笔者：额……这要看你想做什么了。即使把计算机放在你店里，如果你不清楚用它来做什么的话，那就等于没放。

小老板：摆上计算机后，它不能帮我做点什么吗？

笔者：嗯，如果只是摆上一台计算机的话，还真帮不上什么忙。因为计算机不是装饰品。

小老板：那让计算机和店里的客人聊天，咋样？

笔者：好，好。这个目的的话也是可以的。

小老板：那你能给我做个程序吗？

笔者：没问题，我正好带着笔记本呢，现在就给你做一个。这个就算你请我喝酒的回礼了，呵呵。

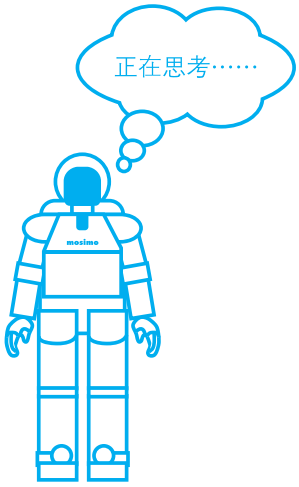
小老板：在这就可以啊。真了不起！

笔者：好了。让我们按下“对话”按钮看一下。

小老板：就是这个么？“欢迎光临”“您喝点什么”“让您久等了”“感谢您的光临”……就这些吗？只会这四句啊？

笔者：那么，你想要什么样的会话呢？

小老板：嗯……



笔者：啊，有了！

小老板：小点声！突然这么大声吓我一跳。

笔者：你刚才是不是在思考应该如何进行会话呢？对方说了什么之后，自己就要回答点什么。把这个顺序整理一下做成程序的话，就可以让计算机实现和人类一样的对话了。就像是计算机自己进行了思考一样。

小老板：啊，这样啊，明白了！不过，这个还真够麻烦的。必须要考虑上百、上千、上万种对话方式才行啊。

笔者：是啊。让计算机进行思考，确实是有点困难。

附录
让我们开始 C 语言之旅

本书涉及的程序示例，基本上都是用 C 语言编写的。但是那些完全没有编程经验的读者以及刚开始学习编程的读者，一下子看到这么多的 C 语言代码，肯定会感到很困惑吧。考虑到这一点，我们特意在本书的最后增加了补充章节，来对 C 语言的基本语法进行说明。

C 语言的特点

C 语言是 AT&T 贝尔实验室的 D. M. Ritchie 在 1973 年推出的程序开发语言。C 语言虽是高级编程语言，但它也具备了能够和汇编语言相媲美的低层处理（内存操作及位操作）功能。AT&T 贝尔实验室开发的 Unix，最初是用汇编语言编写的，但后来大部分都用 C 语言进行了重写。借助 C 语言，Unix 的移植性得到了大幅提升，进而使得更多类型的计算机开始应用 Unix 操作系统。此外，作为 Unix 系列操作系统之一的 Linux 也是用 C 语言来编写的。

即使在现在，C 语言也依然是常用的编程语言。我们知道，信息处理技术员职称考试中可以选择的编程语言有汇编语言、COBOL、C 语言、Java，从这一点就可以看出 C 语言的重要性。同时这也表明，在

当前的信息处理中，这四种语言是最常使用的。

在最新的 Web 编程中，Java、C# 等编程语言最有人气。Java 和 C# 都不是全新的编程语言，而是在对 C 语言语法进行了扩张的 C++ 的基础上发展而来的。因而，只要掌握了 C 语言，也就能很快掌握 Java 及 C#。另外，大部分的 C 语言编译器，都具有将 C 语言源代码转换成汇编语言源代码的功能，以及可以在 C 语言源代码中嵌入汇编语言的特点。

变量和函数

不管使用什么样的编程语言，程序内容都是由数据和处理构成的。至于程序的数据和处理具体该如何表示，则根据编程语言的不同而不同。在 C 语言中，数据用**变量**来表示，处理用**函数**来表示。因而，C 语言的程序就是由变量和函数构成的（图 A-1）。

