第3章 与计算机面对面地交流——数据的输入与输出

想象一下，我们人类之间面对面交流的场景。我们彼此注视着对方的眼睛，倾听着对方的语言，观察着彼此的神情和动作，完成一次愉悦的对话。当我们的交谈对象是计算机时，我们是如何与计算机面对面交流的呢？

我们紧紧地盯着计算机的屏幕，观察着屏幕上出现的字符，不断地敲击着键盘。计算机接收键盘传给它的数据，经过它的处理后，屏幕再一次出现一排排新的字符，这就是我们与计算机交流的场景——无声的“对话”。

3.1 我们与计算机的交流方式

我们与计算机之间交流的内容是数据，交流的方式是输入和输出数据，交流的工具是计算机的输入设备（如键盘、鼠标、摄像头和扫描仪等）和输出设备（如显示器、打印机、音箱等）。计算机通过程序不断地接收、处理和输出这些数据，推动我们与计算机之间进一步的数据交互。

3.1.1 人类与计算机理解数据的差异性

计算机的世界是二进制的数字世界。计算机能够存储二进制数据，也能够计算二进制数据，但是它一般不显示二进制数据，因为我们人类不容易看懂二进制数据。对于计算机存储的二进制数据，我们可以要求它按整型、浮点型、字符型等任意数据类型进行显示，但是只有按照数据存储时的数据类型进行读取并显示才有意义。例如，

```
char ch = 'A';
```

在变量 ch 中存储了字符'A'的 ASCII 码值 65，即 01000001。如果现在让计算机在屏幕上显示变量 ch 中的内容，我们必须告诉它按照什么样的数据类型解释 ch 中的数据。如果按照 int 类型解释，那么显示器上将显示 65。如果按照 char 类型解释，那么将显示英文字母 A。如果按照浮点型解释，那么将显示 0.000000。显然，对变量 ch 中所存储数据的最恰当输出方式是按 char 类型解释并显示。

向计算机中输入数据，也存在着同样的问题。当我们要从键盘上为字符变量 ch 输入字符'A'时，我们是输入字符'A'，还是输入它的 ASCII 码值 65 呢？如果告诉计算机，现在输入的数据按字符解释，那么就要输入字符'A'。如果此时输入 65，那么计算机就会认为 65 是两个字符 6 和 5，从而产生错误。如果告诉计算机，现在输入的数据按整型解释，那么就必须输入 65。如果此时再输入字符'A'，那么同样也会产生错误，因为阿拉伯数字中没有字符'A'，计算机无法按整型数据理解此时的字符'A'。

当我们在阅读计算机屏幕上所显示的数据时，我们的大脑是不会去考虑哪些数据是整数，哪些数据是字符，哪些数据是浮点数的。对我们来说，我们只关心这些数据所包含的信息而不关心它们的数据类型。但是，计算机却必须关注这些数据的数据类型，否则它就无法正确地将二进制数据转换成我们所需要的信息内容。因此，如何正确地向计算

机中输入和输出数据是程序员必须了解与掌握的内容。

3.1.2 计算机如何输入和输出数据

我们通过输入输出系统与计算机交互数据。**输入输出系统是计算机系统与外部进行通信的子系统**,它是计算机系统的重要组成部分。输入输出系统由输入设备、输出设备和输入输出控制系统3部分组成。其中,输入输出控制系统是在计算机中对外围设备实施控制的系统,它的主要功能是通过向外围设备发送控制命令控制输入和输出数据的传送以及检查外围设备的状态。

输入设备是向计算机输入数据和信息的设备,用于把数据和处理这些数据的程序输入到计算机中。键盘、鼠标、摄像头、扫描仪、光笔、手写输入板、游戏杆、语音输入装置等都属于输入设备。计算机既可以接收数值型的数据,也可以接收各种非数值型的数据,如图形、图像、声音等。非数值型的数据可以通过不同类型的输入设备转换成二进制数据后输入到计算机中,进行存储、处理和输出。

输出设备是计算机输出数据的设备。常见的输出设备有显示器、打印机、音箱等。利用各种输出设备可将计算机输出的信息转换成人能够识别的形式显示在屏幕上,如数字、文字、符号、图形和语音等,或者记录在磁盘、磁带、纸带和卡片上,或送给相关控制设备。

虽然计算机有各种各样的输入和输出设备,但是这些设备都有一个共同的目标——数据的格式转换。把我们容易理解的文本、图像、语音等数据转换成计算机能够处理的二进制数据输入到计算机中,再把计算机处理后的二进制数据转换成文本、图像、语音等数据输出。

本书将介绍利用C语言通过键盘、显示器、内存和硬盘等设备与计算机进行交互数据的方法。

3.1.3 两种对话方式的选择

在生活中,人与人之间的对话内容需要保存吗?我们一般不会记录聊天的内容,而对于重要的会议内容,我们会用纸笔记录下来。在与计算机对话的过程中,也存在着类似的场景。有的时候,我们通过键盘输入数据后,计算机直接将计算结果显示在屏幕上,我们看看就可以了,不需要保存结果数据。而有的时候需要将对话内容保存下来,以便于日后随时查看和使用。当然保存我们与计算机之间的对话内容是需要付出一定代价的,需要使用硬盘等具有持久性存储数据能力的设备来保存数据。

微课3.1 两种对话方法

下面介绍与计算机交互数据的两种方式:一种是不保存对话内容,通过键盘和显示器与计算机交互数据;另一种是保存对话内容,通过内存与硬盘交互数据,参见图3.1。

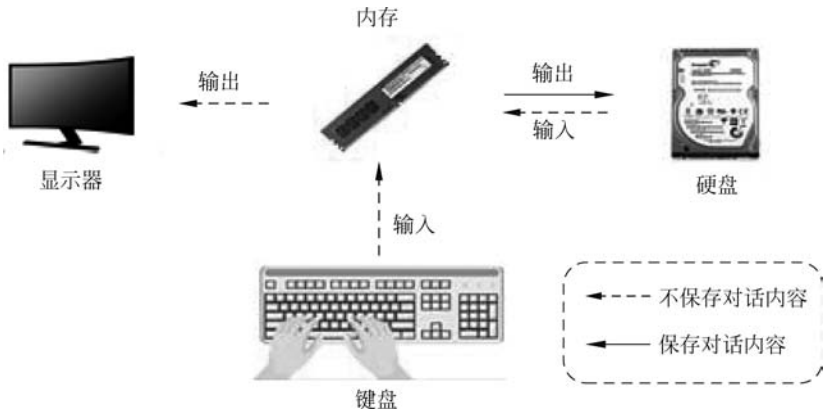

图 3.1　计算机的两种数据输入输出方式示例

1. 通过键盘和显示器交互数据

利用键盘和显示器可以实现与计算机之间的实时数据交互。我们从键盘上将数据输入到内存中并存储在程序的变量里面。在程序运行过程中,计算机依据指令将变量中的数据输出到显示器上。根据显示结果,我们再次通过键盘输入数据,如此循环。在这个过程中,数据被暂时保存在内存中,但是内存不能持久存储数据。当程序运行结束时,操作系统会清除内存中的程序数据,从而无法再访问程序中的数据。

2. 通过内存和硬盘交互数据

利用内存和硬盘可以将程序中的数据持久地保存在硬盘中。计算机可以将内存中的数据写入到硬盘中,以文件的方式保存下来。即使程序退出运行,程序中的数据也已经保存在文件中了,不会丢失。当程序重新运行时,我们可以让计算机从硬盘的文件中将数据读入到内存中供程序使用,从而不需要通过键盘再次输入数据。

通过这两种方式,既可以利用键盘实时地将数据输入到程序的变量中,也可以通过读取文件的方式向变量中输入数据。既可以将程序中变量的数据输出到显示器上,也可以将变量的数据输出到文件中。将数据输出到文件的交互方式可以达到持久保存程序数据的目的。

3.2　通过键盘和显示器与计算机交流

键盘和显示器的工作原理是复杂的。为了降低使用它们的复杂性,专业的程序人员开发了通过键盘向变量中输入数据以及将程序中的数据输出到显示器上的相关指令。在 C 语言中,这些指令集被称为标准输入输出库函数。

除了标准输入输出库函数以外,C 语言还提供了其他库函数,这些库函数统称为 C 语言标准库函数。下面对 C 语言标准函数库进行简要的介绍。

3.2.1 C语言标准函数库

C语言标准函数库并不是C语言标准中的一部分，而是由C语言编译器根据一般用户的需要编制并提供给用户使用的一组程序代码。不同版本的C语言编译系统提供的库函数是由不同编译软件研发公司开发的，由于版权原因，库函数的源代码一般是不可见的，但是库函数和一些自定义符号常量的声明会用一个独立的文件进行保存，以方便使用者调用，这个文件被称为头文件。在程序中调用这些库函数时，需要使用include包含指令将这些库函数的头文件复制到源文件中。

C语言标准函数库包括了C语言建议的全部标准函数，还根据用户的需要补充了一些常用函数，并且已经对这些函数的源代码进行了编译，形成了目标文件。当我们在程序中使用了库函数时，编译系统在编译源程序时并不会检查库函数的源代码，但是会检查是否包含了所调用的库函数的头文件。在程序连接阶段，编译系统会连接库函数的目标文件，与其他源文件的目标文件共同生成一个可执行的目标程序。在程序中未使用库函数与使用库函数的编译和连接过程参见图3.2。

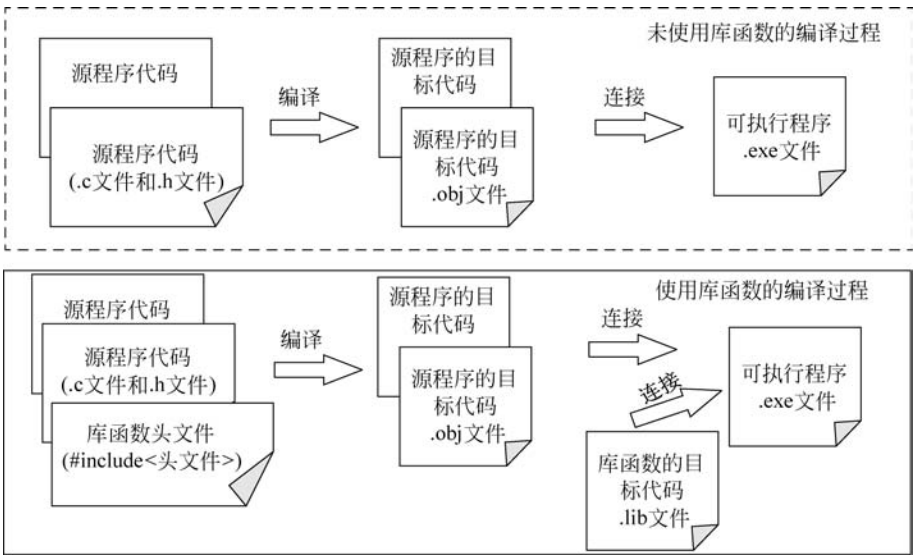

图3.2 未使用和使用库函数的编译过程

如图3.2中的虚线框所示，在用户编写的源程序代码中未使用库函数。在程序编译过程中，多个源文件经编译后，生成了多个目标文件。当程序连接成功后便生成了可执行程序。

如实线框所示，在用户编写的源程序代码中使用了库函数。在源文件中需要包含所使用的库函数的头文件，如“#include <头文件>”。include指令是程序预编译的包含指令，指示编译器将文件的全部内容插入此处。编译系统会将该头文件中关于函数声明的程序代码复制到源文件中。

预编译又称为预处理，是为程序编译做预备工作的阶段，主要完成一些代码文本的

替换工作。预编译的内容主要包括宏定义、文件包含和条件编译。我们前面学习的符号常量的定义就是宏定义的一种。include 预编译指令属于文件包含，当我们在程序引入某个头文件并调用其中的函数时，编译系统在连接阶段会连接“#include <头文件>”中头文件对应的目标文件，与源文件生成的目标文件一起生成可执行文件。

