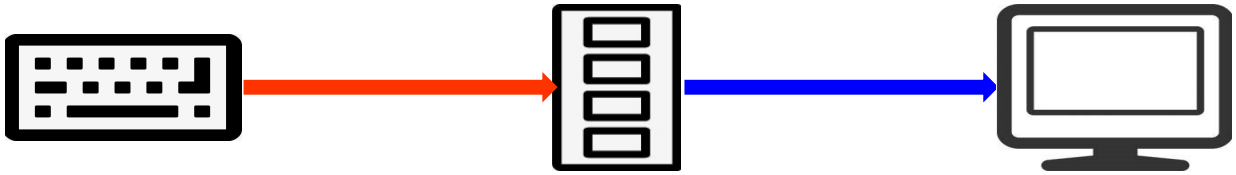
第八章 文件

模块8：C语言的文件操作

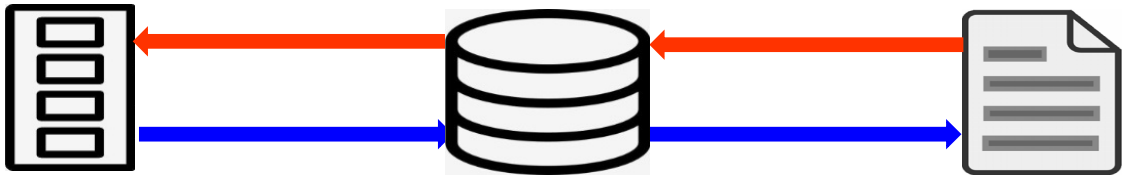
8.1 文件概述

- 标准输入输出



  - 缺点：受限于数据类型，处理数据数量有限，程序结束不保存计算结果。

- 文件输入输出



  - 优点：可以处理大批量的数据，长久的保存计算结果。

8.1 文件概述

- 文件是存储在外存（如磁盘）上的数据集合。
- 每个文件通过唯一的文件名来标识。

  --文件标识，包括三部分：

  - 文件路径：指明文件在外存中的位置
  - 主文件名：要遵循标识符命名规则
  - 文件后缀（扩展名）：表示文件性质

  如：

  D:\document\file1.txt

  表示file1是存放在d盘的document文件夹下的文本文件（txt）
- 计算机按文件名对文件进行读写等操作。

8.1 文件概述

- 文件的物理存储均为二进制，但编码有差异：
  - 文本文件

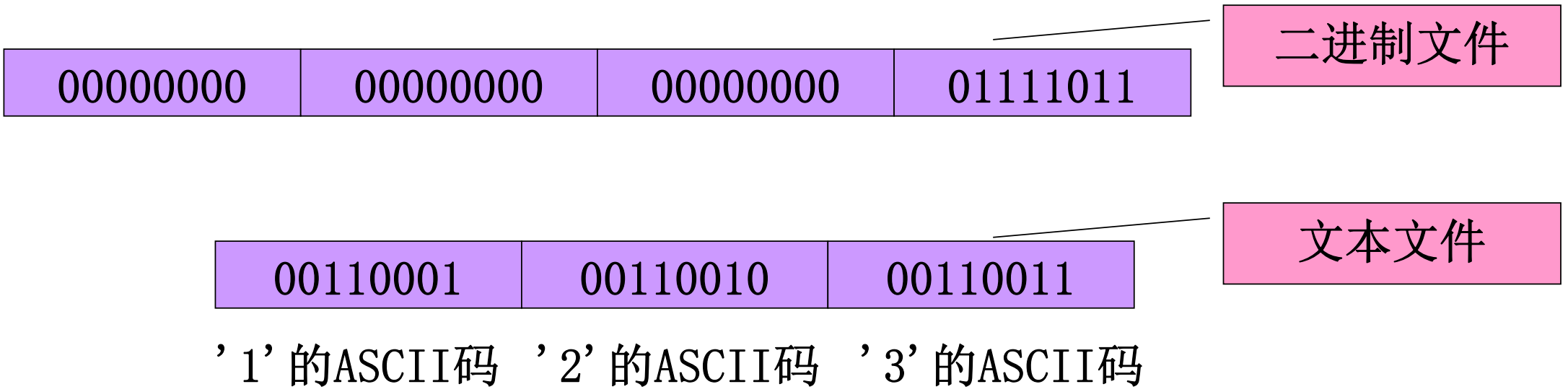
    以ASCII码、Unicode码表示的纯文本文件，只能存储字符信息，不能存储其他信息；文本文件大多是定长编码，每个字符在具体编码中是固定的，如把内存中的数据转换成ASCII码，每个字符用一个ASCII码存储。