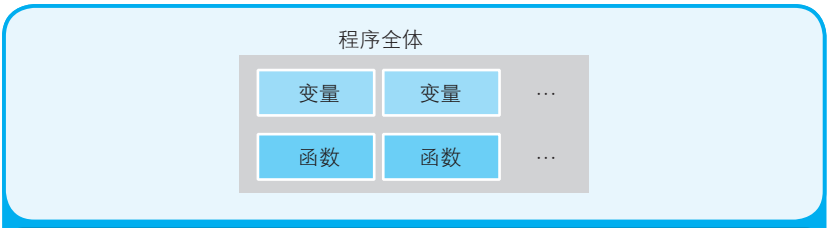


图 A-1　C 语言程序是由变量和函数构成的

看到变量和函数这些术语，大家估计会想到数学。数学的变量通常用 x、y、z 这些字母来表示。数学的函数则基本上都是像 $f(x)$ 这样，在函数名（这里是 f）后面加上括号，并在其中指定变量（这里是 x）。C 语言中变量和函数的描述方法，同数学是一样的。

不过，在 C 语言中，我们要从程序的角度来理解变量和函数，而

不能从数学的角度来理解。例如 x、y、z 这些变量，在数学中是“变化的数值”的意思，但在程序中表示的则是“**存放数值的地方**”。$f(x)$ 这个函数，在数学上表示的是“变量 x 这个参数决定了函数的结果”，但从程序上来看则是“**用 f 函数来处理 x 这个变量**”的意思。

在数学中，$y = f(x)$ 这一表现形式表示的是“y 是 x 的函数”的意思，但在程序中表示的则是“用 f 函数来处理变量 x，并将处理结果代入 y”。数学中的等号（=）表示的意思是“相等”，而程序中的等号表示的则是**赋值**的意思。在 C 语言中表示相等时，要用两个连续的等号。

数据类型

数学变量对位数和精度是没有任何限制的。与此相对，程序变量则受位数和精度的限制。这是因为，计算机的存储容量是有限的。计算机中预先被定义过的位数和精度称为**数据类型**。C 语言中主要的数据类型如表 A-1 所示。其中，Char、short、int 是整数用的数据类型。float 和 double 是小数用的数据类型。

表 A-1　C 语言中主要的数据类型

名　称	长度（位长）	精度（可以表示的 10 进制数）
char	8	– 128 ～ + 127
short	16	– 32 768 ～ + 32 767
int（或 long）	32	– 2 147 483 648 ～ + 2 147 483 647
float	32	-3.4×10^{38} ～ $+3.4 \times 10^{37}$
double	64	-1.7×10^{308} ～ $+1.7 \times 10^{307}$

在程序中使用变量（赋值、运算、显示等）时，需要同时对数据类型和变量名进行定义，如代码清单 A-1 所示。在 C 语言中，每个指令行的末尾都要用分号（;）来区分。// 后面是注释（对程序的说明）。$a = 123$; 部分表示的是给变量 a 代入数值 123，也就是对 a 进行赋值。在这部分

之前，需要对数据类型和变量名进行定义，这里使用了 int *a*;。通过对变量进行定义，就可以确保该变量对应的数据类型长度所需要的内存空间，并使用变量名来对内存空间进行读写。

代码清单 A-1　使用变量前需要先定义变量

```
int a;     // 定义 int 类型的变量 a
  :
a = 123;   // 为变量 a 赋值 123
```

标准函数库

函数包括程序员自己编写的函数以及系统提供的函数。其中，后者通常称为**标准函数库**。标准函数库是指具有可被各种程序使用的通用功能的函数。本书的示例程序中涉及到的 printf、scanf、rand 等都是标准函数库的一种。这些函数分别有“输出到显示器上显示”“从键盘输入信息”“产生随机数”等通用功能。

函数的括号中，除变量以外，也可以放置通过文字串、数值等指定的数据信息，这些统称为**参数**。被作为函数的处理结果而返回的数值称为**返回值**。利用函数称为函数调用。根据函数种类的不同，也有一些函数是不需要参数或没有返回值的。

把函数比喻成“工厂”的话可能更好理解。这样一来，参数就是被拉入工厂的“原材料”。在工厂中对原材料进行加工后，得到的“产品”就是返回值（图 A-2）。

在这一过程中，计算机的基本操作大体可以划分为“输入数据”“处理数据”“输出数据”三块。如果是一个简单的示例程序的话，想必应该会通过键盘来输入数据，并将处理结果输出到显示器上。因而，如果编写出来的程序包含了从键盘输入数据、对数据进行相应处理、

把结果输出到显示器上这一系列操作的话，就说明具备了 C 语言基础。如代码清单 A-2 所示。该示例是对键盘输入的两个数值求平均值，并把结果输出到显示器上。这里，变量用的是整数数据类型 int，平均值如果有小数部分的话就将小数部分舍弃。如果不想舍弃的话，将变量的数据类型指定为 float 或 double 即可。

