据的类型和数据的组织形式，即数据结构。前面介绍的数据类型、常量、变量及后续章节的数组、结构体等都属于这方面的内容。操作的描述是指程序中对数据的操作方法和操作步骤，也就是算法。数据的描述和操作的描述是程序设计过程必不可少的组成部分，数据是操作的对象，操作的目的是对数据进行加工处理，以得到所期望的结果。著名的瑞士计算机科学家N.沃思提出了“数据结构＋算法＝程序”。实际上，在程序设计中，除了考虑数据和操作的描述外，还应当确定程序设计方法和语言环境。

本章将介绍算法的概念及描述工具，概述C语言的语句，重点介绍C语言中如何实现基本的输入输出操作，以及如何编写简单的顺序结构程序。

3.1 算法

3.1.1 算法的概念

广义上讲，算法是指解决问题的方法和步骤。

【例3.1】 一个人带3只狼和3只羚羊过河，只有一条船，同船可以容纳一个人和两只动物。没有人在的时候，如果狼的数量不少于羚羊的数量，狼就会吃掉羚羊，要保证狼不吃掉羚羊。

过河的算法可以设计如下：

步骤1：人带两只狼过河；

步骤2：人自己返回；

步骤3：人带一只羚羊过河；

步骤4：人带两只狼返回；

步骤5：人带两只羚羊过河；

步骤6：人自己返回；

步骤7：人带两只狼过河；

步骤8：人自己返回带一只狼过河。

狭义上，算法指的是计算机算法，即对特定问题求解步骤的一种描述，它是计算机指令的有限序列，其中每一条指令表示计算机可以进行的一个或多个操作。人们利用计算机编写程序处理各种不同的问题，必须先对各类问题进行分析，确定解决问题的具体方法和步骤，即算法，然后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。算法是程序设计的关键之一，程序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。

计算机算法可以分为数值运算算法和非数值运算算法。数值运算算法的目的是求解数值，如求方程的根、求函数的定积分等。非数值运算算法主要用于事务管理领域，如排序、查找、调度等。

