第3章 与计算机面对面地交流

3.1 习题解析

3.1 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？

参考答案：人类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，见图 3.1。

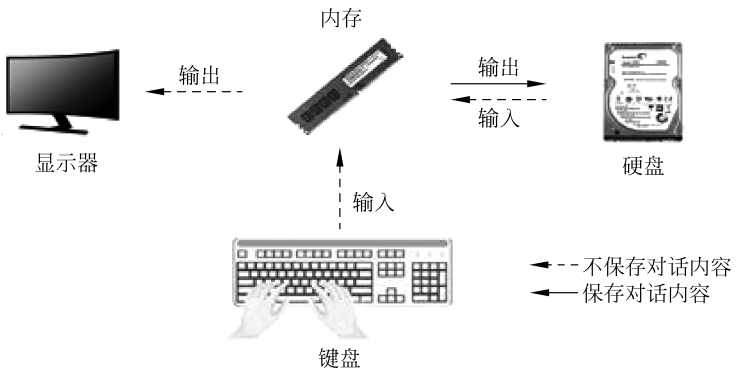

图 3.1 计算机的两种数据输入输出方式示例

(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并存储在程序的变量中。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里面，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，为其他程序运行提供内存空间。此时，我们无法再访问原来程序中的数据。

(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硬盘，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内存中供程序使用，从而不需要通过键盘再次输入数据。

3.2 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？参数的含义是什么？

参考答案：从键盘输入数据使用的函数是 scanf 函数，向屏幕输出数据使用的是 printf 函数。scanf 和 printf 函数的主要参数及其含义如下：

(1) scanf 语句一般形式表示为

```
printf(参数 1,参数 2,…,参数 n);
```

参数 1：格式控制字符串。它是用西文双引号引起来的一个字符串，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。

(2) printf语句一般形式为

```
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。

3.3 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?

参考答案:根据数据的组织形式,文件可分为文本文件和二进制文件。其各自的特点如下:

(1) 文本文件。

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,例如常用的以.txt作为扩展名的文件。

(2) 二进制文件。

以二进制数字的方式存储数据,打开文件后无法直接阅读文件的内容,例如C程序中以.exe作为扩展名的可执行文件。

3.4 C语言打开文件的主要方式有哪些?打开函数的形式是什么?

参考答案:C语言打开文件的方式及函数形式如下:

(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的首字母。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。"rb"中的"b"是单词binary的首字母,表示按二进制方式打开文件。

(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,再创建一个新文件。若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,若被访问的文件不存在,则fopen函数也

将创建一个新文件。

```
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。

有的时候,我们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访问模式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读写访问模式,并且将文件缓冲区中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。

总的来说,创建与打开文件都需要调用 fopen 函数,针对不同类型的文件格式,可以发现,二进制文件与文本文件相比,其“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 3.1。

表 3.1　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,也可以写数据	a+	ab+

3.5　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?分别在何种情况下使用?

参考答案:根据数据的组织形式,C 语言读写文件的函数包括 fprintf、fscanf、fwrite、fread 函数等,其中 fprintf、fscanf 用于文本文件的读写,fwrite、fread 用于二进制文件的读写。函数的具体形式和参数如下:

(1) 文本文件的读写。

fprintf 函数是将程序中的数据输出到文件中,其一般形式为

```
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。从上面的函数形式可以发现,fprintf 函数比 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。

fscanf 函数可以将文件中的数据读入到变量中,其一般形式为

```
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。从上面的函数形式可以发现,fscanf 函数比 scanf 函数多了一个参数文件指针。

(2) 二进制文件读写。

fwrite 函数将数据写入二进制文件,其一般形式为

```
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。这样设计的目的是将一组连续存储并且数据类型相同的变量中的数据写入文件缓冲区中。

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“无类型”指针类型。因为是按字节读写,字节中存储的数据类型不再重要,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。

fread 函数可以实现从文件缓冲区将二进制文件中的数据读入变量,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。

3.6 *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?*

参考答案: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的相关内存资源。在对文件的内容进行修改后,如果想保存到硬盘中,则必须调用 fclose 函数。若不关闭文件,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。

3.7 *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
```