  - 二进制文件

    可存储图形文件和文字处理程序等计算机程序；把内存中的数据按其内存中的存储形式不进行格式转换直接存放在文件上，是变长编码。

8.1 文件概述

如：整数123在内存中占4个字节，其二进制文件也占4个字节，文本文件占3个字节（每个数字以其字符的ASCII码存储）



8.1 文件概述

- 文件存取方式

  ➢顺序存取：

  每当“打开”文件进行读或写操作时，总是从文件的开头开始，从头到尾顺序地读写；

  ➢随机存取：

  可以通过调用C语言的库函数去指定开始读写的字节号，然后直接对此位置上的数据进行读写操作。

8.1 文件概述

- I/O的级别

  ➢底层I/O（low-level I/O）：

  使用操作系统提供的基本I/O服务；

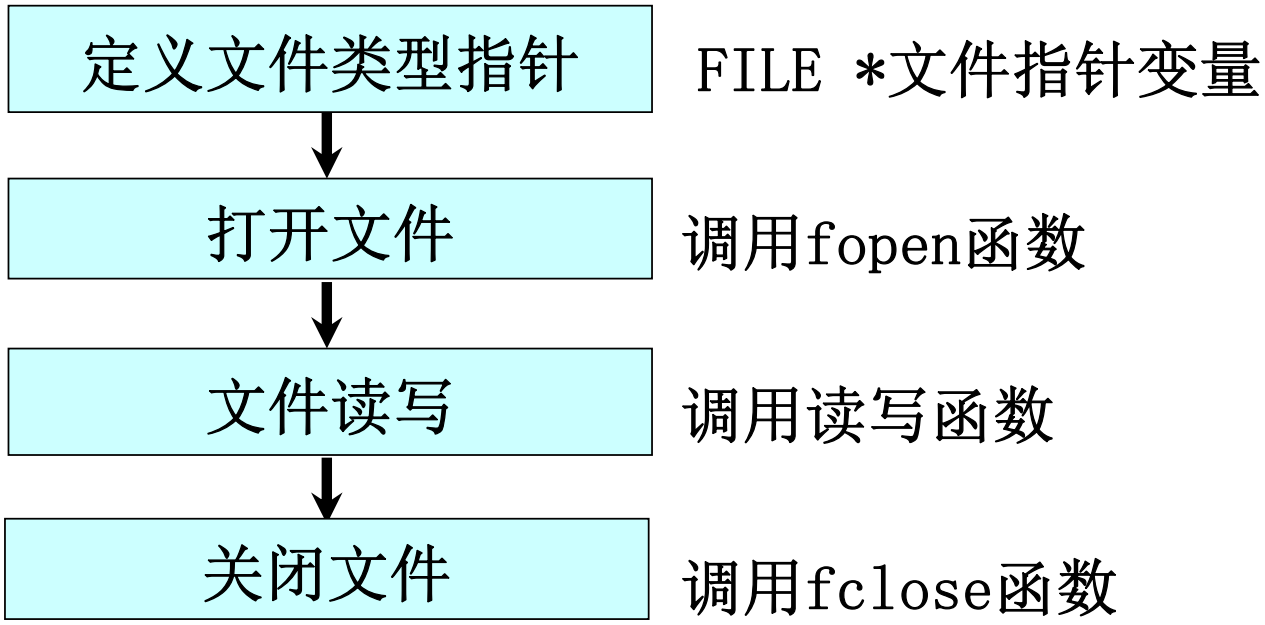
  ➢标准高级I/O（standard high-level I/O）：

  使用C库的标准包和stdio.h头文件定义，可移植性好。

因为无法保证所有的操作系统都使用相同的底层I/O模型，C标准只支持标准I/O包。

8.1 文件概述

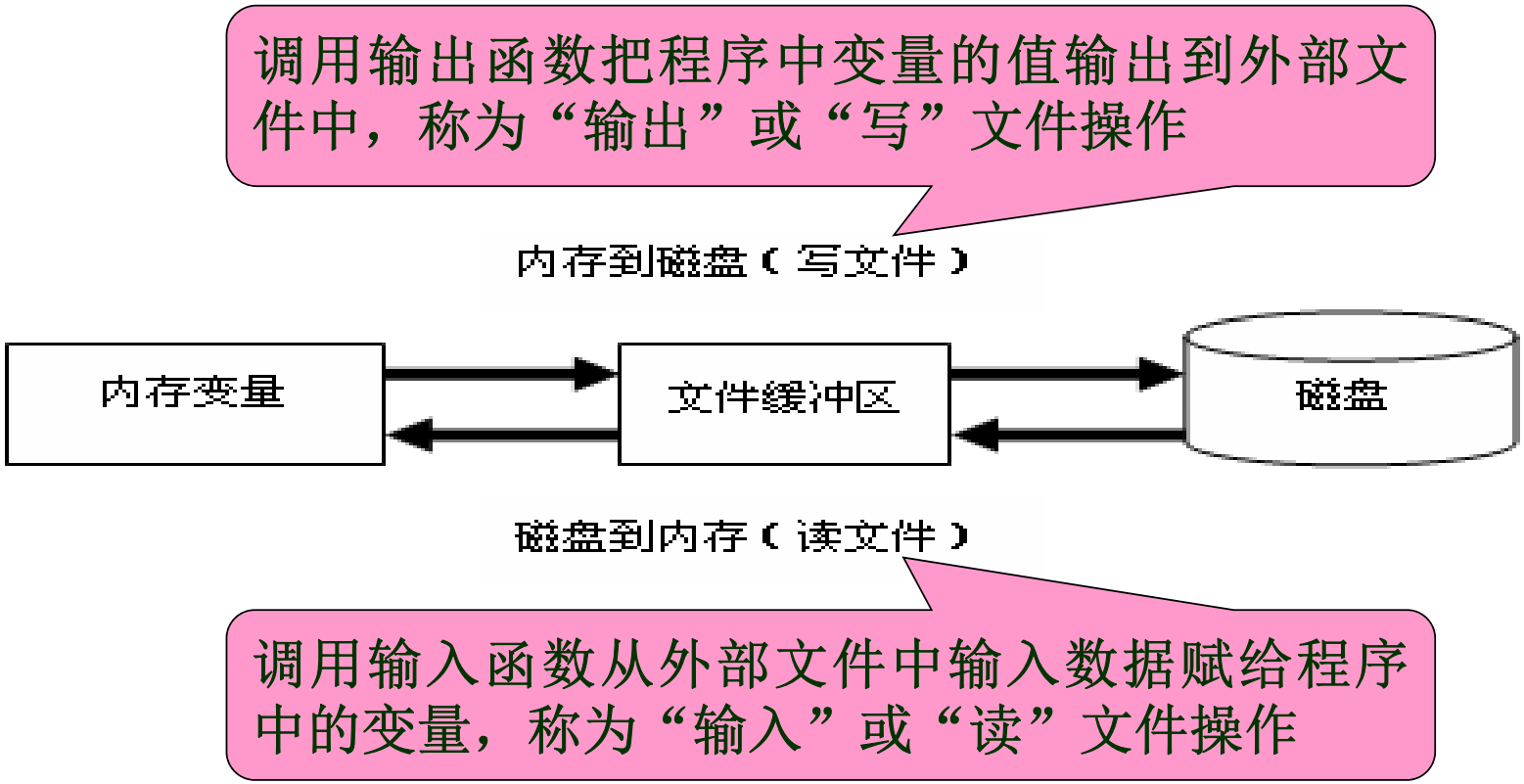
- C语言文件操作步骤



- 标准I/O对文件的处理通过调用标准的输入输出库函数实现。这些专门的函数简化了处理不同I/O的问题。

8.1 文件概述

- 标准I/O的文件操作一般采用“缓冲文件系统”的方式



一次转移一大块信息；缓冲在后台处理，让人有逐字符访问的错觉。

8.1 文件概述

- C语言文件处理的关键是定义一个文件指针，通过该指针对文件进行打开、读写、关闭等操作。

定义：

```
FILE  *文件指针标识符;