常用的库函数头文件有 stdio.h、math.h、string.h 等。在 stdio.h 头文件中声明了输入输出的库函数，如 scanf 函数、printf 函数以及打开文件的 fopen 函数等。在 math.h 头文件中声明了常用的一些数学函数，如指数函数 exp、开方函数 sqrt、正弦函数 sin 等。在 string.h 头文件中声明了常用的字符串操作函数，如字符串长度函数 strlen、字符串复制函数 strcpy 等。

3.2.2 通过键盘输入数据

由于键盘与内存硬件之间的数据传输速度存在差异，因此 C 语言提供了标准输入流对输入数据进行缓冲管理。在利用键盘连续输入数据时，数据不会马上送入缓冲区。只有当我们输入了换行符后，从键盘输入的数据才会被送到缓冲区中。当调用相关函数读取数据时，函数会先去缓冲区查看是否有数据存在。如果缓冲区中没有数据，函数则需要等待，直到用户从键盘将数据输入到缓冲区中。

在 C 程序中，可以使用 scanf 函数从缓冲区中将数据读取到指定的变量中。用户可以指定输入数据的类型和数量。scanf 函数能够输入整型、浮点型、字符型等数据，并且可以一次性输入若干个任意类型的数据。

2.6.1 节介绍了表达式语句、复合语句和空语句，现在介绍函数调用语句。

函数调用语句由一个函数调用和一个分号组成，格式为：

```
函数名称(参数 1,参数 2,…,参数 n);
```

函数名称的命名规则与变量相同，它们都是标识符。参数是输入到函数中的数据，由函数处理后再输出结果数据。关于函数的知识，将在第 5 章中详细介绍。这里只要学会如何调用这些库函数就可以了。

1. scanf 语句

scanf 语句是对 scanf 函数调用的语句，其一般形式表示为：

```
scanf(参数 1,参数 2,...,参数 n);
```

微课 3.2　scanf 语句

参数 1：格式控制字符串。它是用双引号括起来的一个字符串，它指定了要输入到变量中的数据的格式。例如，为一个字符变量输入数据，格式字符串是"%c"。

参数 2～参数 n：需要输入数据的变量的地址。如果要为多个变量输入数据，就需要指定多个参数。如果要获得变量的地址，需要使用取地址运算符“&”。在使用 scanf 语句时，需要特别注意的是，**格式控制串中指定的数据格式应与变**

量的数据类型一致。

为什么参数 2～参数 n 是变量的地址呢？这个道理很容易理解。当你网购了一件货物，快递员不仅需要知道你的名字，而且需要知道你的地址才能把货物准确地送达给你。因此，当我们从键盘上为一个变量输入数据时，计算机不但需要知道变量的名称，还需要知道变量的地址，这样才能找到变量，并把数据准确地写入到变量中。

例如，从键盘向 char a 变量中输入一个字符，scanf 语句参见图 3.3。

图 3.3 scanf 语句举例

其中，"%c"是格式控制字符串，c 表示从键盘输入数据的类型是字符型，&a 表达式的运算结果是获得变量 a 的地址。"%c"格式控制符的作用是表示此时从缓冲区中按照字符类型读取一个字节的数据并存储到变量 a 中。通过程序调试的方法，观察该语句的执行结果，参见图 3.4。

在 scanf 语句执行之前，局部变量 a 中的数值未知。当执行 scanf 语句后，程序中出现了命令行窗口，参见图 3.5。在命令行窗口中，按下键盘上的数字 0 键，并按下回车(Enter)键输入换行符后，此时观察变量 a 中的数据内容，可以发现字符'0'的 ASCII 码值 48 已经成功地输入到变量 a 中。

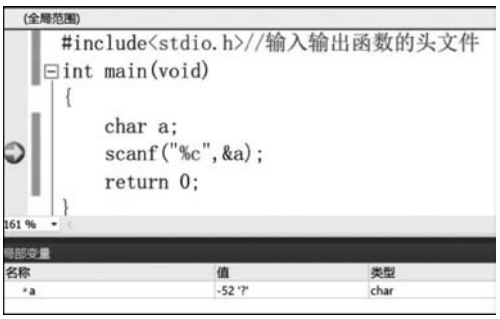

图 3.4 scanf 语句运行前

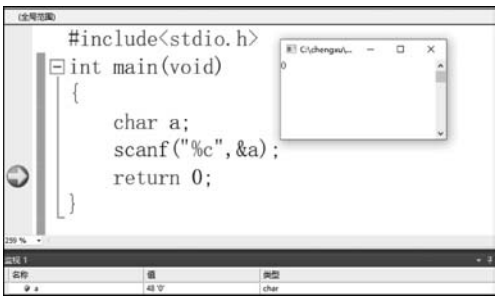

图 3.5 执行 scanf 语句

如果不用格式字符串"%c"，那么当我们按下数字键 0 时，计算机将无法知道到底是输入字符'0'，还是整数 0，还是浮点数 0 呢？对于计算机来说，它们是不同的。

如果要为 int b 变量输入数据，它的格式控制符是"%d"，相信大家很容易写出下面的代码：

```
scanf("%d",&b);
```

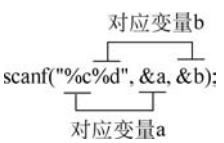

图 3.6 通过 scanf 语句输入多个数据

使用一条 scanf 语句可以同时为多个变量输入数据。我们可以使用一条 scanf 语句同时为 char a 和 int b 两个变量输入数据，参见图 3.6。

在格式字符串中，每个格式符的位置需要与变量地址的位置一一对应。如果格式符号%c 和%d 的位置要调换，那么&a 和 &b 的位置也要调换。如果现在需要为变量 a 输入字

符'0',为变量 b 输入整数 25,那么程序运行后,在命令行窗口中连续输入“025”,数据将正确地传递到变量 a 和变量 b,参见图 3.7。

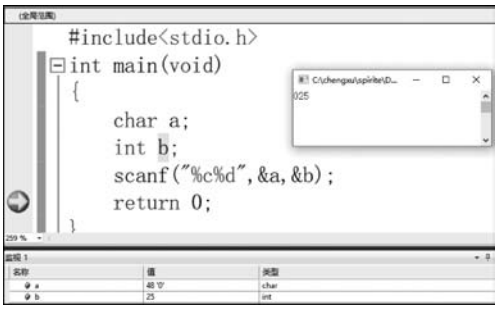

图 3.7　为多个变量同时输入数据

当连续输入“025”时,不容易看出这是输入了两个数据。如果用字符、空格或者逗号将输入的数据分隔开,那么就容易分辨输入数据之间的区别了。例如,输入“0　25”或者“0,25”,或者“a=0,b=25”。如果从键盘输入数据时使用了一些空格、逗号、字母等字符常量对数据进行了分隔,那么在格式控制字符串中也需要加入相应的空格、逗号、字母等字符常量。当 scanf 语句从缓冲区中读取数据时,它会将输入数据与格式控制字符串中的字符常量进行一一匹配,从而从输入数据中读取出不是字符常量的数据,送给相应的变量。

例如,从键盘为变量 a 和变量 b 输入数据的格式为"a=0,b=25",那么格式字符串"%c%d"需要修改为"a=%c,b=%d"。在这个格式字符串中多出来的'a'、'='、'b'都是常量字符。scanf 语句如下:

```
scanf("a=%c,b=%d",&a,&b);
```

程序运行结果参见图 3.8。

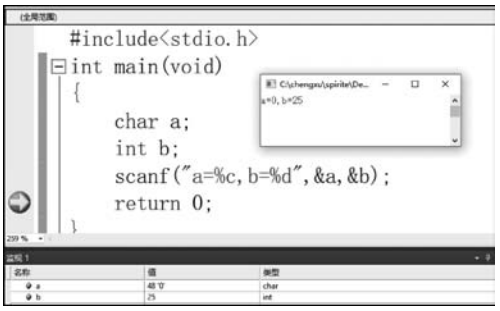

图 3.8　在格式字符串中加入了常量字符

假设将从键盘输入的数据“a=0,b=25”替换为“a=0 b=25”,也就是把“,”改成空格,那么输入数据的格式与 scanf 语句中的格式字符串就会不匹配,从而导致数据输入错误。在此,如果将英文逗号输入成中文逗号,程序也会报错。

2. scanf 语句中的格式符

scanf 语句中常见的格式符参见表 3.1。

表 3.1 scanf 语句中常见的格式符

格式字符	说明
d,i	输入有符号的十进制整数
u	输入无符号的十进制整数
o	输入无符号的八进制整数
x,X	输入无符号的十六进制整数(大小写作用相同)
c	输入单个字符
s	输入字符串,将字符串送到一个字符数组中
f	输入实数,可以用小数形式或指数形式输入
e,E,g,G	与 f 作用相同,e 与 f、g 可以相互替换(大小写作用相同)

scanf 语句中常见的附加格式符参见表 3.2。

表 3.2 scanf 语句中常见的附加格式符

格式字符	说明
l	输入长整型数据(可用%ld,%lo,%lx,%lu),以及 double 型数据(用%lf 或%le)
h	输入短整型数据(可用%hd,%ho,%hx)
域宽	指定输入数据所占宽度(列数),域宽应为正整数
*	本输入项在读入后不赋给相应的变量

在表 3.2 中,域宽在“%”和格式字符之间,用于限制从对应域读取的最大字符数。例如,

```
scanf("%5d",&x);
```

当输入“1234567”时,变量 x 读取的数值为 12345。*表示读取指定类型的数据但不保存到变量中。例如,

```
scanf("%d%*c%d",&x,&y);
```

当输入“10A25”时,变量 x 存储 10,变量 y 存储 25,字符 A 会被读取但是不会存储到任何变量中。

3. 输入缓冲区

在输入数据时,只有按下回车键后,scanf 语句才开始读数据。这是因为 scanf 语句并不是直接从键盘上读数据,而是从输入缓冲区中读数据。为了提高从键盘上输入数据的效率,操作系统设置了输入数据缓冲区。从键盘上输入的数据会先输送到输入缓冲区,再从输入缓冲区中读到程序中。在输入数据时,每按下一次回车键,系统就会将当前输入的数据输送到缓冲区里面。一旦缓冲区里面有数据了,scanf 语句就会读走它需要

的一批数据。当它读取后，这些数据就会从缓冲区中清除掉。

假设我们一次性通过键盘向缓冲区中写入了一批数据，其中前面一部分是 scanf 语句需要读取的，后面一部分是多余的。当 scanf 语句读取了前面的数据后，后面多余的数据还会留存在缓冲区中。如果此时再用 scanf 语句或者其他输入函数读取数据，就会读取到这些多余的数据。如果不继续读取这些多余的数据，那么这些多余的数据就会继续留存在缓冲区中。只有当程序退出时，计算机才会自动清除缓冲区中的数据。

4. scanf 语句使用时需要注意的问题

从键盘输入的每个字符都会被输送到输入缓冲区中，包括输入数据结束后输入的回车符。当从键盘上连续输入多个数时，数与数之间需要用空格、Tab 键或者换行字符分开（一个空格、Tab 键或者换行字符和多个的作用相同），否则无法区别这是一个数还是多个数。但是在连续输入字符时，字符之间不能用空格分隔，因为空格也是字符，这样就会将空格作为字符读给字符变量，从而产生错误。另外，当 scanf 语句读取数时，如果遇到不是数字的字符，则会认为读入数据结束。

【例 3.1】 使用 scanf 语句连续读入数和字符。

程序代码如下：

```
1   #include <stdio.h>
2   int main( )
3   {
4         int a;
5         float b;
6         char c;
7         scanf("%d%f%c",&a,&b,&c);                    //输入数据
8         return 0;
9   }
```