3.1.2 算法的特性

1. 有穷性

一个算法（对任何合法的输入值）必须在执行有限步骤之后结束，且每一步都可在有限的时间内完成。这里的有限的概念不是纯数学的，而是在实际上是合理的，可以接受的。

2. 确定性

算法中每一条指令必须有确切的含义，不会产生二义性。在任何条件下，算法只有唯一的一条执行路径。

3. 有效性

一个算法中的每一步都能有效地执行，并能得到确定的结果。

4. 输入

一个算法有零个或多个输入。数据是程序处理的对象，如果算法中的数据是程序自带的，而不是来自计算机外部，可以没有输入操作；否则，算法必须包含有输入操作。

5. 输出

一个算法有一个或多个输出。通过输出可以了解算法执行的情况和执行的结果。

3.1.3 算法设计的要求

通常设计一个“好”的算法，应考虑达到以下目标。

1. 正确性

算法应当满足具体问题的需求。

2. 可读性

良好的可读性有助于人对算法的理解。

3. 健壮性

当输入数据非法时，算法也能适当地做出反应或进行处理，而不会产生莫名其妙的输出结果。

4. 效率与低存储量需求

效率指的是算法执行时间。对于同一个问题如果有多个算法可以解决，执行时间短的算法效率高。低存储量需求指算法执行过程中所需要的最大存储空间。

下面给出两个简单的算法举例：

【例 3.2】 求 5!。

原始的算法如下：

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。

这样的算法虽然正确，但太烦琐。如果要求 1×2×…×10 000，则要写 9999 个步骤，显然是不可取的。

对于算法的改进，可以设定一个变量表示乘数，一个变量表示被乘数，并且将每一步的乘积放在被乘数变量中。现设 t 为被乘数，i 为乘数。用循环来设计改进的算法如下：

步骤 1：使 t=1；

步骤 2：使 i=2；

步骤 3：使 t×i，乘积仍然放在变量 t 中，可表示为 t×i→t；

步骤 4：使 i 的值+1，即 i+1→i；

步骤 5：如果 i≤5，返回重新执行步骤 3 以及其后的步骤 4 和步骤 5；否则，算法结束。

【例 3.3】 求 2×4×6×8×10。

本问题只需对例 3.2 的算法作很少的改动，算法如下：

步骤 1：使 t=2；

步骤 2：使 i=4；

步骤 3：使 t×i，乘积仍然放在变量 t 中，可表示为 t×i→t；

步骤 4：使 i 的值+2，即 i+2→i；

步骤 5：如果 i≤10，返回重新执行步骤 3 以及其后的步骤 4 和步骤 5；否则，算法结束。

3.1.4 算法的描述

算法的描述方法有多种，最常用的有自然语言、伪代码、流程图、N-S 图、PAD 图和计算机语言等。本书不介绍伪代码和 PAD 图，下面简单介绍自然语言、流程图、N-S 图和计算机语言如何描述算法。

1. 自然语言

自然语言是人们日常所用的语言，如汉语、英语、德语。使用这些语言不用专门训练，所描述的算法也通俗易懂。

例如求 3 个数中最大值问题，先将 a，b 两个数进行比较，找出其中的较大数，然后再把它和第 3 个数 c 进行比较，如果它比 c 大，则它就是最大数；否则 c 是最大数。

但是用自然语言描述法也存在明显的缺点：由于自然语言的歧义性，容易导致算法执行的不确定性。比如张三对李四说他的孩子考上了大学，到底谁的孩子考上了大学？此外，自然语言的语句一般太长，从而导致了用自然语言描述的算法太长；由于自然语言表示是按照步骤的标号顺序执行的，因此当一个算法中循环和分支较多时就很难清晰地表示出来；自然语言表示的算法不便翻译成计算机程序设计语言。因此，使用此种语言描述算法要求尽可能精确、详尽。

2. 流程图

(1) 流程图的概念。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。

(2) 流程图的符号。流程图是描述算法的常用工具。流程图采用美国国家标准化协会(American National Standard Institute，ANSI)规定的一组图形符号来表示算法，如图 3-1 所示。

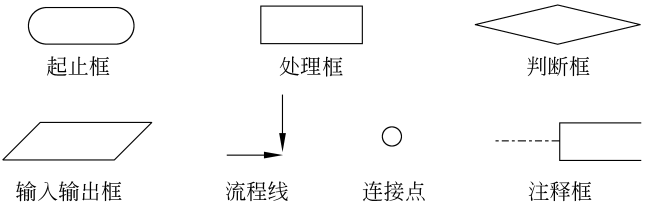

图 3-1　流程图符号

其中，起止框表示算法的开始或结束，处理框表示算法中计算与赋值，输入输出框表示算法中输入或输出，判断框表示算法中的条件判断，流程线表示算法中的流向，连接点表示算法中的转接，注释框表示对算法加注释。

(3) 3 种基本结构的流程图。早期的非结构化语言中都有随意跳转语句，允许程序从一个地方直接跳转到另一个地方。非结构化语言虽然使程序设计十分方便灵活，减少了人工复杂度，但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，排错更是十分困难。

人们经过研究，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 3 种基本结构组成。在构造一个算法时，仅以这 3 种基本结构为基础，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。整个算法都由 3 种基本结构组成，就像用模块构建的一样，结构清晰，易于正确性验证，易于纠错，这就是结构化方法。遵循这种方法的程序设计，就是结构化程序设计。结构化程序设计不允许有随意跳转语句。所以，只要规定好 3 种基本结构的流程图，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，按顺序执行。其流程图的基本形态如图 3-2 所示，语句的执行顺序如下：A→B→C。

② 选择(分支)结构。选择(分支)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内容。其基本形状有两种，如图 3-3 所示。图 3-3(a)的执行序列如下：当条件为真时执行 A，否则执行 B；图 3-3(b)的执行序列如下：当条件为真时执行 A，否则什么也不做。