参考程序:

```
#include <stdio.h>                                  // 输入输出操作,需要包含 stdio.h
int main(void)
{
    int a = 0, b = 0;
    double f = 0;

    scanf("%d",&a);
    scanf("%lf",&f);
    scanf("%d",&b);
    printf("%d%d\n",a,b);                           // 两个整数连续输出,不用空格分隔
    printf("%.2f %d %d\n",f,a,b);                   // 按照格式分别输出浮点数和整数
    printf("%+10d%+10.2f%+10d\n",a,f,b);            // 按照格式分别输出浮点数和整数

    return 0;
}
```

解析:

(1) 输入整数采用“%d”格式控制字符,浮点数使用“%lf”或“%f”,分别对应 double 和 float 两种不同精度的浮点型。

(2) 输出格式注意浮点数“%m. nf”的格式控制,其中 m 表示总的字符宽度,n 表示小数点后的有效位数。

(3) 正负号采用“+”或“-”符号,负号表示左对齐,整数“%md”格式控制中 m 表示总的字符宽度。

3.8 分析下面的程序:

```
#include <stdio.h>                                  // 第 1 行
int main(void)                                      // 第 2 行
{                                                   // 第 3 行
    int a = 2,c = 5;                                // 第 4 行
    printf("a=%%d,c=%%d\n",a,c);                    // 第 5 行
    return 0;                                       // 第 6 行
}                                                   // 第 7 行
```

(1) 运行时会输出什么信息? 为什么?

(2) 如果将程序的第 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？

参考答案：(1) 程序编译后会给出警告“warning: too many arguments for format”(意思是有太多的参数需要格式化),运行后在控制台窗口中会输出“a=%d,c=%d”。这是因为两个“%”连续出现时,系统会理解成程序想要输出一个“%”,前一个“%”用来转义第二个“%”,从而无法输出 a 和 c 的值。

(2) 修改后的格式正确,输出“a=2,c=5”。

3.9 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？

参考答案：(1) 程序运行输出“c1=P,c2=L,c3=Ac1=80,c2=76,c3=65”。其中,“%c”表示字符以 ASCII 码符号形式进行输出,80、76、65 在 ASCII 码表中对应字符'P'、'L'、'A'；而“%d”表示按字符的数值进行输出。

(2) 程序输出“c1=?c2=?c3=?c1=-76,c2=-80,c3=-91”。其中,由于字符在内存中仅用 1 字节进行存储,默认为 signed char,即最高位为符号位,因此表示范围为-128~127。当赋予 c1、c2、c3 的值大于 127 时,它超出了有效的数值表示范围。采用%c 输出时,无法将负数转换为 char 类型的 ASCII 码。采用%d 输出时,由于最高位为 1,系统认为它是一个负数的补码,180、176 和 165 以二进制表示分别是-76、-80 和-91 的补码。不同

的编译系统中,按字符方式输入 c1、c2 和 c3,显示结果可能会不一样,但都无法正确显示。

(3) 程序输出"c1=P,c2=L,c3=A c1=80,c2=76,c3=65"。其中,"%c"会将整数按照 ASCII 码值 80、76、65 进行解释,c1、c2、c3 对应输出'P'、'L'、'A',"%d"正常输出整数值。

3.10 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(void)
{
    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;
}
```

参考答案:a=1,b=2(空格)3.4(空格)5.678Xy(回车)。

解析:

(1) 答案不唯一,参考答案中空格可替换为回车或 Tab 键。

(2) scanf 对输入数据的判断按照与格式控制字符串匹配的原则,因此在输入 a=1,b=2 时,两个式子中间的逗号不能用空格替换。

(3) 5.678Xy 输入必须连续,中间不能有空格、回车、Tab 键等,因为这些都会被视作字符。

3.11 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。

参考程序:

```
#include <stdio.h>
int main(void)
{
    FILE* fp;
    char a,b,c,d,e;
    char t;

    fp = fopen("output.txt", "w+");
    scanf("%c%c%c%c%c", &a,&b,&c,&d,&e);
    fprintf(fp, "%c%c%c%c%c", a-'A'+'a',b-'A'+'a',c-'A'+'a',
        d-'A'+'a',e-'A'+'a');
    rewind(fp);
```