采用程序调式的方式运行程序，执行语句 7 后，从命令行中输入“2 92.5 A”，程序的执行结果参见图 3.9。

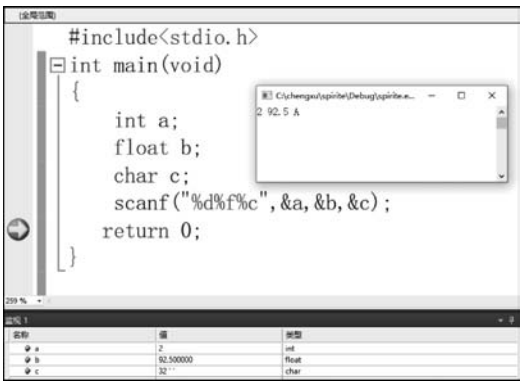

图 3.9　scanf 语句连续读入数字和字符

从局部变量监视窗口中，可以发现变量 a 中输入的数值是 2，变量 b 中输入的数值是 92.500000，变量 c 中输入的数值是 32，而不是字符'A'。数值 32 是空格字符的 ASCII 码值。

当按下回车键时，将数据“2 92.5 A”输送到输入缓冲区，输入数据在缓冲区中的排列顺序参见图 3.10。从图 3.10 中可以看出，此次从键盘输入了 9 个字符，最后一个字符是换行符（相当于按 Enter 键）。

序号	1	2	3	4	5	6	7	8	9
数据	50 '2'	32 空格	57 '9'	50 '2'	46 '.'	53 '5'	32 空格	65 'A'	10 换行符

图 3.10　输入数据存储位置示意图

当执行“scanf("%d%f%c",&a,&b,&c);”语句时，先执行%d，从缓冲区中读取第一个数据“2”，送给变量 a，同时删除缓冲区中的数据“2”。再执行%f，从缓冲区中读取了数据“32”，发现“32”是空格字符的 ASCII 码，于是删除“32”，继续读取第 3～6 个数据 57、50、46、53，对应为“92.5”，送给变量 b。继续执行%c，读取第 4 个数据“32”，空格是一个合法的字符，于是将数据“32”赋值给变量 c，此次读取数据结束。读完数据后，缓冲区还有残留数据字符'A'和换行符，参见图 3.11。

序号	1	2	3	4	5	6	7	8	9
数据								65 'A'	10 换行符

图 3.11　缓冲区还有残留数据字符示意图

通过上面的例子可以看出，如果在格式控制字符串中没有用空格字符来分隔字符格式“%c”，那么从键盘输入字符时也不能用空格字符来分隔字符。如果从命令行输入数据时去掉了 92.5 与 A 之间的空格，如“2 92.5A”，那么就可以完成正确的数据输入了，参见图 3.12。

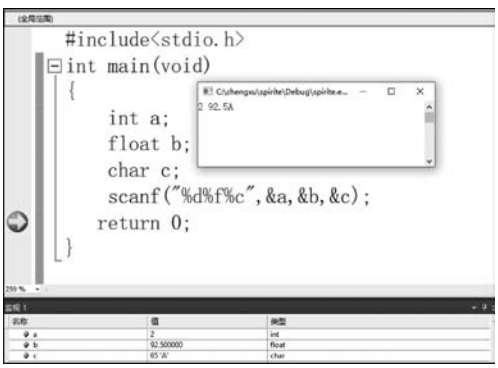

图 3.12　输入数据

当字符'A'读入变量 c 中后，缓冲区中还存留了一个回车符，这会不会对后续的数据读取造成影响呢？有可能。如果下一次读取的是一个数字，则不会造成影响，因为 scanf

语句在读取数字的时候，遇到空格、Tab 键或者回车符会自动过滤掉这些字符。但是如果下次读取的是字符呢？那么回车符就会被当成合法的字符读入，从而产生错误。

下面通过一个例子介绍这种错误产生的场景。首先对上面的代码进行修改，增加一个 char d 变量，为变量 a、b、c、d 输入数据。现在需要将 scanf 语句的格式控制字符串修改为"%d%f%c%c"。

从命令行中分两次分别输入数据“2 92.5A”和“B”，将字符'B'赋值给变量 d，参见图 3.13。第一次在命令行中输入“2 92.5A”，按下回车键，变量 a、b、c 中都读入了正确的数值，变量 d 中也有了数据 10(换行符)。从命令行继续输入字符'B'，此时发现已经无法再输入数据，这是怎么回事呢？

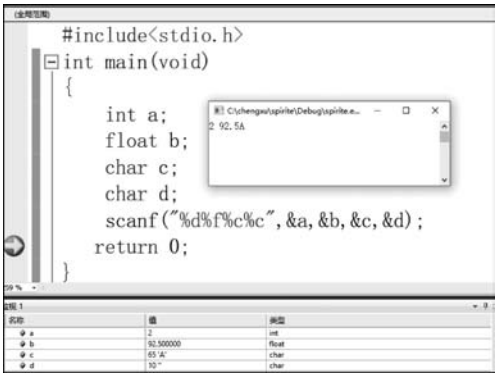

图 3.13　输入数据

这是因为从键盘输入“2 92.5 A”并按下回车键后，换行符也被系统送入了输入缓冲区，scanf 语句将它读取后并赋值给变量 d，它无法分辨这个回车符不是我们所需要的字符。当使用 scanf 语句连续输入字符数据的时候，这是会经常遇到的问题。如果想解决这个问题，只需要再执行一次 scanf("%c",&d)语句，就可以从命令行输入字符'B'，大家可以动手尝试一下。另外，利用 scanf 语句的"%s"格式符读取字符串数据也是一个很重要的功能，具体将在第 6 章字符数组中进行介绍。

【例 3.2】 使用 scanf 语句利用指针变量通过间接访问变量的方式为变量输入数据。

程序代码如下：

```
1   #include <stdio.h>
2   int main( )
3   {
4       int a, *pa = &a;                    //定义指针变量 pa,并初始化为 a 的地址
5       float b, *pb = &b;                  //定义指针变量 pb,并初始化为 b 的地址
6       char c, *pc = &c;                   //定义指针变量 pc,并初始化为 c 的地址
7       scanf("%d%f%c",pa,pb,pc);           //以指针变量代替所指向普通变量的地址
8       return 0;
9   }
```

与【例 3.1】相比，在第 4、5、6 行的变量定义中，增加了指向 int 变量的指针变量 pa，

指向 float 变量的指针变量 pb，指向 char 变量的指针变量 pc，并且初始化对应指针变量的值分别为变量 a、b、c 的地址。这样 pa、pb、pc 就可以直接作为 scanf 输入语句中的参数 2～参数 4 使用。当从命令行输入数据“2 92.5A”时，同样可以完成对变量 a、b、c 的数据输入。

3.2.3 通过显示器输出数据

通过 printf 语句可以将程序中变量的数据输出到显示器上。printf 语句与 scanf 语句的格式很相似，它可以一次性将若干个变量、常量、表达式的数据输出到显示器上。

微课 3.3 printf 语句

1. printf 语句

printf 语句是对 printf 函数调用的语句，其一般形式为：

```
printf(参数 1,参数 2,…,参数 n);
```

参数 1：格式控制字符串，它指定了输出数据的格式或者是输出的一个字符串常量。

参数 2～参数 n：需要输出数据的变量、常量、表达式。

在 printf 语句中，参数 1 的作用与 scanf 语句的参数 1 是相同的，但是参数 2～参数 n 与 scanf 语句中的参数不同。

2. printf 语句中的格式符

printf 语句中的格式符参见表 3.3。

表 3.3 printf 语句中常见的格式符

格式字符	说　明
d,i	以带符号的十进制形式输出整数(正数不输出符号)
u	以无符号十进制形式输出整数
o	以八进制无符号形式输出整数(不输出前导符 0)
x,X	以十六进制无符号形式输出整数(不输出前导符 0x)，用 x 则以小写形式输出十六进制数的 a～f，用 X 时，则以大写字母输出 A～F
c	以字符形式输出，只输出一个字符
s	输出字符串
f	以小数形式输出单、双精度数，默认输出 6 位小数
e,E	以指数形式输出实数，用 e 时指数以“e”表示(如 3.1e+01)，用 E 时指数以“E”表示(如 3.1E+01)
g,G	选用%f 或%e 格式中输出宽度较短的一种格式，不输出无意义的 0。用 G 时，若以指数形式输出，则指数以大写表示

printf 语句中常见的附加格式符参见表 3.4。

表 3.4　printf 语句中常见的附加格式符

格 式 字 符	说　　明
l	长整型数据，可加在格式符 d、o、x、u 前面
m	一个正整数，表示数据最小宽度
n	一个非负整数，对实数表示输出 n 位小数；对字符串表示截取的字符个数
-	输出的数字或字符在域内向左靠

printf 语句中的绝大部分格式符的作用与 scanf 语句中的格式符作用相一致。下面介绍几种常用的格式符。

1）指定数据在屏幕的显示位置

为了让数据在屏幕上排列得更美观，当要输出多行数据的时候，可以指定每行数据输出时的位置。这种格式指令采用了附加格式符“m”，m 格式符定义了在屏幕上显示数据的最小宽度，只能是正整数。它需要和其他格式符联合使用，与整型格式符联合使用，如“%md”。与浮点型格式符联合使用，如“%mf”。

【例 3.3】 用字符'*'在屏幕上输出一个三角形。

程序代码如下：

```
1   #include <stdio.h>
2   int main( )
3   {
4       printf("%3s\n","*");          //其中 3 表示显示'*'需要占用的列宽度是 3 列
5       printf("%3s\n","**");
6       printf("%3s\n","***");
7       return 0;
8   }
```

程序运行后，输出显示参见图 3.14。

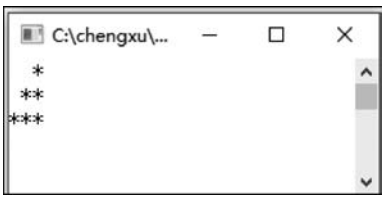

图 3.14　输出结果

继续执行下列语句：

```
printf("%3d\n",5);
printf("%3f\n",3.1);
```

在显示器上输出数据时，数据按照行和列排列输出，每个显示的字符占一个位置。在屏幕上输出数据的排列参见图 3.15。

在屏幕第 5 行输出显示浮点数“3.1”时多了 5 个 0。这是因为 printf 语句在输出 float 和 double 类型数据时默认保留 6 位小数。虽然输出的数字“3.100000”的长度超过了列宽 3，但是为了保持它的精度，超过列宽的部分数位仍然显示。**当输出数据的长度超过了列宽 m 规定的宽度时，printf 语句按照数据的实际长度输出数据值**。另外，当指定 m 时输出的数据靠右对齐，当指定－m 时，输出的数据则靠左对齐。

	1列	2列	3列	4列	5列	6列	7列	8列
1行	空格	空格	*					
2行	空格	*	*					
3行	*	*	*					
4行	空格	空格	5					
5行	3	.	1	0	0	0	0	0

图 3.15　数据在显示器屏幕中输出时的位置示意图

在 scanf 语句中，也可以使用“%md”“%ms”等指定读入数据的宽度。例如，

```
int a;
scanf("%3d\n",&a);
```

语句执行后，只能从键盘输入 3 个数位的数字给变量 a。

2）指定输出浮点数的小数位数

如果想指定输出浮点数的小数位数，则需要用到附加格式符“.n”，表示输出显示该浮点数的 n 位小数点。

例如，

```
printf("%.2f\n",3.14);
printf("%.10f\n",3.14);
```

语句执行后，输出的结果如下：

```
3.14
3.1400000000
```

列宽格式符“m”和“.n”格式符可以联合使用。

例如，

```
printf("%10.2f\n",3.14);
```

语句执行后，输出的结果如下：

```
      3.14
```

3）以指数形式输出浮点数

对于浮点数可以以指数形式输出显示，此时需要使用“%e”或者“%E”格式符。例如，

```
printf("%e",31.415926);
```

语句执行后，输出的结果如下：

```
3.141593e+001
```

31.415 926 有 8 位有效数字，但输出时只显示了 7 位有效数字。采用“%e”格式符输出数据时默认显示 6 位小数位，不显示的小数部分会自动“四舍五入”。

printf 语句在输出浮点型数据时，它会自动地对未显示的小数部分“四舍五入”。另外，利用 printf 语句显示的小数位数越多，并不代表数据越精确。例如，float 类型数据的有效位是 6 或 7 位，尽管可以使用“%.15f”格式符显示小数点后 15 位小数，但是超过 7 位有效位以外的其他数位都是无效的。