```

例如：

```
FILE  *fp1, *fp2;
```

FILE是结构体变量，存放每一个被打开文件的有关信息(如缓冲区的状态、大小，文件当前位置等)。使用FILE类型，需包含stdio.h文件。

8.2 文件的打开与关闭

- 文件的打开

① 函数原型：FILE *fopen(char *fname, char *mode)

② 功能说明：按照mode规定的方式，打开由fname指定的文件。

③ 参数说明：

fname：字符指针，指向要打开或建立的文件名字符串。

mode：字符指针，指向文件处理方式字符串。

④ 返回值：

正常返回：被打开文件的文件指针。

异常返回：NULL，表示打开操作不成功。

8.2 文件的打开与关闭

mode	含义
r	以读模式打开文件，如果文件不存在，则打开文件失败
w	以写模式打开文件，把现有文件的长度截为0，如果文件不存在，则创建一个新文件
a	以写模式打开文件，在现有文件末尾添加内容，如果文件不存在，则创建一个新文件
r+	以更新模式打开文件（即可以读写文件）
w+	以更新模式打开文件（即，读和写），如果文件存在，则将其长度截为0；如果文件不存在，则创建一个新文件
a+	以更新模式打开文件（即，读和写），在现有文件的末尾添加内容，如果文件不存在则创建一个新文件；可以读整个文件，但是只能从末尾添加内容

8.2 文件的打开与关闭

续表：

mode	含义
rb, wb, ab, rb+, r+b, wb+, w+b, ab+, a+b	与上一个模式类似，但是以二进制模式而不是文本模式打开文件
wx, wbx, w+x, wb+x, w+bx	（C11）类似非x模式，但是如果文件已存在或以独占模式打开文件，则打开文件失败