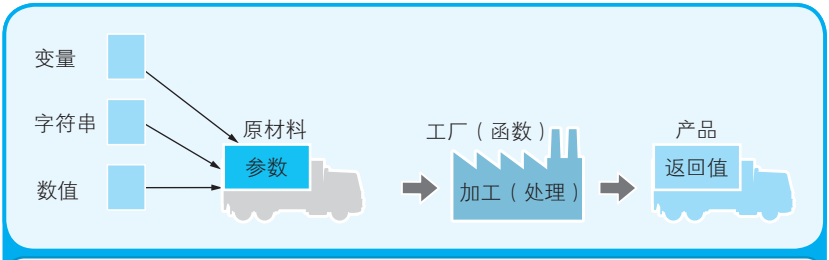


图 A-2　像工厂一样的函数

代码清单 A-2　具备输入、运算、输出功能的程序示例

```
int a, b, ave;              // 定义 3 个 int 类型的变量 a、b、ave
scanf("%d", &a);            // 接收从键盘输入的 a
scanf("%d", &b);            // 接收从键盘输入的 b
ave = (a + b) / 2;            // 计算 a 和 b 的平均值，并将结果赋值给 ave
printf("%d\n", ave);        // 把 ave 的值输出到显示器上
```

函数调用

在 C 语言中，是不能像代码清单 A-2 那样直接记述处理的，而是必须将这一系列的处理整合到函数中。而“整合到函数中”，就是程序员自己来记述函数的意思。

大规模的程序是由大量函数构成的，而像示例程序这种简单的程序，只需要一个函数就可以实现了。该函数的名称就是 main，这是规定。main 是程序启动时最初运行的函数。在由多个函数构成的程序中，程序启动时运行 main 函数，并在 main 函数中调用其他函数，然后该

函数又调用其他函数……，像这样，所需要的函数会被一个接一个地调用。而简单的程序中则仅仅包含了最初执行的 main 函数，因此，所有的处理都会集中在该部分进行。

代码清单 A-3 是把代码清单 A-2 的 5 行代码都整合到 main 函数中时的情况。函数的处理内容是用 {} 围起来的部分。{} 围起来的部分称为**模块**。模块（block）也有“整合”的意思。这里，为了便于大家理解模块的处理内容在 {} 之中，编写时特意在每行的开头空出了一些位置。运行时，按照代码记述的顺序，各个处理就会被从上往下依次执行。

代码清单 A-3　将所有处理整合到 main 函数的程序示例

```
#include <stdio.h>

void main(void) {
    int a, b, ave;          // 定义 3 个 int 类型的变量 a、b、ave
    scanf("%d", &a);        // 接收从键盘输入的 a
    scanf("%d", &b);        // 接收从键盘输入的 b
    ave = (a + b) / 2;        // 计算 a 和 b 的平均值，并将结果赋值给 ave
    printf("%d\n", ave);    // 把 ave 的值输出到显示器上
    }
```

void main(void) 中的 void 表示的是该 main 函数没有参数也没有返回值的意思。void 的字面意思是“空的”。文章开头的 #include ，表示的是参考 stdio.h 文件的意思。include 的字面意思是“包含”。在 stdio.h 文件中，定义了标准函数库 printf 和 scanf。该文件就称为**头文件**。头文件的扩展名为 header 的头一个字母“.h”。各标准库函数用到的头文件，都是同编译器一起安装的。

该函数的处理内容比较短，因此并不需要多个函数。即便如此，如果我们非要将其分成两个函数的话会怎样呢？我们不妨来看一下。分开后如代码清单 A-4 所示。在 main 函数中，从键盘输入两个数值并分别赋值给 *a* 和 *b*，然后把这两个数值作为参数传递给刚做成的

average 函数，再将 average 函数的返回值赋值给变量 *ave*，这样 *ave* 的数值就被输出在显示器上了。由此可见，average 函数成功地实现了求解作为参数的两个数值的平均值并把结果返回这一操作。这是因为 main 函数调用了 average 函数。

代码清单 A-4　从 main 函数中调用 average 函数的程序示例

```
#include <stdio.h>

int average(int, int);          // 定义 average 函数的原型

void main( void ) {
    int a, b, ave;              // 定义 3 个 int 类型的变量 a、b、ave
    scanf("%d", &a);            // 接收从键盘输入的 a
    scanf("%d", &b);            // 接收从键盘输入的 b
    ave = average(a, b);        // 计算 a 和 b 的平均值，并将结果赋值给 ave
    printf("%d\n", ave);        // 把 ave 的值输出到显示器上
}

int average(int a, int b) {
    return (a + b) / 2;         // 把 2 个参数的平均值作为返回值返回
}
```

int average(int a, int b){…} 开头的 int，表示 average 函数的返回值是 int 类型，括号中的 int 表示的是参数 a 和 b 是 int 类型。return 是返回函数返回值的指令。这里返回的是 (a+b)/2，也就是 a 和 b 的平均值。