4）输出浮点数有效位不超过 6 位

如果要输出不超过 6 位有效位的浮点数，可以采用“%g”格式符。如果在小数位的后几位全是 0，那么 printf 语句在显示时会自动舍去。例如，

```
printf("%g\t%g\t%g",10.0/3,1000000.0/3,100000000.0/3);
```

语句执行后，输出的结果如下：

```
3.33333 333333 3.33333e+007
```

可以看出，对于表达式 1000000.0/3 的计算结果没有输出小数位，因为整数位正好是 6 位。如果整数位超过 6 位，会采用指数形式输出数据。

3.2.4 通过键盘和显示器完成一次完整对话

有了 scanf 输入语句和 printf 输出语句，现在可以通过下面的剧本设计一次与计算机之间的有趣对话。

【例 3.4】 让计算机分面包的对话实现。

我们可以给计算机起个名字“小精灵”，让它称呼我们“大怪兽”。

脚本如下：

① 小精灵：你好啊，很高兴认识你！

② 大怪兽：我也是，我遇到一个问题，你能帮我解决吗？

③ 小精灵：什么事情呢？

④ 大怪兽：我烤了一些面包，你能帮我算算怎么平均分给我的家人？

⑤ 小精灵：你烤了几个面包呢？

⑥ 大怪兽：X 个。

⑦ 小精灵：你家有几个人呢？

⑧ 大怪兽：Y 个。

⑨ 小精灵：我知道，应该每个人分配 Z 个面包。

在这个剧本中，通过键盘输入面包的数量和人数，计算机帮我们完成面包的分配，你能完成吗？

问题分析：从键盘输入面包的数量 X 和家人的人数 Y，通过除运算 Z＝X/Y 可以求得每个人分得的面包数量，最后输出 Z 的值。

算法设计：在算法中，对剧本中的对话进行了描述，参见图 3.16。

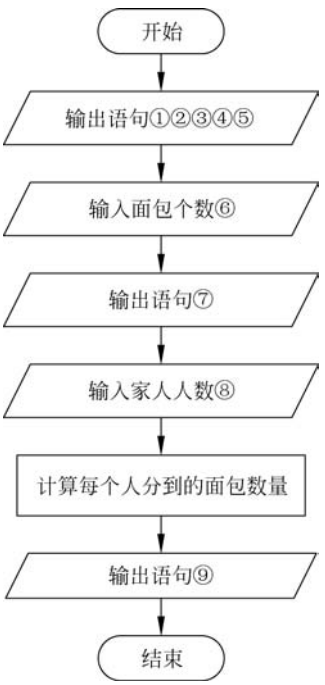

图 3.16 对话流程图

程序代码如下：

```
1   #include <stdio.h>
2   int main( )
3   {
4       int bread_num,person_num;
5       float average_num;
6       printf("小精灵:你好啊,很高兴认识你!\n");                                    //①
7       printf("大怪兽:我也是,我遇到一个问题,你能帮我解决吗?\n");                  //②
8       printf("小精灵:什么事情呢?\n");                                            //③
9       printf("大怪兽:我烤了一些面包,你能帮我算算怎么平均分给我的家人?\n");      //④
10      printf("小精灵:你烤了几个面包呢?\n");                                      //⑤
11      printf("大怪兽:");
12      scanf("%d个",&bread_num);                                                  //⑥
13      printf("小精灵:你家有几个人呢?\n");                                        //⑦
14      printf("大怪兽:");
15      scanf("%d个",&person_num);                                                 //⑧
16      average_num = (float)bread_num/person_num;
17      printf("小精灵:我知道,应该每个人分配%.1f个面包。\n",average_num);          //⑨
18      return 0;
19  }
```

由于面包的数量和家人的人数都是整数，而每个人分得的面包数量并一定是整数，因此分别定义变量如下：

```
int bread_num, person_num;
float average_num;
```

脚本中的语句⑥采用了两条程序语句实现：一条输出语句显示“大怪兽：”，另外一条输入语句读入“X个”。脚本中的语句⑧也采用了相似的两条程序语句实现。程序运行到第12行scanf语句后，等待从命令行中输入面包的个数。参见图3.17。

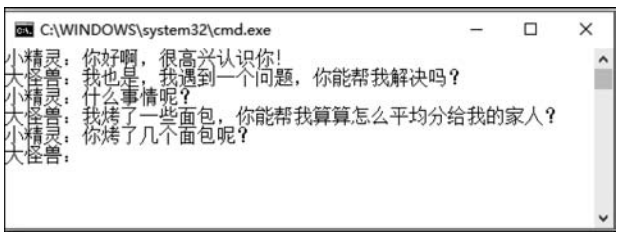

图3.17　对话等待输入

从命令行中输入“3个”后，程序继续运行到第15行scanf语句后，等待从命令行中输入家人的人数，参见图3.18。

从命令行中输入“2个”后，程序继续运行到第16行后计算平均值，然后运行到第17行，输出每个人得到的面包数量，并采用“%.1f”格式符保留1位小数，运行结果参见图3.19。

每次改变输入的面包数量和家人的人数，计算结果也会随之改变。你是否能感觉到，计算机好像比计算器更灵活、更人性化了一些呢？

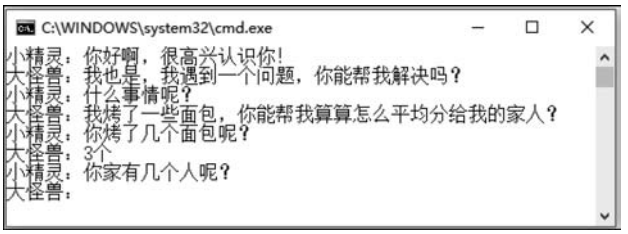

图 3.18　对话等待新的输入

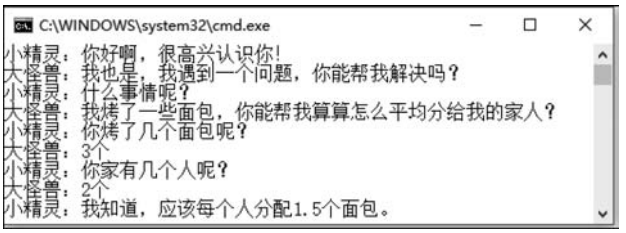

图 3.19　对话输出结果

3.3　通过文件与计算机交流

如果想保存我们与计算机之间的对话内容，可以将交互的数据以计算机文件的形式存储在计算机的硬盘中。有时，需要输入到程序中的数据量很大，通过键盘的方式输入数据很烦琐。可以先将要输入的数据保存在文件中，让程序从文件中读取数据。有时，我们想查看程序中的数据而又不想再次运行程序，此时也可以将程序中的数据保存在文件中。这样只要打开文件就可以查看到相关数据了。

3.3.1　记录我们与计算机之间的对话

在计算机的硬盘中，数据一般都是以文件的方式存储的。**文件是指存储在外部介质上的数据的集合，它是操作系统管理数据的一种方式**。如果要访问存放在硬盘上的数据，一般是先根据文件的名称找到文件，再按照文件中数据存储的格式读取数据。文件中的数据存储格式不同，访问它们的方式也不相同。

例如，有的文件是以字符的方式存储数据，只要打开文件就可以直接阅读文件的内容。文本文件就是这样的文件，它的扩展名一般是.txt。有的文件是以二进制数字的方式存储数据，当打开文件后我们无法直接阅读文件的内容。C 程序的可执行文件就是这样的文件，它的扩展名一般是.exe。可执行文件存储的内容是二进制的机器指令，而不是由字符组成的可以直接阅读的文本。

C 程序在编辑、编译和运行过程中会涉及两类文件：

(1) 程序文件，它的内容是程序代码，主要包括源程序文件、目标文件、可执行文件。

其中，源程序文件是文本文件，目标文件和可执行文件是二进制文件。

（2）数据文件，它的内容是程序的输入和输出数据，主要包括文本文件和二进制文件。

3.3.2 我们可以阅读的文件

微课 3.4 文本文件

文本文件一般指 ASCII 文件。在文本文件中，文件的内容由字符组成并且按照字符的 ASCII 码进行存储。在计算机中，当用记事本工具打开文本文件后，文件中的数据将按字符类型读取并显示，我们可以直接阅读文件的内容。如果想经常查看程序的输入输出数据，可以使用文本文件来存储程序中的数据。

假如要将“My salary is 240000 dollars a year.”这句英文存储到一个文件中，该如何存储呢？在这句英文中既有字母又有数字，字母只能用它对应的 ASCII 码值存储，但是数字既可以用每个数字所对应的字符的 ASCII 码值来存储，也可以按照一个整数类型进行存储。如果将“240000”看作一串字符，那么它包含 6 个字符，需要占用 6 字节的存储空间。如果将“240000”看作一个整数，用一个 int 变量来存储它，那么只需要占用 4 字节的存储空间。

无论是字母还是数字都以字符方式存储的文件就是文本文件。文本文件更适合于存储用户需要阅读的数据。计算机只要逐个字节地读取文本文件中的数据，并按照字符的 ASCII 码值显示数据就可以了。需要计算的数据不适合存储于文本文件中。例如，从文本文件中读取“240000”并进行运算。一方面，计算机在读字符时需要判断如何完整地读出“240000”这串字符；另一方面，还需要把这串字符转换成一个整数类型，这是十分烦琐的工作。

如果将“240000”按照 int 类型存储，它占用 4 字节。将这 4 字节的数据存储在文件中，当需要从文件中读取“240000”时，只需要再读取 4 字节的数据，存储到指定的 int 类型的变量中就可以完成数据的读取。**这种按照数据的字节存储的文件就是二进制文件或者数据文件**。将数“240000”按照字符和 int 类型分别存储在文本文件和二进制文件中，它所占用的字节数是不同的，参见图 3.20。

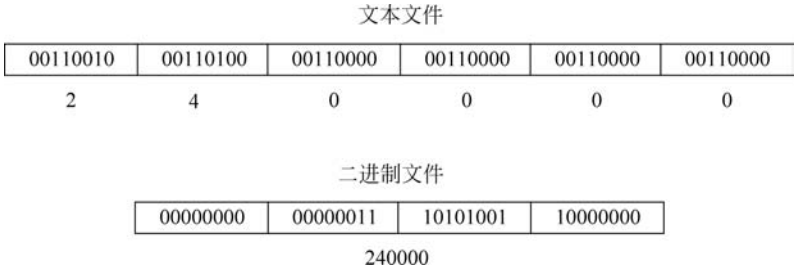

图 3.20 文本文件和二进制文件存储格式示意图

对比图 3.20 中的二进制文件和文本文件的存储大小可以发现，存储数“240000”的二进制文件比文本文件所需的空间要少，更加节省存储空间。但是，当我们用记事本打开二进制文件时，很多时候会显示乱码，这是因为用记事本打开文件时对存储的每个字节是按照字符解释的，遇到无法用可见 ASCII 字符显示的数值的时候就会产生乱码。如果为了方便人们进行阅读，则应该用文本文件来存储数据。

