IO编程

课堂目标

1. 了解标准IO
2. 掌握文件流指针
3. 掌握FILE结构体和流 的概念
4. 掌握缓存机制
5. 熟练使用常见标准IO函数

课堂实验

1. 编写一段程序来测试文件的行数
2. 编写一段程序实现简单的日志文件
3. 编写一段程序，通过标准IO实现文本文件中蜗牛学院人员信息统计和查询

课程引入

> 我们之前学习了文件IO，会遇到一些问题：
>
> - 如果读写中有中文，那么可能会乱码
> - 不能很方便的读写每一行内容
> - 不能很方便的读写字符串：比如额外使用strtok

一、标准IO（input/output） 介绍

1、概念

标准I/O:是指标准I/O库，又称带缓存的I/O，属于ISO实现的**输入输出的标准库函数**，内部调用的是底层的系统调用，是在文件IO的基础上封装出来的函数接口，也就是说基于低级IO的一个封装和抽象。它由ANSI C建立的一个标准I/O模型，是一个标准函数包和 `stdio.h` 头文件中的定义，具有一定的可移植性。标准I/O库代替用户处理很多细节，比如缓存分配、以优化长度执行I/O等，提供缓存的目的是为了尽量减少read和write的调用次数。我们可以使用标准IO 提供的各种c库函数用来进行各种输入和输出 功能实现，并且相比于低级IO ，在使用上会更加方便。

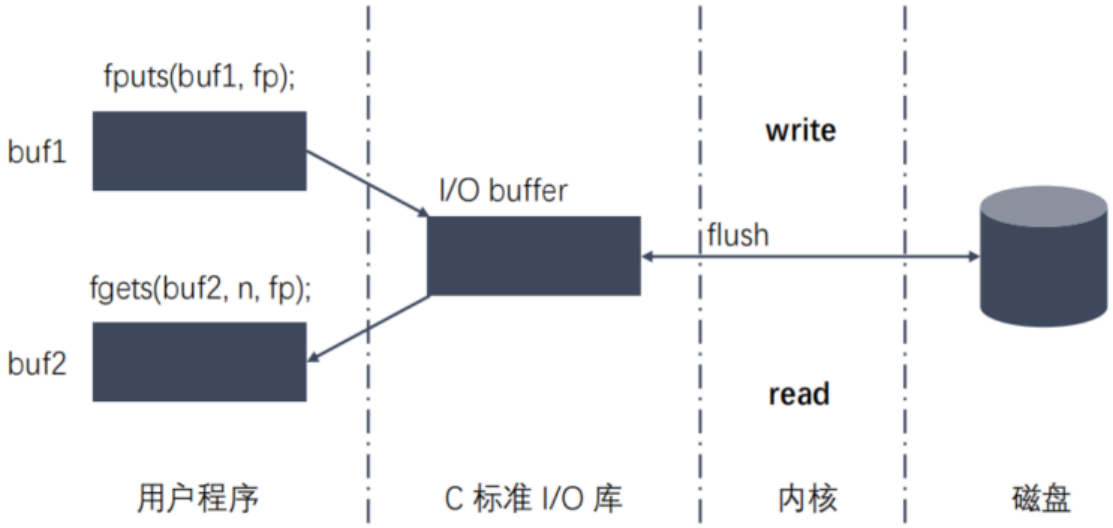
2、标准IO主要函数分类

分类	描述
文件读写	用于文件的打开、关闭、读写或设置读取文件位置等
标准输入	终端进行数据的读取(stdin)，比如scanf
标准输出	将数据输出到stdout（终端）中，printf、puts
标准错误	将错误信息输出到stderr中 perror（）：stderr是内置的一个文件流指针

我们之前使用的printf以及scanf等函数就是标准IO里的函数，我们后续会继续学习fopen和fgets等函数来优化对文件的操作。这些函数都是在c语言标准库里面: `<stdio.h>`

3、标准IO的工作流程

标准IO 相比于低级IO 最大的区别在于自带缓存区并利用其来对io操作进行缓存优化，从而达到资源优化、效率提升的目的：



将逻辑单元中的数据写入文件，根据需求，有三种函数类型可以调用。以fputc、fputs、fwrite为例，这些函数不用人为去控制缓冲区的大小，而是系统自动申请的。当用户使用了相应的I/O函数之后，根据不同的缓存类型（是全缓冲、行缓冲还是无缓冲），**系统自动用malloc等函数申请缓冲区**，即标准I/O缓存。当用户缓冲区满了之后，如系统I/O操作一般，此时调用write从标准I/O缓存中复制数据到内核缓冲区，再写入磁盘。同系统I/O操作，从内核缓冲区调用read读入到用户缓冲区。从逻辑单元中读取文件，同样有三种函数类型可以调用。图中以**fgetc、fgets、fread**为例，读入逻辑单元进行后续的处理。其中fputs可以写入字符串，fgets可以从文件中获取内容。