```
    fscanf(fp, "%c", &t);
    printf("%c", t);
    fscanf(fp, "%c", &t);
    printf("%c", t);
    fscanf(fp, "%c", &t);
    printf("%c", t);
    fscanf(fp, "%c", &t);
    printf("%c", t);
    fscanf(fp, "%c", &t);
    printf("%c", t);
    fclose(fp);

    return 0;
}
```

解析：

（1）文本文件需要写和读功能时，可采用"w+"打开文件，其中具体符号含义可参考习题 3.4 的解答。

（2）fprintf 函数可以通过格式控制符“%c”写入字符，fscanf 函数可以通过格式控制符“%c”读取字符。

（3）fprintf 和 fscanf 函数可以采用多个“%c”进行多个字符的写入和读取。

（4）可以尝试修改参考程序，通过调用一次 fscanf 函数读取 5 个字符。

3.12　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。

参考程序：

（1）文本文件的输入和输出。

```
#include <stdio.h>
int main(void)
{
    FILE* fp;
    int n, cube, size;

    fp = fopen("Cube.txt", "w+");          // 文本文件格式读写为"w+"
    n = 1;                                 // 依次计算 n 和 n*n*n
    cube = n*n*n;
    fprintf(fp,"%d,%d\n", n, cube);        // 将 n 和 cube 写入文件中
    n = 2;
    cube = n*n*n;
    fprintf(fp,"%d,%d\n", n, cube);
    n = 3;
    cube = n*n*n;
    fprintf(fp,"%d,%d\n", n, cube);
    n = 4;
    cube = n*n*n;
```

```
    fprintf(fp,"%d,%d\n", n, cube);
    n = 5;
    cube = n*n*n;
    fprintf(fp,"%d,%d\n", n, cube);
    n = 6;
    cube = n*n*n;
    fprintf(fp,"%d,%d\n", n, cube);
    n = 7;
    cube = n*n*n;
    fprintf(fp,"%d,%d\n", n, cube);
    n = 8;
    cube = n*n*n;
    fprintf(fp,"%d,%d\n", n, cube);
    n = 9;
    cube = n*n*n;
    fprintf(fp,"%d,%d\n", n, cube);

    rewind(fp);                                  // 更新 fp 指向到文件头位置
    fscanf(fp,"%d,%d", &n, &cube);               // 读取文件,验证结果
    printf("%d,%d\n", n, cube);
    fscanf(fp,"%d,%d", &n, &cube);
    printf("%d,%d\n", n, cube);
    fscanf(fp,"%d,%d", &n, &cube);
    printf("%d,%d\n", n, cube);
    fscanf(fp,"%d,%d", &n, &cube);
    printf("%d,%d\n", n, cube);
    fscanf(fp,"%d,%d", &n, &cube);
    printf("%d,%d\n", n, cube);
    fscanf(fp,"%d,%d", &n, &cube);
    printf("%d,%d\n", n, cube);
    fscanf(fp,"%d,%d", &n, &cube);
    printf("%d,%d\n", n, cube);
    fscanf(fp,"%d,%d", &n, &cube);
    printf("%d,%d\n", n, cube);
    fscanf(fp,"%d,%d", &n, &cube);
    printf("%d,%d\n", n, cube);

    fseek(fp,0L, SEEK_END);                      // 调整 fp 指向文件尾
    size = ftell(fp);                            // 获得文件的大小,采用字节数表示
    printf("文件大小为: %d 字节\n", size);       // 57
    fclose(fp);

    return 0;
}
```

(2) 二进制文件的输入和输出。

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

```
{
    FILE* fp;
    int n, cube, size;

    fp = fopen("Cube.txt", "wb+");          //二进制文件格式读写"wb+"
    n = 1;                                  // 依次计算 n 和 n*n*n
    cube = n*n*n;
    fwrite(&n, sizeof(n), 1, fp);           // 将 n 写入文件中
    fwrite(&cube, sizeof(cube), 1, fp);     // 将 cube 写入文件中
    n = 2;
    cube = n*n*n;
    fwrite(&n, sizeof(n), 1, fp);
    fwrite(&cube, sizeof(cube), 1, fp);
    n = 3;
    cube = n*n*n;
    fwrite(&n, sizeof(n), 1, fp);
    fwrite(&cube, sizeof(cube), 1, fp);
    n = 4;
    cube = n*n*n;
    fwrite(&n, sizeof(n), 1, fp);
    fwrite(&cube, sizeof(cube), 1, fp);
    n = 5;
    cube = n*n*n;
    fwrite(&n, sizeof(n), 1, fp);
    fwrite(&cube, sizeof(cube), 1, fp);
    n = 6;
    cube = n*n*n;
    fwrite(&n, sizeof(n), 1, fp);
    fwrite(&cube, sizeof(cube), 1, fp);
    n = 7;
    cube = n*n*n;
    fwrite(&n, sizeof(n), 1, fp);
    fwrite(&cube, sizeof(cube), 1, fp);
    n = 8;
    cube = n*n*n;
    fwrite(&n, sizeof(n), 1, fp);
    fwrite(&cube, sizeof(cube), 1, fp);
    n = 9;
    cube = n*n*n;
    fwrite(&n, sizeof(n), 1, fp);
    fwrite(&cube, sizeof(cube), 1, fp);