如果要实现在程序与文本文件之间交互数据，还需要了解一些关于文件的相关知识，包括文件的名称、文件缓冲区、文件指针和文件的访问。

1. 文件的名称

当我们要访问计算机中的一个文件的时候，首先需要告诉计算机该文件的名称。**文件名称是用户和计算机用于识别和引用文件的唯一标识**。我们每个人都有一个姓名，姓名由字符组成，人们通过姓名来区别彼此，但是姓名并不是区别我们的唯一标识。通过身份证中的一串数字可以唯一地标识每个人。但是由于数字不容易记住，在生活中我们还是习惯用姓名来区别每个人。当然也可以采用一串数字为文件命名，但是不容易记，因此我们还是选择用字符来命名文件的名称。

用字符来命名文件，文件的名称有可能会重名，这样就无法区分两个重名的文件。在学校中，一个班级里面一般不会有重名的学生，学校在分班的时候会避免将他们分配到同一个班级中。但是在两个不同的班级中可以有重名的学生。在学校中，区别重名的学生，会在他们的姓名前加上班级。例如，一班的张三、二班的张三。

在计算机中，文件名称也是运用了类似的命名方式，采用“文件的存储位置＋文件名”的组合方式对文件进行命名。**文件的存储位置也叫文件路径，它是文件在辅助存储设备中的位置，由“盘符＋文件夹”组成**。这里面的盘符可以类比成学校，文件夹可以类比成年级、班级等层级，而文件名则可以类比成学生的姓名。对文件名进行命名，除了希望文件名中能包含它所存储的数据内容的说明信息以外，还希望能包含数据的存储格式信息。例如，是文本文件还是二进制文件，是 C 源文件还是头文件等。因此，我们一般采用**“文件主干名.文件扩展名”**的方式对文件名进行命名，其中文件主干名说明了文件的内容，文件扩展名说明了文件的类型。一个完整的文件名称示例参见图 3.21。

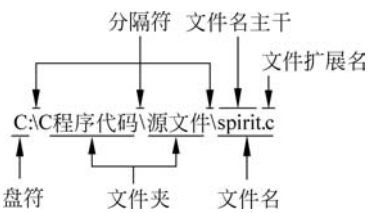

图 3.21　文件名称示例

在计算机中，找到 spirit.c 文件后，在鼠标右键快捷菜单中选择“属性”命令可以查看文件的类型和存储位置信息，参见图 3.22。可以看出，该文件的文件类型是“C Source”，即 C 源文件。文件的存储位置是“C:\C 程序代码\源文件”。文件的大小是 15 862 字节。

在 C 程序中，我们可以创建和访问文本文件和二进制文件。文本文件的扩展名一般是.txt，而二进制文件的扩展名一般是.dat。文件名中也可以没有扩展名，也可以定义与它内容格式不一致的扩展名。虽然这样做合法，但不建议这样做。扩展名只是为用户和操作系统在查看和检索不同类型文件时提供一定的方便，并不能在创建文件名时通过指定扩展名来决定文件中的数据存储格式。

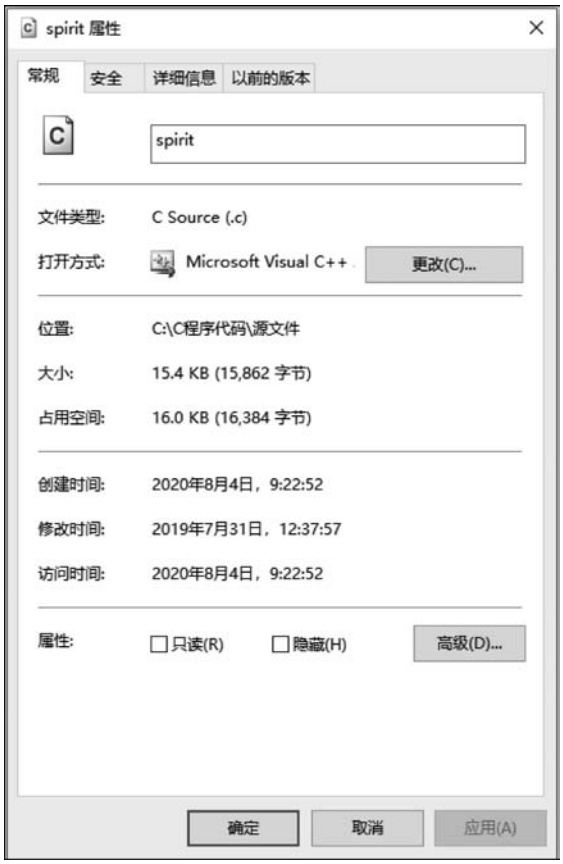

图 3.22　文件属性页

例如，我们命名了文本文件 Readme.txt，但是在创建时却以二进制文件的格式创建与存储数据，那么这个文件的类型就是二进制文件，而不是文本文件。另外，文件名的命名不需要严格遵循 C 语言中标识符的命名规则，不同的操作系统对文件名的命名有不同的规定。

2. 文件的缓冲区

在内存中，为文件开辟了专门的数据存储区域叫作文件缓冲区。如果通过程序访问硬盘中的文件，需要将文件中的一批数据先读入缓冲区，再从缓冲区中将数据送到程序中的变量。如果需要将程序中的数据输出到文件中，则需要先将数据写入到文件缓冲区中。当文件缓冲区中的数据存满后，操作系统会自动地将文件缓冲区中的数据写入硬盘的文件中。文件缓冲区可以减少操作系统对硬盘的读写次数，从而提高计算机的工作效率，但是它也有缺点。如果程序正在对文件的内容进行修改，而此时出现了停电等异常故障，那么在文件缓冲区中的文件数据可能还没有输出到硬盘的文件中，从而产生文件数据丢失的问题。

3. 文件的指针

在内存中,对文件缓冲区的管理比较复杂,因此专门定义了文件的数据类型 FILE 和对 FILE 类型进行操作的库函数,这些函数的使用需要包含 stdio.h。FILE 类型是由 C 编译系统定义的一种结构体数据类型,不同的 C 编译环境会存在一定的差异。结构体数据类型的概念和相关知识将在第 7 章中进行介绍。

由于文件缓冲区由操作系统进行管理,并且使用 FILE 类型存储与管理文件数据的过程相对复杂,因此在对文件数据进行操作时,通常先定义一个 FILE 类型的指针变量,然后调用文件操作的库函数,创建 FILE 类型结构体并返回一个指针变量,再通过库函数和 FILE 指针变量执行读取或写入文件数据的操作。

例如,下面定义了两个 FILE 类型的指针变量:

```
FILE *file_pointl, *file_point2;
```

4. 文件的访问

当要访问一个文件时,这个文件应该在硬盘中存在。如果文件不存在,那么需要先创建这个文件。操作系统在创建文件时需要完成两个工作:一个是在文件缓冲区中创建文件,另一个是将文件缓冲区中的文件存储到硬盘中。

在程序中,对文件的访问过程一般包括创建文件、打开文件、读写文件和关闭文件 4 个阶段。

1) 创建文件

创建文件的工作可以通过调用 fopen 函数来完成,它的一般形式如下:

```
FILE *fopen(文件名称或文件名,文件访问方式)
```

文件名称或文件名:它是一个字符串。如果使用文件名称,系统会根据文件名称中的文件路径在相应的位置创建文件并按照文件名对文件进行命名。如果仅使用了文件名,由于没有指定文件的路径,系统默认在可执行程序的文件夹内创建文件。

文件访问方式:它也是一个字符串,对文件的读写方式进行说明。

例如,

```
fopen("c:\\程序\\程序文档.txt","w");          //"w"是英语单词 write 的首字母
```

在操作系统中,文件的名称是“c:\程序\程序文档.txt”。但是在 C 语言中分隔符“\”用于字符转义,必须采用转义字符“\\”才能表示字符“\”,因此在程序代码中,该文件的名称需要书写为“c:\\程序\\程序文档.txt”。

"w"是一个字符串常量,其含义是创建文本文件。如果要创建二进制文件,则需要用字符串"wb",其中,b 是英语单词 binary(二进制)的首字母。

在调用 fopen 函数来创建文件的时候,并不一定都会成功。例如,如果文件名称中包含的文件路径错误会导致创建文件失败,但是此时程序在运行中并不会报错。我们需要

根据 fopen 函数的返回值来判断是否成功地创建了文件。如果 fopen 函数调用成功，它会返回文件缓冲区中文件数据的指针值；如果调用失败，则返回的 FILE 指针的值为 0，即空指针值。

【例 3.5】 在计算机中创建文本文件“c:\程序\程序文档.txt”，假设在计算机 C 盘根目录中已经存在“程序”文件夹，但不存在“程序 1”文件夹。

问题分析：先定义两个文件指针，然后调用 fopen 函数打开文件进行创建，最后输出函数调用返回的指针值来判断是否成功创建文件。

程序代码如下：

```
1   #include <stdio.h>
2   int main( )
3   {
4       FILE *file_point1, *file_point2;
5       file_point1 = fopen("c:\\程序\\程序文档.txt","w");
6       file_point2 = fopen("c:\\程序 1\\程序文档.txt","w");
7       printf("file_point1 = %d,file_point2 = %d\n",file_point1,file_point2);
8       return 0;
9   }
```

在第 4 行代码中，定义了两个 FLIE 指针变量，用于存储 fopen 函数的返回值。在第 5 行代码中，fopen 函数调用成功，指针变量 file_point1 获得文件缓冲区中的文件数据地址。在第 6 行代码中，由于文件名称的路径存在错误，fopen 函数调用失败，指针变量 file_point2 的值为 0。

程序运行结果如图 3.23 所示。

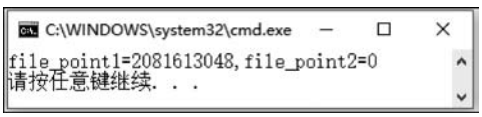

图 3.23 输出结果

2）打开文件

当创建完文件后，可以选择不同的文件打开模式对文件中的数据进行访问。访问文件的操作一般有 3 种方式：第一种是只读取但不更改文件中的内容，即读操作；第二种是只更改但不需要读取文件中的内容，即写操作；第三种是既要读操作也要写操作。针对以上 3 种不同的文件访问方式，fopen 函数分别提供了相对应的文件打开模式。

（1）文件的只读访问模式。

```
fopen(文件名称或文件名, "r");          //文本文件的只读访问模式
fopen(文件名称或文件名, "rb");         //二进制文件的只读访问模式
```

"r"是单词 read 首字母。通过只读访问模式打开文件时被访问的文件必须已经存在，否则调用 fopen 函数将会失败。文件缓冲区分为输入文件缓冲区和输出文件缓冲区。当以只读访问方式打开文件时，文件数据只会加载到输入文件缓冲区，而不会加载到输出文件缓冲区，因此无法对文件数据进行更改。

（2）文件的只写访问模式。

```
fopen(文件名称或文件名, "w");          //文本文件的只写访问模式
```

```
fopen(文件名称或文件名, "wb");                    //二进制文件的只写访问模式
```

"w"和"wb"访问模式都是先创建一个新文件,然后对该文件执行写操作。假设这个文件已经存在,则会先删除该文件,然后再创建一个新文件。如果想保留已有文件中的数据,则不能使用该模式打开文件,可以选择以追加访问模式"a"打开文件。字母“a”是单词 append 的首字母。以追加访问模式打开文件时,若文件不存在,则会建立该文件;如果文件存在,写入的数据会被加到文件尾,即文件原先的内容会被保留。

```
fopen(文件名称或文件名, "a");                     //文本文件的只追加写访问模式
fopen(文件名称或文件名, "ab");                    //二进制文件的只追加写访问模式
```