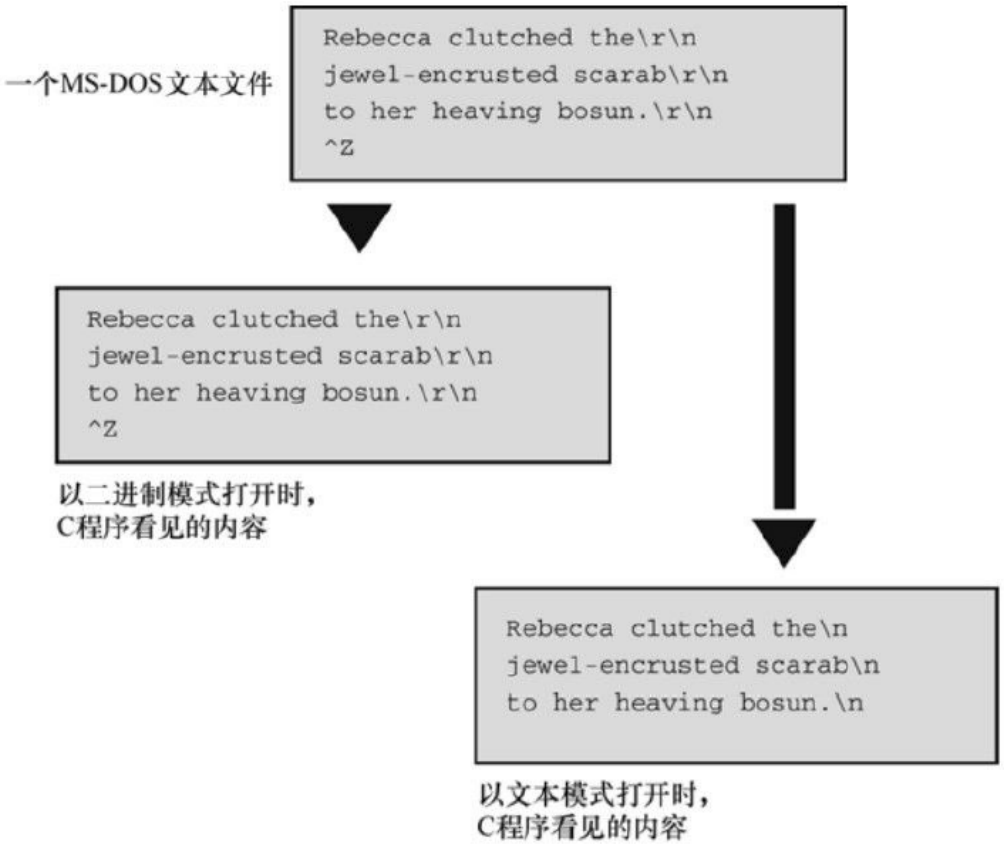
8.2 文件的打开与关闭

- 文件访问途径（文件打开模式）
  - 文本模式

    程序所见的内容与文本实际内容不同。程序会把本地环境表示的行末尾或者文件结尾映射为C模式。

  - 二进制模式

    程序可以访问文件的每个字节，故访问的内容就是文件中存放的内容



```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <stdlib.h>
using namespace std;
int main()
{
    FILE* fp1, * fp2;
    char ch;
    int count = 0;
    fp1 = fopen("..\\file1.txt", "r");
    if (fp1 == NULL)
    {
        cout << "can't open file1.txt";
        exit(EXIT_FAILURE);
    }
    while ((ch = fgetc(fp1))!=EOF)
    {
        putchar(ch);
        count++;
    }
    cout << "文本模式打开文件可读字符数："
         << count << endl;
    count = 0;
    fclose(fp1);
    fp2 = fopen("..\\file1.txt", "rb");
    if (fp2 == NULL)
    {
        cout << "can't open file1.txt";
        exit(EXIT_FAILURE);
    }
    while ((ch = fgetc(fp2)) != EOF)
    {
        putchar(ch);
        count++;
    }
    cout << "二进制模式打开文件可读字符数："
         << count << endl;
    fclose(fp2);
    return 0;
}     //函数细节后面讲，这里观察分析原因即可
```

```
test0343fs
文本模式打开文件可读字符数：11
test0343fs
二进制模式打开文件可读字符数：12
```

8.2 文件的打开与关闭

- 为了程序的通用性，文件名可在程序运行时输入。如：

```
FILE *fp;
char fname[15];
cout << "Input filename:\n";
cin >> fname;
fp = fopen(fname,"r");
```

注意：

在fopen函数中，如果文件名直接给出，则路径中的"\"应写成"\\"；
如果文件名在程序运行时输入，则路径中的分隔符直接输入字符"\"。

```
fp=fopen("D:\\Doc\\ss.txt","a");
```

```
Input filename:
D:\Doc\ss.txt
Press any key to continue
```

8.2 文件的打开与关闭

- 为保证程序正常运行，需对fopen函数的返回值进行检验，以判断文件是否成功地打开。形式如下：

```
if ((fp = fopen(fname, mode)) == NULL)
{
    cout << "can't open file\n";
    exit(EXIT_FAILURE);
}
```

该段程序使得文件打开失败时，显示提示信息，然后调用exit函数结束程序,使用该函数包含文件"stdlib.h"

8.2 文件的打开与关闭

- 文件的关闭

① 函数原型：int fclose(FILE *fp)

② 功能说明：

关闭由fp所指的文件，释放由fp所指的文件类型结构体变量。

③ 参数说明：fp：一个已打开文件的文件指针。

④ 返回值：

正常返回：0。
异常返回：EOF（-1），表示文件在关闭时发生错误。

注意：应该养成及时关闭文件的习惯，防止误操作或其他原因造成丢失数据的情况发生。

8.3 文件的读写

- C语言提供四种顺序读写函数：

  --字符读写、字符串读写、格式读写和数据块读写。

函数	功能	函数	功能
fputc	向文件写入字符	fputs	向文件写入字符串
fgetc	从文件读取字符	fgets	从文件读取字符串
fprintf	向文件格式化写入数据	fwrite	向文件写入数据块
fscanf	从文件格式化读取数据	fread	从文件读取数据块

8.3 文件的读写

➢ 从文件读取字符

① 函数原型：

```
int fgetc(FILE *fp)
```

② 功能说明：

从fp所指文件中读取一个字符，并使文件指针后移一个字符位置。

③ 参数说明：

fp：文件指针，指向要从中读取字符的文件。

④ 返回值：

正常返回：读取的字符。

异常返回：EOF(-1)。

8.3 文件的读写

- 从文件读取字符

```
int  fgetc(FILE  *fp)
```

返回类型的说明：用于区分有效数据和输入的结束(EOF)

- 若返回类型为char：一个字节（无法区分）

  EOF： 0xFF

  数据：0xFF

- 若返回类型为int：四个字节（正确区分）

  EOF： 0xFFFFFFFF

  数据：0x000000XX

8.3 文件的读写

- 文件结束测试函数

① 函数原型：int feof(FILE *fp)

② 功能说明：该函数用来判断文件是否结束。

③ 参数说明：fp：文件指针。

④ 返回值：

0：假值，表示文件未结束。

1：真值，表示文件结束。

⑤ 实例：

```
while(!feof(fp))
{
    ch=fgetc(fp);
    putchar(ch);  //打印到终端
}
```

8.3 文件的读写

- 向文件写入字符

① 函数原型：int fputc(int ch,FILE *fp)