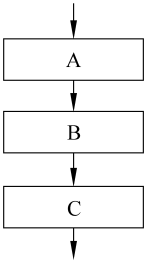

图 3-2　顺序结构流程图

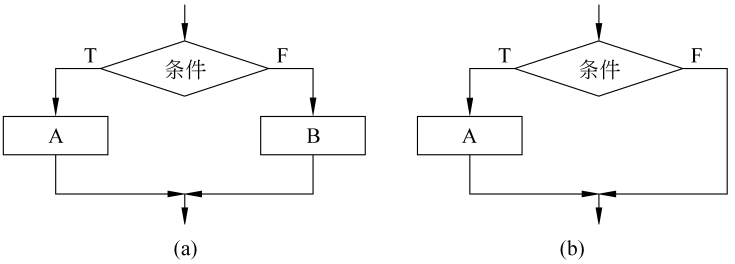

图 3-3　选择(分支)结构流程图

③ 循环结构。循环结构有两种基本形态：当型循环和直到型循环。

当型循环如图 3-4 所示。其执行序列如下：当条件为真时，重复执行 A；一旦条件为假，跳出循环，执行循环后的语句。

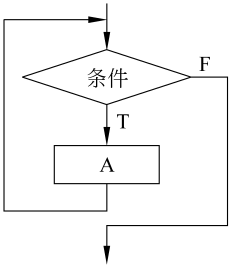

图 3-4　当型循环流程图

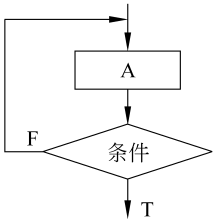

图 3-5　直到型循环流程图

直到型循环如图 3-5 所示。执行序列如下：首先执行 A，再判断条件，条件为假时，一直循环执行 A；一旦条件为真，结束循环，执行循环紧后的下一条语句。

在图 3-4、图 3-5 中，A 被称为循环体，条件被称为循环控制条件。

将例 3.2 的算法使用流程图表示如图 3-6 所示。

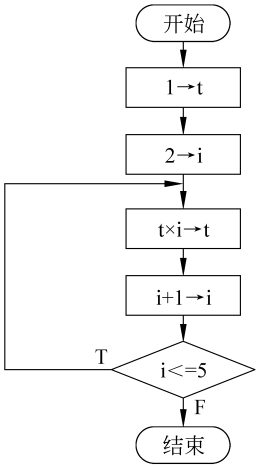

图 3-6　例 3.2 的算法流程图

3. N-S 图

N-S 图是由 I. Nassi 和 B. Shneiderman 共同提出的一种结构化描述方法。在 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。

(1) 顺序结构。顺序结构的 N-S 图如图 3-7 所示。

(2) 选择结构。选择结构的 N-S 图如图 3-8 所示。

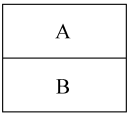

图 3-7　顺序结构

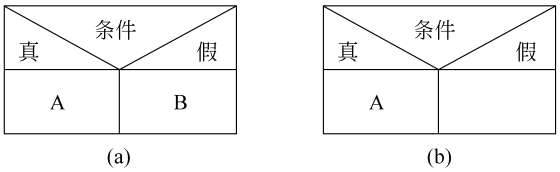

图 3-8　选择结构

(3) 循环结构。当型循环的 N-S 图如图 3-9 所示。直到型循环的 N-S 图如图 3-10 所示。

图 3-9　当型循环

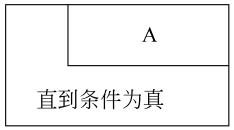

图 3-10　直到型循环

4. 程序设计语言

对于采用自然语言、流程图和 N-S 图描述的算法，计算机是不能执行的。要让计算机执行一个算法，必须把该算法转换成计算机语言。

用计算机语言表示算法必须严格遵循所用语言的语法规则。例 3.2 的算法使用 C 语言表示如例 3.4 所示。

【例 3.4】 编程序求 5!。

```
#include <stdio.h>
void main()
{
    int i,p;
    p=1;
    i=2;
    while(i<=5)
    {
        p=p*i;
        i=i+1;
    }
    printf("%d",p);
}
```