当以只写访问模式打开文件时,文件数据只会加载到输出文件缓冲区,而不会加载到输入文件缓冲区,因此无法对文件的数据内容进行读取操作。

(3) 文件的读写访问模式。

有时既需要读取文件的数据,又需要更改文件的数据,此时就需要使用读写访问模式打开文件。在只读或者只写访问模式标记中加入符号“+”就可以将原来的只读或者只写扩展为同时读写访问模式标记。例如,

```
FILE * file_point1 = fopen("c:\\程序\\程序文档.txt","r + ");
```

当 fopen 函数调用成功后,可以将“程序文档.txt”文件的访问模式设置为可读写访问,并且将文件缓冲区中的文件数据的地址返回给指针变量 file_point1。通过 file_point1 变量可以利用读写文件的函数对文件数据进行读写操作。

3) 读写文件

在利用 fopen 函数设置完文件的访问模式后,就可以使用 fprintf、fscanf 等函数对文本文件中的数据进行写读操作了。如果已经熟练地掌握了 printf 函数和 scanf 函数的使用,就会轻松地掌握 fprintf 函数和 fscanf 函数的使用。

(1) 向文件中写入数据。

printf 函数的功能是将程序中的数据输出到显示器上,而 fprintf 函数则是将程序中的数据输出到文件中。它们的功能非常相似,因此 fprintf 函数只比 printf 函数多了一个文件指针参数,文件指针参数用于说明将数据输出到指定的文件中。两个函数格式对比如下:

```
int printf(格式字符串, 输出数据列表)
int fprintf(文件指针, 格式字符串, 输出数据列表)
```

如果 fprintf 函数执行成功,则返回输出到文件中的字符总数;如果函数执行失败,则返回一个负数。

【例 3.6】 从键盘中连续输入 3 个字符,并把这些字符以及它们的顺序信息写入文件中。

程序代码如下:

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

```
2   int main( )
3   {
4       char ch;                                          //存储从键盘输入的字符
5       int num = 0;                                      //存储序号信息
6       FILE * file_point;                                //定义文件指针变量
7       file_point = fopen("c:\\程序\\程序文档.txt","w"); //创建并打开新文件
8       scanf("%c",&ch);                                  //读入第 1 个字符
9       num++;                                            //第 1 个字符的序号为 1
10      fprintf(file_point,"%c %d\n",ch,num);             //将字符和序号写入文件
11      scanf("%c",&ch);                                  //读入第 2 个字符
12      num++;                                            //第 2 个字符的序号为 2
13      fprintf(file_point,"%c %d\n",ch,num);             //将字符和序号写入文件
14      scanf("%c",&ch);                                  //读入第 3 个字符
15      num++;                                            //第 3 个字符的序号为 3
16      fprintf(file_point,"%c %d\n",ch,num);             //将字符和序号写入文件
17      fclose(file_point);                               //关闭文件
18      return 0;
19  }
```

程序运行后,“c:\程序\程序文档.txt”文件中的内容参见图 3.24。

图 3.24 输出到文件后的数据

(2) 从文件中读取数据。

fscanf 函数可以将文件中的数据读入到变量中。fscanf 函数比 scanf 函数多了一个文件指针参数,两个函数对比如下:

int scanf(格式字符串,输入变量地址列表)
int fscanf(文件指针,格式字符串,输入变量地址列表)

如果 fscanf 函数读取数据成功,则返回读取数据的个数;如果函数执行失败,则返回一个负数。

【例 3.7】 从文本文件中读取数据并将其显示在屏幕上。

在【例 3.6】中,通过程序在“程序文档.txt”文件中存储了一些字母及其对应的序号。现在需要将它们从文件中读入到程序的变量中,然后显示在屏幕上。

程序代码如下:

```
1   #include <stdio.h>
2   int main( )
3   {
4     char ch,backspace;                                  //存储字母和换行符
5     int num = 0;                                        //存储序号
6     FILE * file_point;
7     file_point = fopen("c:\\程序\\程序文档.txt","r");   //以只读方式打开文件
8     fscanf(file_point,"%c%d%c",&ch,&num,&backspace);    //从文件读第 1 行数据
9     printf("%c %d%c",ch,num,backspace);                 //将数据输出到屏幕上
10    fscanf(file_point,"%c%d%c",&ch,&num,&backspace);    //从文件读第 2 行数据
```

```
11    printf("%c %d%c",ch,num,backspace);
12    fscanf(file_point,"%c%d%c",&ch,&num,&backspace); //从文件读第3行数据
13    printf("%c %d%c",ch,num,backspace);
14    fclose(file_point);
15 }
```

在第8行中,“fscanf(file_point,"%c%d%c",&ch,&num,&backspace);”语句中格式字符串是"%c%d%c",对应了图3.24中文件的第一行数据“A 1”,在这行数据的最后有一个换行符。%c对应了读取字符A。%d对应了按整数读取字符1,虽然字符A和字符1中间有一个空格字符,但是%d格式符可以忽略空格字符。最后一个%c对应了读取换行符,这样在第10行中fscanf可以正确地读取到文件中的第二行数据“B 2”。

程序运行结果如图3.25所示。

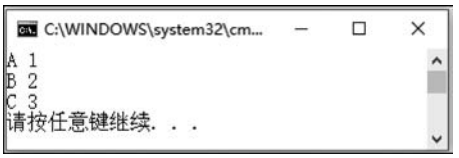

图3.25 输出结果

4) 关闭文件

当读写文件完成后,需要释放文件缓冲区的空间。如果对文件进行了修改,还需要将输出缓冲区的数据写入到硬盘的文件中,则需要调用fclose函数,它的一般形式为:

```
int fclose(文件指针)
```

fclose函数与fopen函数一般成对使用。只要调用了fopen函数将文件数据加载到文件缓冲区中,就需要在文件访问结束后,使用fclose函数释放文件缓冲区相关内存资源。当对文件的内容进行修改后,并想保存到硬盘中时则必须调用fclose函数,否则系统不会将输出缓冲区中的文件内容同步到硬盘的文件中。

3.3.3 我们无法阅读的文件

二进制文件是我们无法阅读的文件。在内存中,数据是以二进制的编码方式存储,如果不将其转换成ASCII码而是直接将二进制数据输出到文件中,该文件就是二进制文件。当我们不需要阅读程序中的数据,而只是需要保存程序中的数据,或者只是需要将文件中的数据输入到程序中时,可以采用二进制文件的方式存储数据。它省去了二进制编码到ASCII码间的数据转换过程。

微课3.5 二进制文件

二进制文件的访问过程与文本文件基本相同,但是它在文件创建与打开方式、文件的读写两个方面与文本文件有所不同。

1. 文件的创建与打开方式

创建与打开二进制文件也需要调用fopen函数,但是在“fopen(文件名称或文件名,访问文件方式)”中“访问文件方式”中比文本文件的“访问文件方式”都增加了字符“b”。文本文件和二进制文件的打开方式参见表3.5。

表 3.5 文本文件和二进制文件的打开方式

文件打开方式	文本文件的访问方式	二进制文件的访问方式
打开文件,只能读数据	r	rb
创建新文件,只能写数据	w	wb
创建新文件,追加写数据	a	ab
既可以读数据,也可以写数据	r+	rb+
创建新文件,既可以读数据,也可以写数据	w+	wb+
创建新文件,既可以读数据,也可以追加写数据	a+	ab+

2. 文件的读取

C 语言中提供了 fread 函数和 fwrite 函数来完成二进制文件的读操作和写操作。当从硬盘向文件缓冲区读取二进制文件时,不需要考虑二进制文件中数据的类型,只需要将一组二进制数据按照字节个数原封不动地、不加转换地复制到文件缓冲区,将文件缓冲区的数据写入磁盘也是如此。

1) 向文件中写入数据

fwrite 函数的一般形式为:

```
int fwrite(void * buffer, int size, int count, FILE * fp)
```

它可以将程序中的数据写入文件缓冲区。如果要完成这个操作,需要告诉计算机变量的指针(地址)、变量的数据类型长度、变量的数量和文件指针。这样设计的目的是可以将一组连续存储并且数据类型相同的变量中的数据写入到文件缓冲区中。

buffer:指针变量,存储要写入文件中的数据的地址。void 是"无类型"数据类型,void * 是指无类型指针类型。因为是按字节读写,所以字节中存储的数据的类型不再重要,buffer 指针变量的数据类型也不再重要,它可以是任何一种数据类型,因此指定了"无类型"的 void 类型作为 buffer 指针变量的类型。

size:要写入文件的每个数据的数据类型长度。

count:要写入文件的数据个数。

fp:要写入数据的文件的指针值。

fwrite 函数根据变量 buffer 中的数值获得需要写入到文件缓冲区中的数据的第一个字节的地址,根据 size * count 的大小获得应该复制多少字节,根据 fp 获得文件缓冲区中用于存放上述数据的存储空间的第一个字节的地址。如果 fwrite 函数调用成功,则函数将返回写入文件中的数据个数;如果失败,则返回数值 0。

【例 3.8】 将程序中变量的数据写入二进制文件中。

程序代码如下:

```
1   #include <stdio.h>
2   int main( )
3   {
4       char ch = 'a';
```

```
5       int grade = 90;
6       FILE * file_point;
7       file_point = fopen("c:\\程序\\程序文档.dat","wb");  //创建并打开二进制文件
8       fwrite(&ch,1,1, file_point);                        //写入 1 个 1 字节的字符
9       fwrite(&grade,4,1, file_point);                     //写入 1 个 4 字节的整数
10      fclose(file_point);
11      return 0;
12  }
```

第 7 行语句创建了二进制文件“c:\程序\程序文档.dat”。第 8 行语句将变量 ch 中的数据写入了文件缓冲区,其中参数“&ch”是变量 ch 的地址,第一个“1”是指 char 类型的字节数是 1,第二个“1”是指 1 个 char 变量。第 9 行语句是将变量 grade 中的数据写入文件缓冲区,参数“&grade”是变量 grade 的地址,“4”是指 int 类型的字节数是 4,“1”是指 1 个 int 变量。

程序运行后,会创建二进制文件“程序文档.dat”,参见图 3.26。用记事本程序打开文件后,可以看到文件中的内容是“aZ”,而不是“a90”,是出现了错误吗?并不是。

图 3.26　二进制文件打开结果

通过 fwrite 函数将变量 ch 和 grade 中的数据写入文件,总计写入了 5 个字节。“aZ”是 2 个字符,占用了 2 字节。当将光标移到“Z”后面,还可以移动 3 次,说明在“Z”后面还有 3 个不可见的字符。也就是说,我们的确将变量 ch 和 grade 总计 5 字节的数据写入了文件。那为什么不是“a90”,而是“aZ”呢?这是因为用记事本程序打开二进制文件时,程序会读取文件中的二进制数据,并将每个字节按照 ASCII 码值显示相应的字符。在二进制文件中对字符'a'进行存储,存储的是 1 字节的'a'的 ASCII 值 97,因此读取 1 字节,并按 97 显示字符,仍然是字符'a'。在二进制文件中采用 int 类型存储 90,第一个字节存储的是 90,后几个字节是 0。因为 90 是字符'Z'的 ASCII 码,因此显示'Z',其他 3 个字节是空字符,无法显示出来。这个例子说明,虽然可以以文本文件的方式打开二进制文件,但无法正确地显示二进制文件中的数据。

2）从文件读入数据

fread 函数可以实现从文件缓冲区将二进制文件中的数据读给变量，它的一般形式为：

```
int fread(void * buffer, int size, int count, FILE * fp)
```

fread 函数的参数的作用与 fwrite 函数相同。如果函数执行成功则返回读出数据的个数，否则返回数值 0。

【例 3.9】 从二进制文件中顺序读取数据并写入变量中。

在【例 3.8】中，利用 fwrite 函数向“程序文档.dat”中写入了 char 变量 ch 中的字符'a'和 int 变量 grade 中的数据“90”共 5 个字节的数据。现在要使用 fread 函数从“程序文档.dat”中读取上述数据，并将这些数据再写入变量 ch 和变量 grade 中。