② 功能说明：把ch中的字符写入fp所指的文件当前位置处，并使文件指针后移一个字符位置。

③ 参数说明：

ch：整型变量，内存要写到文件中的字符。

fp：文件指针，指向要在其中写入字符的文件。

④ 返回值：

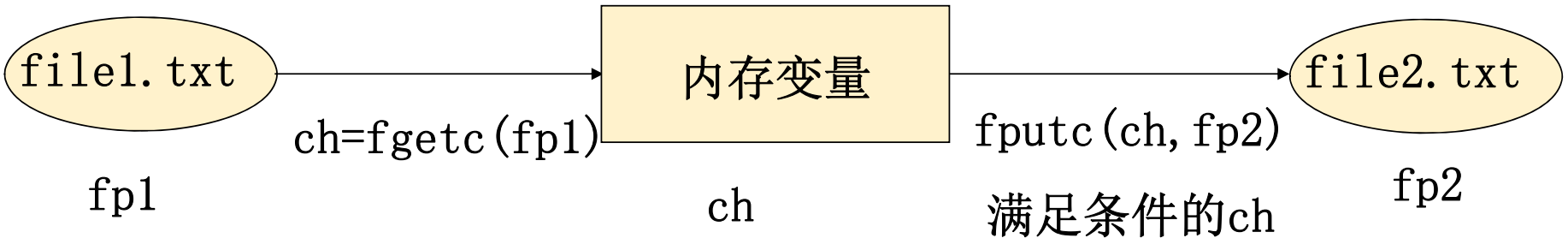
正常返回：写入的字符。

异常返回：EOF(-1)。

8.3 文件的读写

例：编写程序将文件file1.txt的内容显示在屏幕上，同时还将该文件中的数字字符复制到文件file2.txt中。

分析：程序涉及文本文件的读写两种操作，操作方式如下：



```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <stdlib.h>
using namespace std;
int main()
{   FILE *fp1,*fp2;
    char ch;
    fp1=fopen("..\\file1.txt","r");
    if(fp1==NULL)
    {
       cout<<"can't open file1.txt";
       exit(EXIT_FAILURE);
    }
    if((fp2=fopen("..\\file2.txt","w"))==NULL)
    {
       cout<<"can't open file2.txt";
       fclose(fp1);
       exit(EXIT_FAILURE);
    }
    while(!feof(fp1))
    {
       ch=fgetc(fp1);
       putchar(ch);
       if(ch>='0'&&ch<='9')
          fputc(ch,fp2);
    }
    fclose(fp1);
    fclose(fp2);
    return 0;
}
```

..\\ 表示与.sln在同一目录下

```
//例：把文件压缩成原来的1/3!
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <stdlib.h>
int main()
{
  FILE *in, *out;
  int ch;
  int count = 0;
  if ((in=fopen("..\\eddy.txt", "r")) == NULL)
  {
     printf("can't open eddy.txt\n");
     exit(EXIT_FAILURE);
  }
  if ((out=fopen("..\\eddy.red", "w"))== NULL)
  {
     printf("can't open eddy.red\n");
```

该程序以保留每3个字符中的第1个字符的方式压缩文件。压缩后的文本存入新文件，名称是原文件名加上.red后缀（red代表reduced）

```
     fclose(in);
     exit(EXIT_FAILURE);
  }
  while((ch=fgetc(in))!=EOF)
     if (count++ % 3 == 0)
        fputc(ch, out);
  fclose(in);
  fclose(out);
  return 0;
}
```

8.3 文件的读写

➢ 向文件写入字符串

① 函数原型：int fputs(char *str,FILE *fp)

② 功能说明：将str指向的字符串的内容输出到fp所指向文件的当前位置，同时将fp自动向前移动strlen(str)个字符位置。

③ 参数说明：str：可以是字符串常量、字符串指针或字符数组名等。
fp：文件指针，指向字符串要写入其中的文件。

④ 返回值：正常返回：非负整数。
异常返回：EOF（-1）。

⑤ 说明：
- 字符串结束符'\0'不输出到文件
- 不自动在字符串末尾添加换行符

8.3 文件的读写

➢ 从文件读取字符串

① 函数原型：char *fgets(char *str,int n,FILE *fp)

② 功能说明：从由fp指出的文件中读取n-1个字符，并把它们存放到由str指出的字符数组中去，最后自动加上一个字符串结束符'\0'。

③ 参数说明：str：接收字符串的内存地址，可以是数组名或指针。

n： 指出要读取字符的个数。

fp：文件指针，指向要从中读取字符的文件。

④ 返回值：正常返回：字符串的内存首地址，即str的值。

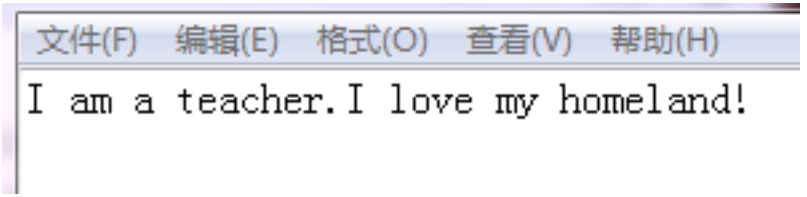
异常返回：NULL。

⑤ 说明：
- 读入n-1个字符到文件，遇到换行符或文件结束符则提前结束
- 读入结束后，系统将自动在最后加'\0'

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <stdlib.h>
using namespace std;
int main()
{   FILE *fp;
    char s[100];
    if((fp=fopen("file1.txt", "w"))==NULL)
    {     printf("can't open file1.txt\n");
          return -1;
    }
    fputs("I am a teacher.",fp);
    fputs("I love my homeland!", fp);
    fclose(fp); //必须先关闭，再以读的方式打开。
    if((fp=fopen("file1.txt", "r"))==NULL)
    {     printf("can't open file1.txt\n");
          return -1;
    }
```

表示与.cpp在同一目录下