    rewind(fp);                             // 更新 fp 指向到文件头位置
    fread(&n, sizeof(n), 1, fp);            // 读取文件,验证结果
    fread(&cube, sizeof(cube), 1, fp);
    printf("%d,%d\n", n, cube);
    fread(&n, sizeof(n), 1, fp);
    fread(&cube, sizeof(cube), 1, fp);
    printf("%d,%d\n", n, cube);
    fread(&n, sizeof(n), 1, fp);
```

```
    fread(&cube, sizeof(cube), 1, fp);
    printf("%d,%d\n", n, cube);
    fread(&n, sizeof(n), 1, fp);
    fread(&cube, sizeof(cube), 1, fp);
    printf("%d,%d\n", n, cube);
    fread(&n, sizeof(n), 1, fp);
    fread(&cube, sizeof(cube), 1, fp);
    printf("%d,%d\n", n, cube);
    fread(&n, sizeof(n), 1, fp);
    fread(&cube, sizeof(cube), 1, fp);
    printf("%d,%d\n", n, cube);
    fread(&n, sizeof(n), 1, fp);
    fread(&cube, sizeof(cube), 1, fp);
    printf("%d,%d\n", n, cube);
    fread(&n, sizeof(n), 1, fp);
    fread(&cube, sizeof(cube), 1, fp);
    printf("%d,%d\n", n, cube);
    fread(&n, sizeof(n), 1, fp);
    fread(&cube, sizeof(cube), 1, fp);
    printf("%d,%d\n", n, cube);

    fseek(fp,0L, SEEK_END);                    // 调整 fp 指向文件尾
    size = ftell(fp);                          // 获得文件的大小,采用字节数表示
    printf("文件大小为: %d 字节\n", size);     // 72
    fclose(fp);

    return 0;
}
```

解析:

(1) 在参考程序中,rewind 函数用于将文件内部的位置指针重新指向文件的开头;ftell 函数用于得到文件位置指针当前位置相对于文件首的偏移字节数,由于预先通过 fseek 函数令文件位置指针指向了文件尾部,因此 ftell 函数返回的结果等价于文件大小。

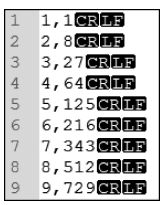

图 3.2 习题 3.12 写入文本文件的内容示意

(2) 文本文件中数值按照 ASCII 码进行写入,等价于字符数,由于 Windows 下'\n'的编码是 CR 和 LF 两个字符(见图 3.2),故每个换行符占 2 字节。从图 3.2 中可知,第 1 行和第 2 行每行 5 个字符,第 3 行和第 4 行每行 6 个字符,其他各行每行 7 个字符。因此文件的总字节数为 57。

(3) 二进制文件中数值直接按编码写入,整数占 4 字节,故 18 个数字共计 72 字节。

3.13 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。

参考程序：

```
#include <stdio.h>
int main(void)
{
    FILE* fp;
    char a,b,c,d,e;
    char t;

    fp = fopen("output.txt", "wb+");          // 按二进制格式打开文件
    scanf("%c%c%c%c%c", &a,&b,&c,&d,&e);  // 从键盘读入字符
    fwrite(&a, sizeof(char), 1, fp);
    fwrite(&b, sizeof(char), 1, fp);
    fwrite(&c, sizeof(char), 1, fp);
    fwrite(&d, sizeof(char), 1, fp);
    fwrite(&e, sizeof(char), 1, fp);
    rewind(fp);
    fread(&t, sizeof(char), 1, fp);           // 读取数据,验证结果
    printf("%c", t);
    fread(&t, sizeof(char), 1, fp);
    printf("%c", t);
    fread(&t, sizeof(char), 1, fp);
    printf("%c", t);
    fread(&t, sizeof(char), 1, fp);
    printf("%c", t);
    fread(&t, sizeof(char), 1, fp);
    printf("%c", t);
    fclose(fp);