程序代码如下：

```
1   #include <stdio.h>
2   int main( )
3   {
4       char ch;
5       int grade;
6       FILE *file_point;
7       file_point=fopen("c:\\程序\\程序文档.dat","rb");  //以只读方式打开二进制文件
8       fread(&ch,1,1,file_point);                        //读入 1 个 1 字节的字符
9       fread(&grade,4,1,file_point);                     //读入 1 个 4 字节的整数
10      fclose(file_point);
11      printf("ch=%c,grade=%d\n",ch,grade);              //将变量数据输出到显示器
12      return 0;
13  }
```

第 7 行语句是以只读的方式打开二进制文件“程序文档.dat”。第 8 行语句是将文件缓冲区中的第 1 字节的内容读取给变量 ch。第 9 行语句是将文件缓冲区中第 2～5 字节的内容读取给变量 grade。第 11 行语句是将变量 ch 和 grade 中的值输出到显示器，如果变量 ch 中存储了字符'a'，变量 grade 中存储了整数 90，则说明读取代码正确。

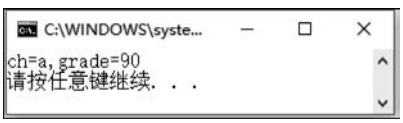

图 3.27 输出结果

程序运行结果如图 3.27 所示。

3.3.4 顺序读写与按需读写

有时，我们需要让程序按照数据存储的物理顺序读取文件中的数据，即从文件头部开始，顺序地读取文件中的每一个数据。数据的读写顺序与数据在文件中存储的物理顺序一致。前面的例子都是采用了顺序读写文件的方法。

有时，我们需要让程序按照指定的位置读取文件中的某些数据，或者向文件中插入、

替换某些数据，而不是从头开始读写，这就是按需读写。如何实现按需读写文件呢？首先需要了解文件缓冲区的管理方式。

缓冲文件系统对文件缓冲区进行读写管理。文件缓冲区是一段连续的物理存储区域，它的示意图参见图 3.28。缓冲文件系统为每个文件缓冲区设置了文件开始位置、文件末尾位置和读写当前位置 3 个标记。当打开文件时，读写当前位置的标记一般都指向文件开始位置，当按追加模式打开文件时，读写当前位置标记指向文件末尾位置。当读取或者写入一个数据后，当前读写位置标记就会移到这个数据后面的一个字节的位置。当读写当前位置标记移到了文件尾处，则文件读取结束。因此，通过将读写当前位置标记指向要读写数据的位置，就可以实现按需读写文件中的数据了。

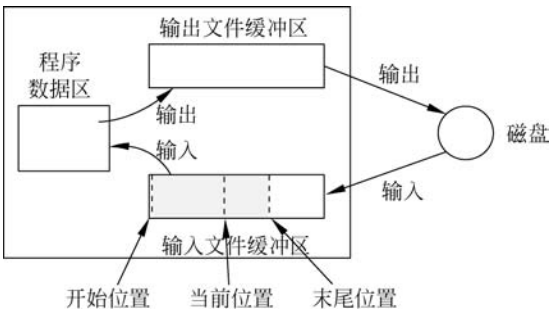

图 3.28　文件缓冲区

1. 移动到指定位置

fseek 函数可以设置读取当前位置标记，fseek 函数的一般形式为：

```
int fseek(文件类型指针,位移量,起始点)
```

文件类型指针：要读写的文件的指针。

起始点：读写文件数据的起始位置的参照点。它有 3 种选择：文件开始位置、文件当前位置和文件末尾位置，分别用符号常量 SEEK_SET、SEEK_CUR、SEEK_END 表示，或者直接使用数字 0、1、2。

位移量：以起始点为基点，向前（向文件末尾方向）或向后（向文件开始方向）移动的字节数，正数表示向前移动的字节数，负数表示向后移动的字节数。

如果 fseek 函数调用成功，函数返回值是 0；如果调用失败，则返回一个非 0 值。

例如，有如图 3.29(a)所示的文件位置标记的初始状态，若执行“fseek(fp,10,0);”语句，则表示将文件位置标记向前移动到距离文件开头 10 字节的位置，如图 3.29(b)所示；若继续执行“fseek(fp,5,1);”语句，则表示将文件位置标记继续向前移动 5 字节，此时当前位置如图 3.29(c)；若继续执行“fseek(fp,－10,2);”语句后，则表示将文件位置标记向后移动到距离文件末尾 10 字节的位置，如图 3.29(d)所示。

【例 3.10】 从键盘读入 3 个字符写入二进制文件中，再读取文件中的第 2 个字符并将它输出到屏幕上。

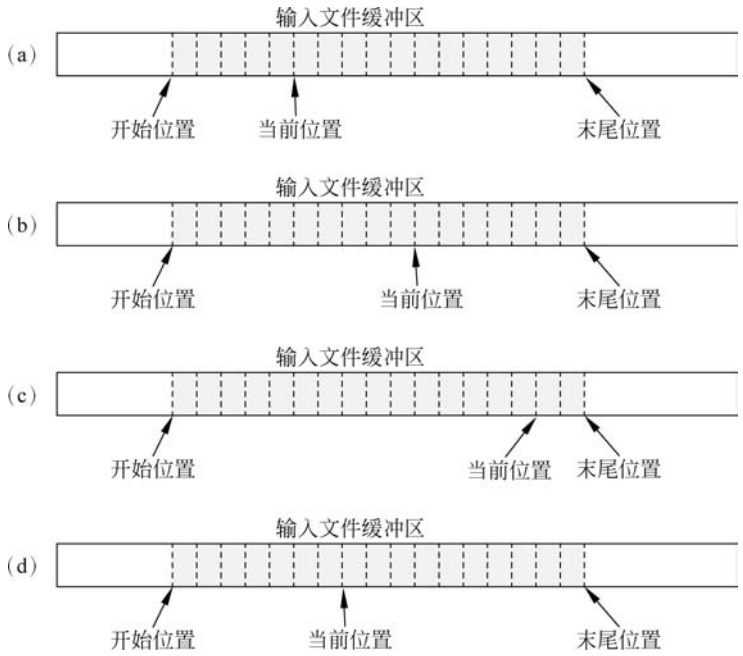

图 3.29　文件指针位置移动示意图

程序代码如下：

```
1   #include <stdio.h>
2   int main( )
3   {
4       char ch1,ch2,ch3;
5       FILE *file_point;
6       scanf("%c%c%c",&ch1,&ch2,&ch3);                          //从键盘读取 3 个字符
7       file_point=fopen("c:\\程序\\程序文档.dat","wb+");  //以可读写方式创建文件
8       fwrite(&ch1,1,1,file_point);                             //将 ch1 数据写入文件
9       fwrite(&ch2,1,1,file_point);                             //将 ch2 数据写入文件
10      fwrite(&ch3,1,1,file_point);                             //将 ch3 数据写入文件
11      fseek(file_point,1,SEEK_SET);                            //从文件头向前移动 1 字节
12      fread(&ch1,1,1,file_point);
13      fclose(file_point);
14      printf("ch1=%c\n",ch1);
15      return 0;
16  }
```

程序的运行结果如图 3.30 所示。

执行第 6 条语句，从键盘连续输入“abc”3 个字符。

执行第 7 行语句以"wb+"方式打开文件，实现了创建新文件“程序文档.dat”，并且可对该文件进行读写操作。

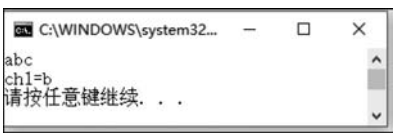

图 3.30　输出结果

第 11 行语句将 fseek 函数的起始点设置为 SEEK_SET，位移量设置为 1，即文件位置标记向前移动到距离文件开头 1 字节的位置。第 12 行语句调用 fread 函数读取 1 字节的数据到变量 ch 中。

第 8～11 行语句执行后，文件头、文件尾和文件当前位置标记变化参见图 3.31。

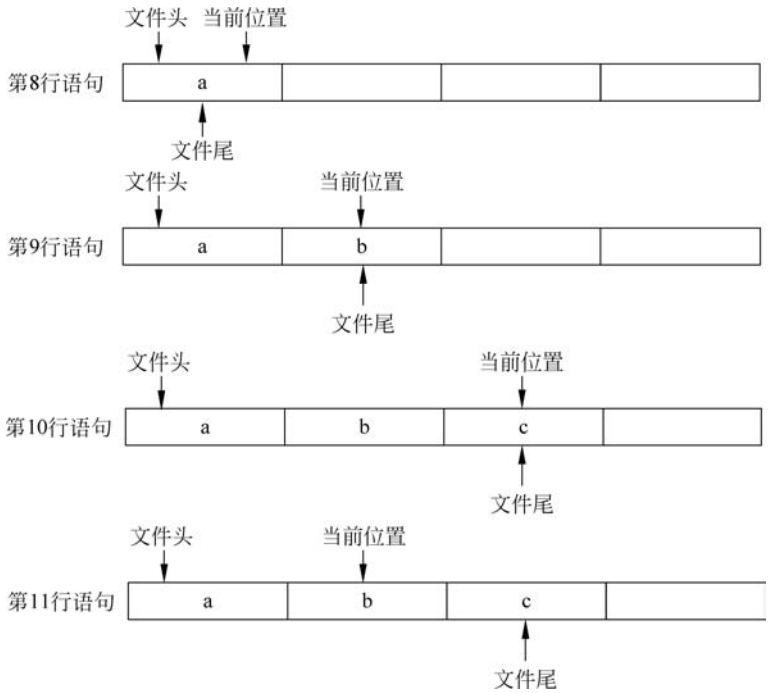

图 3.31　文件指针位置变化示意图

由于第 10 条语句执行完后，当前位置标记和文件尾位置标记相同，因此第 11 行语句 fseek(file_point,1,SEEK_SET)也可以用 fseek(file_point,－1,SEEK_END)替换，即从文件末尾向文件头方向移动一个字节，或者替换为 fseek(file_point,－1,SEEK_CUR)，即从当前位置向文件头方向移动一个字节。

在对文件进行读写操作的时候，我们可能会遇到一种情况：刚刚使用 fprintf 语句或者 fwrite 语句将数据写入文件后，立即想使用 fscanf 语句或者 fread 语句将刚写入的数据再从文件中读取出来，看一看是否已经将数据正确地写入了文件中，但发现根本读不到刚刚写入文件中的数据，这是为什么呢？这是因为当写入数据后，当前位置标记指向了刚写入的数据的最后一个字节。此时再读取数据，会从当前位置标记的下一个字节开始读取数据，因此读不到刚写入的数据。要解决这个问题，可以用 fseek 函数重新设置当前读取位置。

2. *移动到文件开头*

rewind 函数可以将当前位置标记移动到文件头，实现从文件头标记的位置处开始读取数据。rewind 函数的一般形式为：

```
void rewind(文件类型指针)
```

当然也可以使用“fseek(文件类型指针,0,SEEK_SET)”语句将当前位置移动到文件头,但是不如 rewind 函数简洁。

无论是顺序读写还是按需读写文件,文件当前位置是否正确决定了我们是否能够正确地读写文件中的数据,大家需要特别关注它。

3.3.5 * 文件读写的出错问题

在读写文件的过程中,有可能会产生一些错误。通过以下两种途径,可以发现在文件操作过程中产生的绝大部分错误。

1. 检查文件操作函数的返回值是否正确

在打开文件和关闭文件时,需要使用 fopen 函数和 fclose 函数。在读写文件时,需要使用 fprintf 函数、fscanf 函数、fread 函数和 fwrite 函数。在调用这些函数后,可以根据函数的返回值,判断函数调用语句是否执行成功,参见表 3.6。

表 3.6 文件操作函数的返回值

函　数	成　功	失　败
fopen	非 0 值	0
fclose	0	−1
fprintf	写入的字符数	−1
fscanf	读取的字符数	−1
fwrite	正整数	0
fread	正整数	0