```
    while(!feof(fp))
    {     fgets(s,16,fp);
          puts(s);
    }
    fclose(fp);
    return 0;
}
```




```
I am a teacher.
I love my homel
and!
```

8.3 文件的读写

- 向文件格式化写入数据

① 函数原型：int fprintf(FILE *fp,const char *format,arg_list)

② 功能说明：将变量表列（arg_list）中的数据，按照format指出的格式，写入由fp指定的文件。

③ 参数说明：fp：文件指针，指向将数据写入的文件。
format：指向写出数据格式字符串的指针。
arg_list：要写入文件的变量表列，各变量之间用逗号分隔。

④ 返回值：正常返回：输出的字节数。
异常返回：负值。

⑤ 示例：fprintf(fp,"%d%s",4,"China");
表示将整数4和字符串"China"写入fp所指的文件中。

8.3 文件的读写

- 从文件格式化读取数据

① 函数原型：int fscanf(FILE *fp,const char *format,add_list)

② 功能说明：按照format指出的格式，从fp指定的文件中读取数据存放至地址表列（add_list）的变量中。

③ 参数说明：fp：文件指针，指向要从中读取数据的文件。
format：指向读取数据格式字符串的指针。
add_list：存放读取数据的变量的地址表列。

④ 返回值：正常返回：成功读取的参数的个数。类似scanf，如果%d读到了"abc"，返回0。异常返回：EOF（-1）。

⑤ 示例：fscanf(fp,"%d%d",&x,&y);
表示从fp所指的文件中顺序读取两个整数给变量x和y。

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <stdlib.h>
using namespace std;
#define MAX 41
int main()
{
  FILE* fp;
  char words[MAX];
  if ((fp = fopen("..\\wordy", "a+")) == NULL)
  {
    cout << "Can't open wordy file.\n";
    exit(EXIT_FAILURE);
  }
  cout << "Enter words to add to the file;
press the #\nkey at the beginning of a line
to terminate.\n";
```

运行多次观察结果

```
  while((fscanf(stdin,"%40s",
        words) == 1) &&
        (words[0] != '#'))
    fprintf(fp,"%s\n",words);
  cout << "File contents:\n";
  rewind(fp); //返回文件开始
  while (fscanf(fp, "%s",
         words) == 1)
      cout << words << endl;
  cout << "Done!\n";
  fclose(fp);
  return 0;
}
```

8.3 文件的读写

- 上述标准I/O函数都是面向文本的，用于处理字符和字符串。

  思考：如何要在文件中保存数值数据？

  例：double num = 1./3.;

  fprintf(fp,"%f", num); //储存为8个字符：0.333333

- 一般而言，用 fprintf()函数和%f转换说明只是把数值保存为字符串。fprintf()把数值转换为字符数据，这种转换可能会改变值。

- 以二进制形式储存数据，不存在从数值形式到字符串的转换过程，保证数值在储存前后一致。

- fread和fwrite函数用于以二进制形式处理数据。

8.3 文件的读写

- 从文件读取数据块

① 函数原型：

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

② 功能说明：从文件指针fp所指的文件的当前位置读取字节数为size大小的数据块共count个，存到buffer所指的内存储区中。

③ 参数说明：

buffer：指向存放读入数据存储区的首地址的指针。

size：数据块的字节数，即一个数据块的大小。

count：一次读入数据块的数量。

fp：文件指针，指向要从其中读出数据的文件。

④ 返回值：正常返回：<=count。正常情况下，该返回值就是count，但如果出现读取错误或读到文件结尾，该返回值就会比count小；异常返回：0。

8.3 文件的读写

➢ 向文件写入数据块

① 函数原型：

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

② 功能说明：将以buffer为起始地址的长度为size的count个数据块输出到文件指针fp所指的位置去。

③ 参数说明：

buffer：指向存放输出数据存储区的首地址的指针。

size：数据块的字节数，即一个数据块的大小。

count：一次输出数据块的数量。

fp：文件指针，指向要从其中写入数据的文件。

④ 返回值：

正常返回：实际输出数据块的个数，即count。

异常返回：0。

8.3 文件的读写

- fread和fwrite常用于读写二进制文件

例如，假设fp指向以二进制形式打开的可读写文件，并有如下的说明：

```
float f;  double d[10];
```

常见的块读写应用示例：

```
fwrite(&f,sizeof(float),1,fp);      //把浮点数f写入文件
fwrite(d,sizeof(double),10,fp);     //把数组d中所有数写入文件
fread(&f,sizeof(float),1,fp); //从文件中以块形式读一浮点数到变量f中
fread(d,sizeof(d),1,fp); //从文件中一次性读一个d大小的数据块到数组d中
fwrite(&d[0],sizeof(float),1,fp);
                          //参数类型不匹配会导致数据错误（编译不错）
```

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <stdlib.h>
using namespace std;
int main()
{
    FILE* fp; char ch; int count = 0;
    fp = fopen("..\\onlytest.txt", "w");
    if (fp == NULL)
    {
        cout << "can't open onlytest";
        exit(EXIT_FAILURE);
    }
    char d[] = "hello\0chen";
    fwrite(d, sizeof(d), 1, fp); //fprintf(fp,"%s",d);
    return 0;
}
```

分别观察运行后的文件结果：

例：从键盘上读入5个学生的成绩信息，将它们以文件的形式保存在磁盘上。然后再将从文件中读取学生信息，并在屏幕上显示出来。假设学生成绩信息包括学生姓名、学号、总分。

具体步骤如下：

- 声明学生成绩结构类型
- 定义学生成绩结构数组
- 以块读写的方式对学生信息进行文件读写