3.2　C 语句概述

C 的语句分为表达式语句、函数调用语句、控制语句、复合语句和空语句 5 类。

1. 表达式语句

表达式语句由表达式加上分号(;)组成。其一般形式如下：

表达式;

执行表达式语句就是计算表达式的值。

例如：

```
x=y+z;          /*赋值表达式语句*/
y+z;            /*加法运算语句,但计算结果不能保留,无实际意义*/
i++;            /*自增1语句,i值增1*/
```

赋值语句是由赋值表达式加上分号构成的表达式语句。其一般形式如下：

```
变量=表达式;
```

赋值语句的功能和特点都与赋值表达式相同。它是程序中使用最多的语句之一。使用赋值语句时需要注意以下几点。

(1)在赋值符“=”右边的表达式也可以是一个赋值表达式。

例如：

```
a=b=c=d=e=5;
```

按照赋值运算符的右接合性,因此实际上等效于：

```
e=5;
d=e;
c=d;
b=c;
a=b;
```

(2) 注意赋值表达式和赋值语句的区别。赋值表达式是一种表达式,它可以出现在任何允许表达式出现的地方,而赋值语句则不能。

下述语句是合法的：

```
if((x=y+5)>0)z=x;
```

该语句的功能是：若表达式 x=y+5 大于 0,则 z=x。

下述语句是非法的：

```
if((x=y+5;)>0)z=x;
```

因为 x=y+5;是语句,不能出现在表达式中。

2. 函数调用语句

函数调用语句由函数名、参数加上分号组成。其一般形式如下：

```
函数名(参数表);
```

例如：

```
printf("C Program");      /*调用库函数,输出字符串*/
```

3. 控制语句

控制语句用于控制程序的流程,以实现程序的各种结构方式。C 语言的控制语句可分

成条件语句、循环语句和转向语句3类，共9种。这将在后续章节详细介绍。

4. 复合语句

把多个语句用大括号({})括起来组成的语句称为复合语句。在语法上，复合语句相当于单条语句，而不是多条语句。其一般形式：

```
{语句组}
```

例如：

```
{
  x=y+z;
  a=b+c;
  printf("%d%d",x,a);
}
```

上述是一条复合语句。复合语句内的各条语句都必须以分号结尾，在大括号({})外不能加分号。

5. 空语句

只有分号(;)组成的语句称为空语句。空语句在语法上占用一个语句位置，但是不具备任何操作功能。在程序中空语句可用来作空循环体。

例如：

```
#include <stdio.h>
void main()
{
    while(getchar()!='\n')
    ;
}
```

这里的循环体为空语句

```
;
```

没有该空语句程序就是错误的。

3.3 C语言的输入输出

3.3.1 输入输出概述

1. 输入输出的概念

计算机由运算器、控制器、存储设备、输入设备和输出设备五大部件组成，运算器、控制器和存储设备构成了计算机的主机。在C语言中，输入输出是以计算机主机为主体而言的。从计算机向显示屏、打印机、磁盘等外部输出设备输出数据称为输出；从键盘、磁盘、光盘、扫描仪等输入设备输入数据称为输入。

2. C 语言中输入输出的实现

C 语言的输入输出操作是由函数来实现的，C 语言本身不提供输入输出语句。在 C 标准函数库中提供了包括 printf 函数和 scanf 函数等输入输出函数。在使用 printf 函数和 scanf 函数等库函数时，千万不要误认为它们是 C 语言提供的输入输出语句，printf 和 scanf 只是函数的名字，不是 C 语言的关键字。

C 编译系统提供的函数以文件库的形式存放在系统中，它们不是 C 语言本身的组成部分。为了使 C 语言的编译系统简单，C 语言不把输入输出作为语句。没有输入输出语句可以避免在编译阶段处理与硬件有关的问题，使编译系统简化，而且通用性强，可移植性好，对各种型号的计算机都适用，便于在各种计算机上实现。