2. 调用 ferror 函数检查返回值

除了根据各种调用函数的返回值可以判断文件操作是否出现错误以外,还可以使用 ferror 函数来检查文件操作是否有错误。ferror 函数的一般形式为:

```
int ferror(FILE * fp)
```

如果函数的返回值是 0,则表示未出错;如果是非零值,则表示出错。对同一个文件每一次调用输入输出函数,都会产生一个新的 ferror 函数值,因此应当在调用输入输出函数后需要立即检查 ferror 函数的值,否则信息会丢失。

3.3.6 文件合并示例

【例 3.11】 给定 3 个文本文件,要求将这 3 个文件的内容合并到 1 个文件中。其中 3 个文本文件的具体内容[①]如下:

① 文件内容来自百度百科:
https://baike.baidu.com/item/%E4%B8%AD%E5%9B%BD/1122445.[2021-04-27]

文件 input_1.txt 内容：中国，以华夏文明为源泉、以中华文化为基础，是世界上历史最悠久的国家之一。中国各族人民共同创造了光辉灿烂的文化，具有光荣的革命传统。中国是以汉族为主体民族的多民族国家，通用汉语、汉字，汉族与少数民族统称为“中华民族”，又自称“炎黄子孙”“龙的传人”。

微课 3.6　文本合并示例

文件 input_2.txt 内容：中国是世界四大文明古国之一。距今 5800 年前后，黄河、长江中下游以及西辽河等区域出现了文明起源迹象；距今 5300 年前后，中华大地各地区陆续进入了文明阶段；距今 3800 年前后，中原地区形成了更为成熟的文明形态，并向四方辐射文化影响力；后历经多次民族交融和朝代更迭，直至形成多民族国家的大一统局面。20 世纪初辛亥革命后，废除了封建帝制，创立了资产阶级民主共和国。1949 年中华人民共和国成立后，在中国大陆建立了人民民主专政的社会主义制度。

文件 input_3.txt 内容：中国疆域辽阔、民族众多，先秦时期的华夏族在中原地区繁衍生息，到了汉代通过文化交融使汉族正式成型，奠定了中国主体民族的基础。后又通过与周边民族的交融，逐步形成统一多民族国家的局面，而人口也不断攀升，宋代中国人口突破一亿，清代人口突破四亿，到 2005 年中国人口已突破十三亿。

问题分析：为了合并 3 个文件内容，我们需要以写的方式打开一个文件 output_all.txt，然后依次打开上述 3 个文件，读取文件中的内容并写入到 output_all.txt 文件中，最后关闭文件。由于我们不关注文件的具体文字内容，因此可以直接使用二进制格式读取和写入内容。

程序代码如下：

```
1   #include <stdio.h>
2   int main( )
3   {
4       int i, n;
5       FILE *fr = NULL, *fw = NULL;
6       char buffer[2048] = {0};                  //定义字符数组 buffer
7       fw = fopen("D:\\output_all.txt","wb");    //以二进制"写"方式打开 output_all.txt 文件
8       fr = fopen("D:\\input_1.txt","rb");       //以二进制"读"方式打开 input_1.txt 文件
9       n = fread(buffer,1,2048,fr);              //实际读取 n 个字节
10      fwrite(buffer,1,n,fw);                    //写入 n 个字节
11      fclose(fr);                               //关闭 input_1.txt 文件
12      fr = fopen("D:\\input_2.txt","rb");       //以二进制"读"的方式打开 input_2.txt 文件
13      n = fread(buffer,1,2048,fr);              //实际读取 n 个字节
14      fwrite(buffer,1,n,fw);                    //写入 n 个字节
15      fclose(fr);                               //关闭 input_2.txt 文件
16      fr = fopen("D:\\input_3.txt","rb");       //以二进制"读"的方式打开 input_3.txt 文件
17      n = fread(buffer,1,2048,fr);              //实际读取 n 个字节
18      fwrite(buffer,1,n,fw);                    //写入 n 个字节
19      fclose(fr);                               //关闭 input_3.txt 文件
20      fclose(fw);                               //关闭 output_all.txt 文件
21      return 0;
22  }
```

执行第 7 行语句，以"wb"的方式打开，实现了创建新文件 output_all.txt。

执行第 8～11 行语句，以"rb"的方式打开文件，读取文件 input_1.txt 内容存入 buffer 中，其中 2048 表示一次读取 2048 字节，返回值 n 表示实际读取的字节数，然后将实际读取到的数据写入 output_all.txt 文件中。

执行第 12～15 行语句以及第 16～19 行语句，则分别读取 input_2.txt 和 input_3.txt 中的内容写入 output_all.txt 文件中。

最后执行第 20 行语句关闭文件。

3.4 本章小结

我们与计算机之间的交流方式非常重要。如果我们向计算机中输入数据时，采用的数据输入方法不正确，那么计算机就无法正确"理解"我们要处理的数据，从而导致计算错误。当我们要求计算机输出数据时，如果采用的数据输出方法不正确，即使计算机正确地完成了计算任务，我们也无法看到正确的计算结果。因此，在学习 C 语言时，首先要掌握正确的数据输入输出方法。

本章主要介绍了 printf 函数、scanf 函数、fprintf 函数和 fscanf 函数等，熟练掌握这些函数，基本可以满足与计算机的交流需求。除了这些函数，还有其他输入输出函数，例如 getchar 函数、putchar 函数、gets 函数、puts 函数、fgetchar 函数、fputchar 函数、fgets 函数、fputs 函数等。这些函数可以更灵活地完成对字符数据的输入和输出，在后面章节中将陆续介绍这些函数。本章的知识点参见图 3.32。

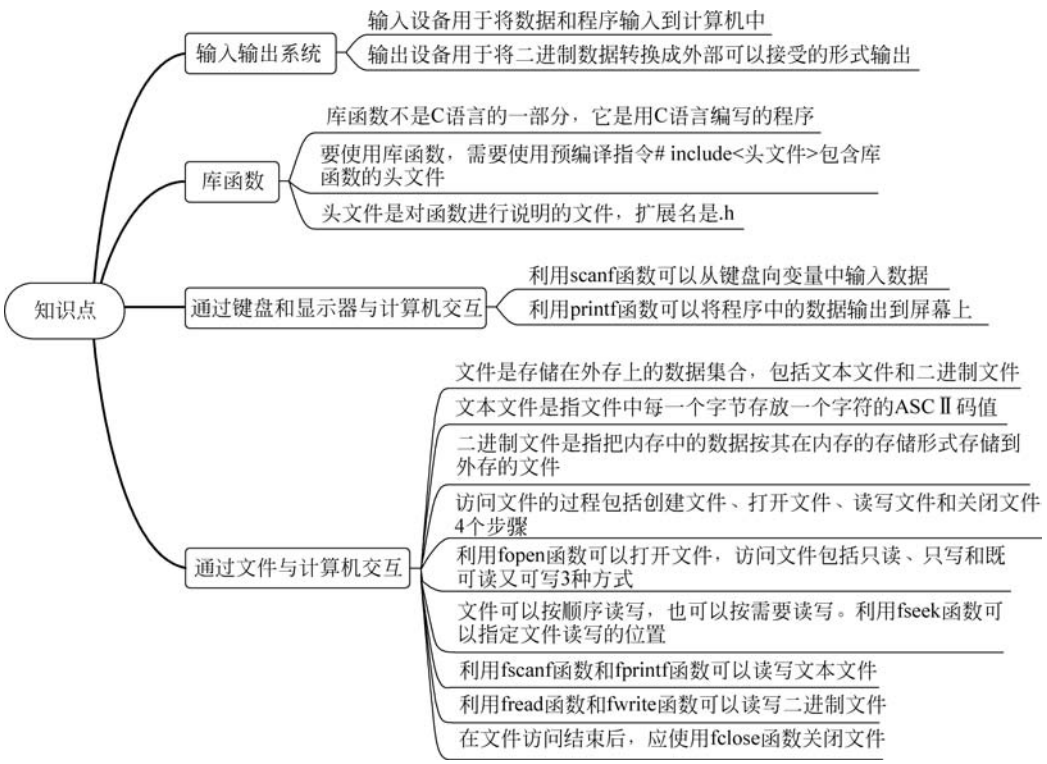

图 3.32 与计算机面对面地交流

3.5 习题

1. 人类与计算机之间进行对话一般包括哪两种方式？各有什么特点？

2. 在C语言中从键盘输入数据和向屏幕输出数据各使用了什么函数？对应函数的主要参数包括哪些？参数的意义是什么？

3. 根据数据的组织形式，文件可分为哪两种不同的类型？各有什么特点？

4. C语言打开文件主要方式有哪些？打开函数的形式是什么？

5. C语言读写文件的主要函数有哪些？它们之间的区别是什么？何种情况下使用？

6. 为什么在文件读写完成后需要关闭文件？不关闭文件会有何影响？

7. 编写一个程序实现接收从键盘依次输入的一个整数a、一个浮点数f、一个整数b(－100＜a,b,f＜100)。要求分3行输出它们的值，其中第一行连续输出a和b(中间无分隔符)；第二行依次输出f、a、b，3个数之间用一个空格分隔，f精确到小数点后两位；第三行依次输出a、f、b，每个数占位10个字符位，包含正负号，右对齐，f精确到小数点后两位，任意两个数之间不添加空格。

输入样例

```
12 34.567 89
```

输出样例

```
    1289
34.57 12 89
+12+34.57+89
```

8. 分析下面的程序：

```
#include <stdio.h>
int main()
{
    int a=2, c=5;
    printf("a= %%d,c= %%d\n",a,c);
    return 0;
}
```

(1) 运行时会输出什么信息？为什么？

(2) 如果将程序第4、5行改为

```
printf("a= %d,c= %d\n",a,c);
```

运行时会输出什么信息？为什么？

9. 分析下面的程序：

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

```
{
    char c1,c2,c3;                                  //①
    c1 = 80;                                        //②
    c2 = 76;                                        //③
    c3 = 65;                                        //④
    printf("c1 = %c,c2 = %c,c3 = %c",c1,c2,c3);
    printf("c1 = %d,c2 = %d,c3 = %d",c1,c2,c3);
    return 0;
}
```

(1) 运行时会输出什么信息？为什么？

(2) 如果将程序的语句②③④分别改为

```
c1 = 180;
c2 = 176;
c3 = 165;
```

运行时会输出什么信息？为什么？

(3) 如果将程序语句①改为

```
int c1,c2,c3;
```

运行时会输出什么信息？为什么？

10. 用下面的 scanf 函数输入数据，使 a=1，b=2，x=3.4，y=5.678，c1='X'，c2='y'。应该如何从键盘上输入数据，才能够保证下面的 scanf 语句能够正确执行？

```
#include <stdio.h>
int main()
{
    int a,b;
    float x,y;
    char c1,c2;
    scanf("a = %d,b = %d",&a,&b);
    scanf("%f%f",&x,&y);
    scanf("%c%c",c1,c2);
    printf("a = %d,b = %d,x = %f,y = %f,c1 = %c,c2 = %c\n",a,b,x,y,c1,c2);
    return 0;
}
```

11. 从键盘输入 5 个大写字母，将其全部转化为小写字母，然后输出到一个磁盘文件 output.txt 中保存。

12. 将自然数 1～9 以及它们的立方写入名为 Cube.txt 的文件中，然后再读出显示在屏幕上。要求分别按文本文件格式和二进制文件格式进行数据的存储和读取，比较写入文件的大小。

13. 任意输入 5 个字符，按二进制格式写入一个文件，再按二进制方式读取并显示在屏幕上。

14. 任意输入 6 个字符，将其写入一个文件中，从文件头开始，读取其中的第 3 个字符和第 5 个字符并显示在屏幕上。