```
#define _CRT_SECURE_NO_WARNINGS
#define N 5
#include <cstdio>
using namespace std;
struct student
{
  char name[20];
  char num[8];
  double score;
};
int main()
{
  struct student s[N], t[N];
  int i;
  FILE* fp;
  if ((fp=fopen("student.dat", "wb")) == NULL)
  {  printf("can't open student.dat\n");
     return -1;  }
```

```
    for (i = 0; i < N; i++)
    {
        scanf("%s%s%lf", s[i].name, s[i].num, &s[i].score);
        fwrite(&s[i], sizeof(student), 1, fp);//逐元素向文件写数据
    }
    fclose(fp); //写文件结束先关闭，再以读的方式打开。
    if ((fp = fopen("student.dat", "rb")) == NULL)
    {
        printf("can't open student.dat\n");
        return -1;
    }
    fread(t, sizeof(t), 1, fp); //一次从文件读出整个数组
    for (i = 0; i < N; i++)
        printf("%s %s %lf ", t[i].name, t[i].num, t[i].score);
    fclose(fp);
    return 0;
}
```

//本示例受限篇幅，没有输入输出提示及输入数据合理性检验等

8.3 文件的读写

- 随机读写

  ➢顺序读写文件只能从头开始，顺序读写各个数据。

  ➢随机读写可按需要只读写文件中某些指定的部分。

  ➢随机读写的关键是要按要求移动位置指针，即进行文件的定位。

  ➢实现文件定位、移动文件内部位置指针的函数主要有rewind、ftell、fseek、fgetpos和fsetpos()等函数。

8.3 文件的读写

- 随机读写

  1.rewind 函数

  ① 函数原型：void rewind(FILE *fp)

  ② 功能说明：使指示文件位置的指针重新返回到文件开始。

  ③ 参数说明：fp是文件指针

8.3 文件的读写

- 随机读写

  2.ftell 函数

  ① 函数原型：long ftell(FILE *fp)

  ② 功能说明：求文件指针的当前位置。

  ③ 参数说明：fp是文件指针。

  ④ 返回值：距文件开始处的字节数。

8.3 文件的读写

- 随机读写

  3. fseek 函数

  ① 函数原型：int fseek(FILE *fp, long offset, int whence)

  ② 功能说明：使文件指针fp移到基于whence的相对位置offset处。

  ③ 参数说明：offset是相对whence的字节位移量，用长整型表示。

  whence是移动的基准，常用符号常量表示。

  ④ 返回值：正常返回：0。

  异常返回：-1，表示定位操作出错。

8.3 文件的读写

- 参数whence的意义

符号常量	值	基准位置
SEEK_SET	0	文件开头
SEEK_CUR	1	当前读写的位置
SEEK_END	2	文件尾部

- 示例