总结:标准IO是基于低级IO进行设计的带缓存的IO机制。提供了一系列的库函数用于完成**输入和输出**相关操作。

4、标准IO 缓存机制

（0）总结

- 标准错误：无缓存.stderr. `fputs("hello error",stderr)`
- 终端的写入和输出：行缓存。如果要及时查看输出，尽量输出是带一个 `\n`

- 文件的读写：全缓存。为了保证写入成功。一般操作结束后加fclose，如果要立即看效果，就可以用fflush();

（1）概念

C语言的标准IO提供了缓冲机制，这是为了提高文件IO操作的效率。缓冲区是内存中的一块区域，用于临时存储输入或输出数据.所谓的缓存机制是指在进行IO操作时，不会理解去改变文件里的数据或直接近输出，而是会保存到一个专门的内存区域（缓存区）中，然后满足一定条件时才会将缓冲区里的数据进行真实的输出或写入操作。

（2）缓存区

缓冲区是一块专门的内存区域，用于实现程序和设备之间的数据交互的缓存，从而减少过多的设备写入和读取操作，增加性能以及减少资源开销。

缓冲区一共有以下几种模式进行操作

1. **全缓冲（Fully Buffered）**
   当流是全缓冲的，数据会被积累在缓冲区中，直到缓冲区满了或者你手动刷新缓冲区（比如使用 fflush 函数），才会进行实际的IO操作。这种缓冲方式**常用于对文件**的操作，因为它可以减少对磁盘的读写次数。
2. **行缓冲（Line Buffered）**
   行缓冲意味着当输入或输出中遇到换行符时，缓冲区的内容才会被写出。标准输出stdout（向终端输出）和标准输入(stdin)通常是行缓冲的，这样可以在写出每行之后立即看到结果。
3. **无缓冲（Unbuffered）**
   如果一个流是无缓冲的，那么每个输入或输出操作都是直接执行的。标准错误输出stderr通常是无缓冲的，以保证错误信息可以立即输出。

默认情况下所有流在第一次IO操作时都会被赋予一个缓冲区，除非流被设置为无缓冲。**对于标准输入和标准输出，通常情况下它们会是行缓冲的**，但如果它们并不关联到交互式设备，例如重定向到文件时，它们可能会变成全缓冲的。即针对于普通的文件操作，一般是全缓存。如果是标准输出或

标准输入（比如终端、控制台之类的）是行缓存

缓存区触发时间：

1. **缓冲区满了自动触发（文件）**
2. **程序结束或调用文件关闭函数：自动触发**
    1. fclose close exit(0) main**函数的**return： **程序自然结束**
3. **修改了缓冲区，自动触发一次**
    1. setbuf
4. **手动调用刷新函数：**fflush
5. 其他方式

（3）修改缓存机制类型

可以通过调用 `setvbuf` 库函数可以用来设置流的缓冲类型：

```
#include <stdio.h>

int setvbuf(FILE *stream, char *buffer, int mode, size_t
size);
```

- **stream**：这是指向 `FILE` 对象的指针，该 `FILE` 对象标识了要修改缓冲策略的流。
- **buffer**：这是指向一个预分配缓冲区的指针，用于流的缓冲。如果设置为 `NULL`，C库会自动为你分配一个缓冲区。
- **mode**：这是一个整数，用来指定缓冲的类型。可以是以下之一
    - `_IOFBF`：全缓冲，在缓冲区满或显式调用 `fflush()` 或文件被关闭时才进行物理IO操作。
    - `_IOLBF`：行缓冲，每当有一个换行符被写入缓冲，或者缓冲满了的时候，缓冲区的内容会被刷新。
    - `_IONBF`：无缓冲，输出操作会直接写入到文件中，无任何缓冲。

- size：这是缓冲区大小，以字节为单位。
- 返回值：只要是非负数，就是成功的。

示例代码：

```
FILE *fp = fopen("myfile.txt", "w");

if (fp == NULL) {
    perror("Error opening file");
    return -1;
}

char buffer[BUFSIZ]; // BUFSIZ is a macro constant that
stands for buffer size, defined in <stdio.h>
if (setvbuf(fp, buffer, _IOFBF, BUFSIZ) != 0) {
    perror("setvbuf failed");
} else {
    // The stream now uses full buffering with the
specified buffer and size
}
```

5、文件流指针和结构体

（1）文件流指针

文件流指针是指指向某个文件结构体（FILE：c语言内置结构体）的指针。在C语言中，文件流是一个用于文件输入/输出的抽象数据类型。文件流指针通常使用**FILE类型的指针**来表示，通过使用文件流指针，我们可以对文件进行读写操作。

总结：文件流指针是一个文件在c语言程序里的表现形式。每一个文件都有自己的文件流指针。开发者就可以利用文件流指针以及对应的读取和写入函数来完成文件数据的读取和写入。