各种版本的 C 语言函数库是各计算机厂商或软件开发公司针对某一类型计算机编写的，并且已编译成目标文件(.obj 文件)。它们在连接阶段与目标文件相连接，生成一个可执行程序。源程序中的 printf 函数在编译时并没有翻译成目标指令，而是在执行阶段中调用已被连接的函数库中的 printf 函数。由于 C 编译系统与函数库是分别进行设计的，因此不同的计算机系统所提供函数的数量、名字和功能是不完全相同的。

3. C 语言标准输入输出函数的使用

各种计算机系统都提供了通用的函数，这些通用的函数已经成为各种计算机系统的标准函数。C 语言函数库中的标准输入输出函数，是以标准的输入输出设备作为对象的。

在使用 C 语言库函数时，要将有关的头文件包括到用户源文件中。在头文件中包含了与用到的函数有关的信息。在使用标准输入输出库函数时，要用到 stdio.h 文件。文件后缀 h 是 head 的缩写，#include 命令都放在程序的开头，因此这类文件被称为头文件。在调用标准输入输出库函数时，文件开头应有以下预编译命令：

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

或

```
#include "stdio.h"
```

其中，stdio.h 是 standard input & output 的缩写，它包含了与标准 I/O 库有关的变量定义和宏定义。

3.3.2 格式输出函数 printf

printf 函数称为格式输出函数，其关键字最末一个字母 f 即为格式(format)之意。在前面的例题中已多次使用过这个函数。

1. printf 函数调用的一般形式

printf 函数调用的一般形式如下：

```
printf("格式控制字符串",输出表列)
```

该函数的功能是按照指定格式，向终端(本书均设定输出设备是显示器)输出若干个指

定类型的数据。

2. printf 函数的参数

printf 函数有两类参数，一类是格式控制字符串，一类是输出表列。

(1) 格式控制字符串。格式控制字符串用于指定输出格式，在使用时必须包含在" "中。格式控制字符串可由格式字符串和非格式字符串两种组成。

① 格式字符串。格式字符串是以%开头的字符串，在%后面跟有各种格式字符，以说明输出数据的类型、形式、长度、小数位数等。如%d 表示按十进制整型输出，%ld 表示按十进制长整型输出，%c 表示按字符型输出等。

② 非格式字符串。非格式字符串在输出时按原样输出，在显示中起提示作用。

(2) 输出表列。输出表列中给出了格式控制字符串中的格式字符串对应的各个输出项，格式字符串和各输出项在数量和类型上应该一一对应。

printf 函数按格式控制字符串的格式输出信息，输出时逐个考察函数中格式控制字符串的每个字符，如果是普通字符，就把它原封不动地输出到显示器上；如果是格式字符，就在输出表列中从左到右找到对应的数据项，按格式字符指定的类型和格式输出。

【例 3.5】 输出表列中包含非格式字符举例。

```
#include <stdio.h>
void main()
{
    int a=88,b=89;
    printf("%d %d\n",a,b);
    printf("%d,%d\n",a,b);
    printf("%c,%c\n",a,b);
    printf("a=%d,b=%d",a,b);
}
```

程序运行结果：

```
88 89
88,89
X,Y
a=88,b=89
```

本例中 4 次输出了 a、b 的值，但由于格式控制字符串不同，输出的结果也不相同。

(1) printf("%d %d\n",a,b);的格式控制字符串中，两格式串%d 之间加了一个空格(非格式字符)，所以输出的 a、b 值之间有一个空格。

(2) printf("%d,%d\n",a,b);的格式控制字符串中加入的是非格式字符逗号，因此输出的 a、b 值之间加了一个逗号。

(3) printf("%c,%c\n",a,b);的格式串要求按字符型输出 a、b 值。

(4) printf("a=%d,b=%d",a,b);为了提示输出结果又增加了非格式字符串。

3. printf 函数的格式字符串

在 C 语言中，格式输入输出函数对不同类型的数据必须采用不同的格式字符串。