请大家注意一下代码清单 A-4 中的注释“定义 average 函数的原型”这一部分。编译器会按照从上到下的顺序解析源代码的内容。而如果 main 函数中突然出现 average 函数的话，编译器就会理解为“没有该函数”，从而进行报错。正因为如此，就有必要在代码的开头部分加上 int average(int,int)，来告诉编译器“在后面有一个名是 average、返回值是 int 类型、两个参数也是 int 类型的函数”。这个就称为**函数原型定义**。

前面已经提到，stdio.h 文件中定义了标准函数库中的 printf 和 scanf 函数。具体来说，就是在 stdio.h 文件中定义了 printf 和 scanf 的原型。

局部变量和全局变量

在函数模块中定义的变量，只能在该函数中使用。这样的变量就称为**局部变量**。局部（local）是“当地的、地区的”的意思。在代码清单 A-4 中，main 函数中定义的 *a*、*b*、*ave* 都是局部变量。如果将局部变量的数值传递给其他函数来处理的话，该数值就会被作为参数来使用。代码清单 A-4 就是把 main 函数的局部变量 *a*、*b* 作为参数传递给了 average 函数。

变量也可以在函数模块外进行定义（虽然函数处理必须要在函数的模块中进行，但变量是可以在模块外进行定义的），该变量称为**全局变量**。全局（global）是“全世界的、全体的”意思。全局变量在程序的所有函数中都可利用。因而，通过利用全局变量，在函数中就可以获取其他函数的数值。不过，在大规模的程序中过多使用全局变量的话，就会使程序内容变复杂（无法清楚掌握是哪些函数在使用全局变量），这一点一定要注意。

将代码清单 A-4 的程序示例改造为使用全局变量的形式后，结果就如代码清单 A-5 所示。可以看出，average 函数的参数没有了。而 *ave* 还是局部变量的形式。这是因为 *ave* 仅仅被用在了 main 函数中。

代码清单 A-5　利用全局变量的程序示例

```
#include <stdio.h>

int average(void);          // 定义 average 函数的原型
int a, b;                   // 定义全局变量 a、b

void main(void) {
    int ave;                // 定义局部变量 ave
    scanf("%d", &a);        // 接收从键盘输入的 a
    scanf("%d", &b);        // 接收从键盘输入的 b
    ave = average();        // 计算 a 和 b 的平均值，并将结果赋值给 ave
    printf("%d\n", ave);    // 把 ave 的值输出到显示器上
```

```
}

int average(void) {
    return (a + b) / 2;  // 把两个参数的平均值作为返回值返回
}
```

数组和循环

处理大量数据是计算机擅长的领域之一。例如，在求解 100 万个数据的平均值的时，利用计算机瞬间就能完成。在程序中表现大量数据时，通常会使用**数组**的形式。数组的全体数据用同一个名字（数组的名字）来表示，各数据（称为元素）则通过从 0 开始的连续编号（称为索引）来进行区分。100 万个数据的话，输入起来太过麻烦，因此，这里我们就来做一个求解 10 个数据的平均值的程序，如代码清单 A-6 所示。

代码清单 A-6　求解 10 个数据的平均值的程序示例

```
#include <stdio.h>

void main(void) {
    int data[10];       // 定义具有 10 个元素的数组 data，数据类型为 int
    int sum, ave, i;    // 定义 3 个 int 类型的变量 sum、ave、i

    sum = 0;            // 把用来保存总和结果的 sum 清 0

    // 将 i 从 0 ~ 9 逐一 + 1，递增循环
    for (i = 0; i < 10; i++) {
        scanf("%d", &data[i]); // 把从键盘输入的数值存入 data[i] 中
        sum += data[i];        // 把把 data[i] 的数值累加到 sum 中
    }

    ave  = sum / 10;    // 用 sum 除以 10 得到平均值
    printf("%d\n", ave);    // 将 ave 的值输出到显示器上
}
```

int data[10]; 部分是数组的定义。表示的是“请准备好数据类型是 int、有 10 个元素、数组名为 data 的数组”。定义数组后，data[0]、data[1]、data[2]、data[3]、data[4]、data[5]、data[6]、data[7]、data[8]、