```
fseek(fp, 0L, SEEK_SET);     // 定位至文件开始处
fseek(fp, 10L, SEEK_SET);    // 定位至文件中的第10个字节
fseek(fp, 2L, SEEK_CUR);     // 从文件当前位置前移2个字节
fseek(fp, 0L, SEEK_END);     // 定位至文件结尾
fseek(fp, -10L, SEEK_END);   // 从文件结尾处回退10个字节
```

8.3 文件的读写

- 随机读写
  - fseek()和ftell()潜在的问题是，它们都把文件大小限制在long 类型能表示的范围内。
  - ANSI C新增了两个处理较大文件的新定位函数：fgetpos()和fsetpos()。
  - 这两个函数不使用long类型的值表示位置，它们使用一种新类型：fpos_t（代表file position type，文件定位类型）。

8.3 文件的读写

- 随机读写

  4. fgetpos 函数

  ① 函数原型：int fgetpos(FILE * fp, fpos_t * pos)

  ② 功能说明：调用该函数时，它把fpos_t类型的值放在pos指向的位置上，该值描述了文件中的一个位置。

  ③ 参数说明：fp是文件指针；pos是fpos_t类型的指针。

  ④ 返回值：如果成功，函数返回0；

  如果失败，返回非0。

8.3 文件的读写

- 随机读写

  5. fsetpos 函数

  ① 函数原型：int fsetpos(FILE *fp, const fpos_t *pos)

  ② 功能说明：调用该函数时，使用pos指向位置上的fpos_t类型值来设置文件指针指向该值指定的位置。fpos_t类型的值应通过之前调用fgetpos()获得。

  ③ 参数说明：fp是文件指针；pos是fpos_t类型的指针。

  ④ 返回值：如果成功，函数返回0；如果失败，则返回非0。

8.4 其他标准I/O函数

- ungetc 函数

① 函数原型：int ungetc(int c, FILE *fp)

② 功能说明：把c指定的字符放回输入流中，下次调用标准输入函数时将读取该字符。

③ 参数说明：c是要被放回的字符，以其对应的int值进行传递；

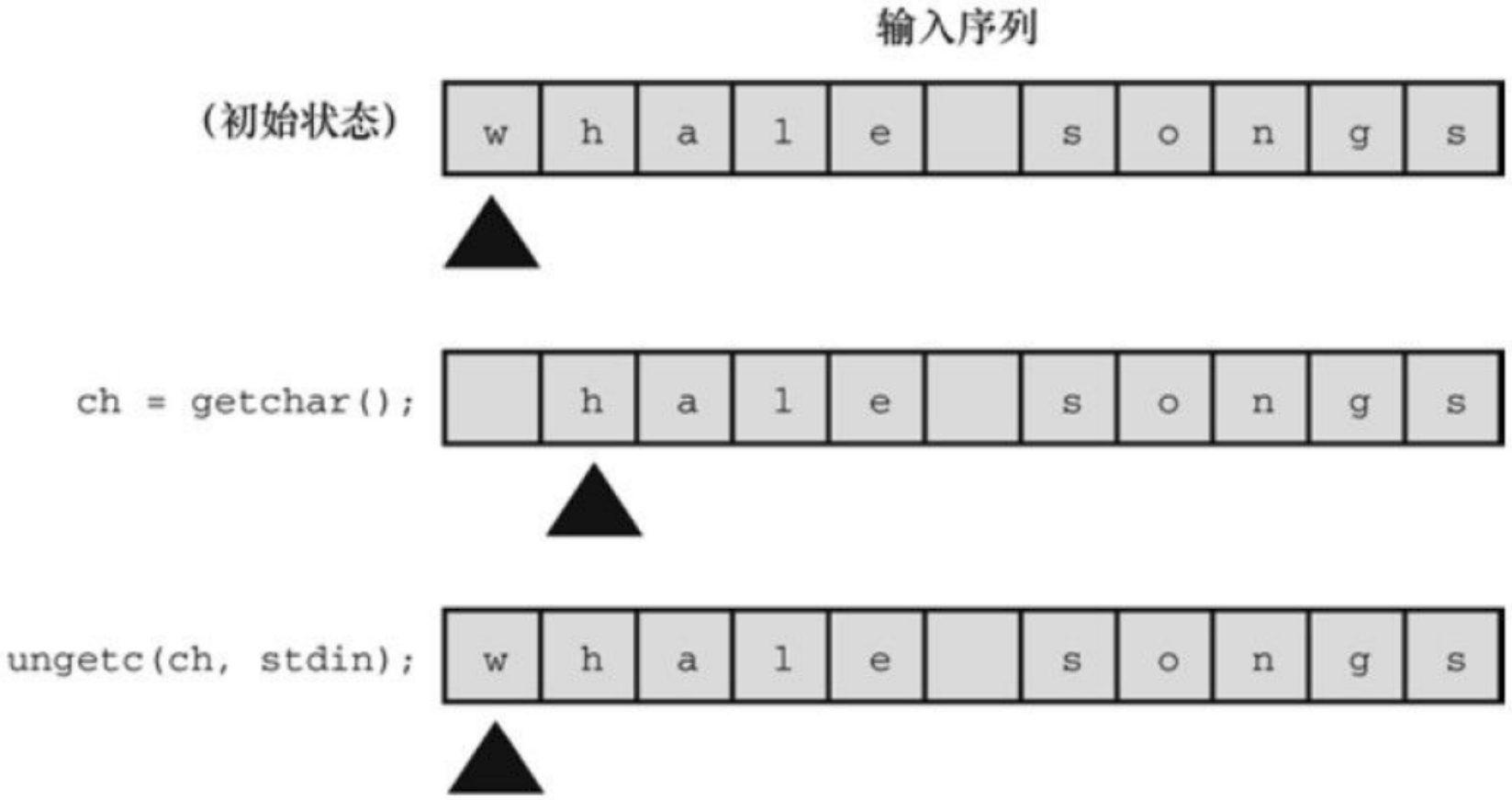
fp是文件指针。

① 返回值：如果成功，则返回被放回的字符；

否则返回EOF。

8.4 其他标准I/O函数

- ungetc 函数



8.4 其他标准I/O函数

- fflush 函数

① 函数原型：int fflush(FILE *fp)

② 功能说明：会强迫将缓冲区内的数据写回参数fp指定的文件中；如果参数fp为NULL，fflush()会将所有打开的文件数据更新。

③ 参数说明：fp是文件指针。

④ 返回值：如果成功刷新，则返回0。指定的流没有缓冲区或者只读打开时也返回0值；否则返回EOF。

8.4 其他标准I/O函数

- setvbuf 函数

① 函数原型：

```
int setvbuf(FILE *fp, char *buf, int mode, size_t size)
```

② 功能说明：设置文件的缓冲区。

③ 参数说明：指针fp识别待处理的流；buf指向待使用的存储区，如果buf的值不是NULL，则必须创建一个缓冲区；变量size告诉setvbuf()数组的大小；

8.4 其他标准I/O函数

- setvbuf 函数（续）

mode的选择如下：

_IOFBF表示完全缓冲（在缓冲区满时刷新）；

_IOLBF表示行缓冲（在缓冲区满时或写入一个换行符时）；

_IONBF表示无缓冲（没有缓冲区，直接从流中读入/写入数据）。

④ 返回值：如果操作成功，函数返回0；

否则返回非零值。

8.4 其他标准I/O函数

- ferror 函数

① 函数原型：int ferror(FILE *fp)

② 功能说明：判断文件是否出错。

③ 参数说明：fp是文件指针。

④ 返回值：如果读写未出错，则返回0；否则返回一个非零值。

注意：

- 同一文件每次调用输入输出函数，均产生一个新的ferror函数值。
- 在执行fopen函数时，ferror函数的初始值自动置为0。

8.5 本节小结

- C语言文件操作相关函数：fopen()、fclose()、exit()、feof()、fgetc()、fputc()、fgets()、fputs()、fprintf()、fscanf()、fread()、fwrite()、rewind()、fseek()、ftell()、fgetpos()、fsetpos()、ungetc()、fflush()、setvbuf()、ferror()
- 如何使用C标准I/O系列的函数处理文件
- 文件模式和二进制模式、文本文件和二进制文件
- 文件的顺序访问与随机访问