    return 0;
}
```

解析：

(1) 5 个字符需要采用“%c”格式控制符依次输入给变量 a,b,c,d,e。

(2) 以上程序存在需要重复执行的读写语句。在学习完后续内容后,可采用循环控制语句实现重复读写。

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

参考程序：

```
#include <stdio.h>
int main(void)
{
    FILE* fp;
    char a;
    char t;
```

```
    fp = fopen("output.txt", "wb+");        // 按二进制格式打开文件
    scanf("%c", &a);
    fwrite(&a, sizeof(char), 1, fp);
    scanf("%c", &a);
    fwrite(&a, sizeof(char), 1, fp);
    scanf("%c", &a);
    fwrite(&a, sizeof(char), 1, fp);
    scanf("%c", &a);
    fwrite(&a, sizeof(char), 1, fp);
    scanf("%c", &a);
    fwrite(&a, sizeof(char), 1, fp);
    scanf("%c", &a);
    fwrite(&a, sizeof(char), 1, fp);
    fseek(fp, 2, SEEK_SET);                 // 调整文件位置标记到第 3 个字符位置
    fread(&t, sizeof(char), 1, fp);
    printf("%c", t);
    fseek(fp, 4, SEEK_SET);                 //调整文件位置标记到第 5 个字符位置
    fread(&t, sizeof(char), 1, fp);
    printf("%c", t);
    fclose(fp);

    return 0;
}
```

解析：

(1) fseek 函数可以设置读取当前位置标记,其一般形式为

```
fseek(文件类型指针,位移量,起始点)
```

其中,起始点有 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；位移量表示以起始点为基点,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。

(2) 第 3 个和第 5 个字符需要文件指针分别相对开始移动 2 个和 4 个字节。

(3) 请尝试使用 SEEK_END 来实现文件指针的定位。

3.2 补充习题

1. 若有定义“int a = 100;”,则语句“printf("%d%d%d\n",sizeof("a"),sizeof(a),sizeof(3.14));”的输出是(　　)。

(A) 328　　(B) 248　　(C) 238　　(D) 421

2. 有如下程序段：

```
int x = 12;
double y = 3.141593;
```

```
printf("%d%8.6f ", x, y);
```

其输出结果是(　　)。

(A) 123.141593　　(B) 12 3.141593

(C) 12,3.141593　　(D) 123.1415930

3. 已知字符'A'的 ASCII 码值是 65,字符变量 c1 的值是'A',c2 的值是'D',则执行语句"printf("%d,%d",c1,c2-2);"的输出结果是(　　)。

(A) 65,68　　(B) A,68　　(C) A,B　　(D) 65,66

4. 设变量均已正确定义,若要通过

```
scanf("%d%c%d%c", &a1, &c1, &a2, &c2);
```

语句为变量 a1 和 a2 赋数值 10 和 20,为变量 c1 和 c2 赋字符 X 和 Y。下面输入形式中正确的是(　　)。(注:□代表空格字符)

(A) 10□X<回车>
　　20□Y<回车>

(B) 10□X20□Y<回车>

(C) 10X<回车>
　　20Y<回车>

(D) 10□X□20□Y<回车>

5. 以下程序段完全正确的是(　　)。

(A) int *p; scanf("%d", &p);

(B) int * p; scanf("%d", p);

(C) int k, *p=&k; scanf("%d", p);

(D) int k, *p; *p=&k; scanf("%d", p);

6. 设文件指针 fp 已定义,执行语句"fp=fopen("file","w");"后,以下针对文本文件 file 操作叙述的选项中正确的是(　　)。

(A) 只能写不能读　　(B) 写操作结束后可以从头开始读

(C) 可以在原有内容后追加写操作　　(D) 可以随意读和写

7. 编写程序,用 * 号输出大写字母 E 的图案,并将该图案写入 data.txt 中。

8. 编写程序,从键盘输入一些字符,逐个把它们送到磁盘上去,直到输入一个#为止。