例如，可以使用fopen()函数打开一个文件并返回一个指向该文件的FILE类型的指针。然后，可以使用该指针以及函数来读取或写入文件里的数据，比如使用fscanf()函数从文件中读取数据或将使用fprintf将数据写入文件中。最后，可以使用**fclose()**函数关闭文件流并释放资源。

文件流指针指向一个包含文件名、文件状态和文件的当前信息等数据结构的结构体的地址。这个结构体通常包含一个缓冲区，用于存储从文件中读取的数据或写入文件的数据。文件流指针还包含一些控制标志位，例如读写状态标志位、错误标志位等。

在使用文件流指针时，我们可以打开一个文件并创建一个指向该文件的文件流指针。然后，可以使用文件流指针来读取或写入文件。当完成文件操作后，需要关闭文件流以释放资源。

我们通过文件流指针对应结构体来指向某一个文件，作用类似于低级IO里的文件描述符

（2）File类型

FILE结构体：该结构体中包含以下内容：

1. **文件描述符**：操作系统提供的底层文件标识符（低级io使用的fd）。
2. **缓冲区指针**：指向流的缓冲区，用于临时存储输入或输出的数据。
3. **错误标识**：标记是否在流的操作中发生了错误。
4. **文件位置指示器**：表示文件内当前操作的位置。，俗称文件位置光标
5. **文件状态标志**：存储文件的状态，例如读写模式。
6. **缓冲区大小**：指示缓冲区的大小。
7. **结束-of-文件(EOF)标识**：标记是否到达了文件末尾。一般是-1（end of file）

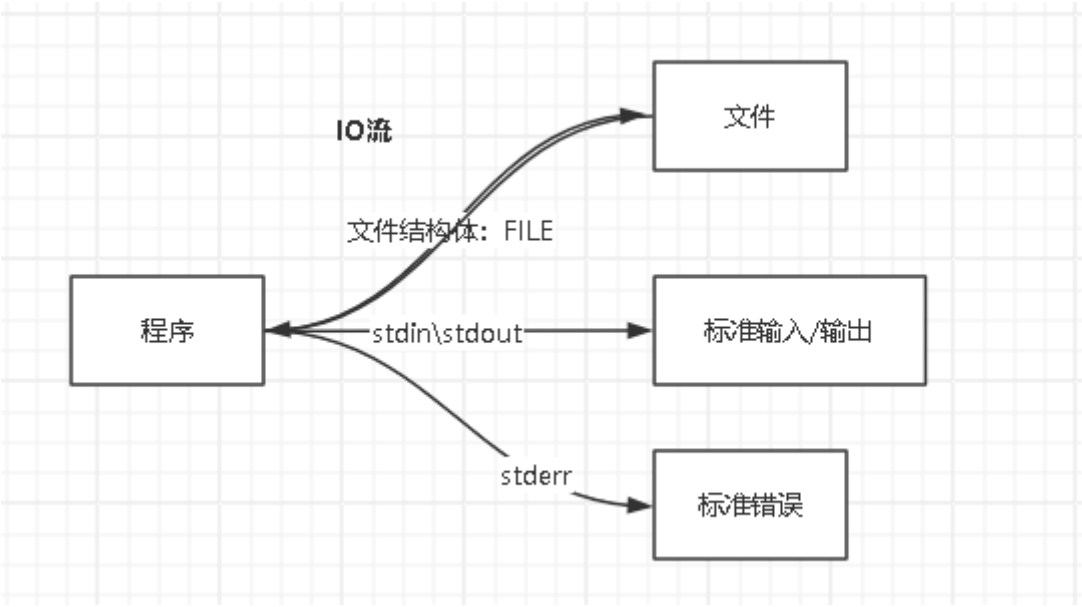
FILE结构体一般是在使用fopen函数时会生成并指向一个FILE类型的结构体。当你调用 `fopen()` 函数时，会返回一个指向 `FILE` 结构体的指针。这个结构体内部维护了文件的操作状态，包括当前读写的位置、是否遇到错误等。在使用例如 `fread` 、 `fwrite` 、 `fprintf` 、 `fgetc` 等函数进行文件操作时，这些函数会通过 `FILE` 指针访问这些状态信息来执行相应的操作。

注意，由于 `FILE` 结构是内部实现的一部分，你作为程序员通常不需要直接操作这个结构体，而是通过标准I/O函数利用 `FILE` 指针来进行文件操作

（3）IO流

IO流（Input/Output Stream）是一个表示输入源或输出目的地的抽象概念。可以把IO流想象成水流，数据就像是流水中的水一样，从一个地方流向另一个地方。

- **输入流**：这就像是水管中的水源，水（数据）从水源（输入设备或文件）流入程序中。输入流用于从某个来源（可以是文件、终端或其他输入设备）读取数据。
- **输出流**：这像是水流的出口，水（数据）从程序流到目的地（输出设备或文件）。输出流用于将数据写出到某个目的地（文件、终端或其他输出设备等）。

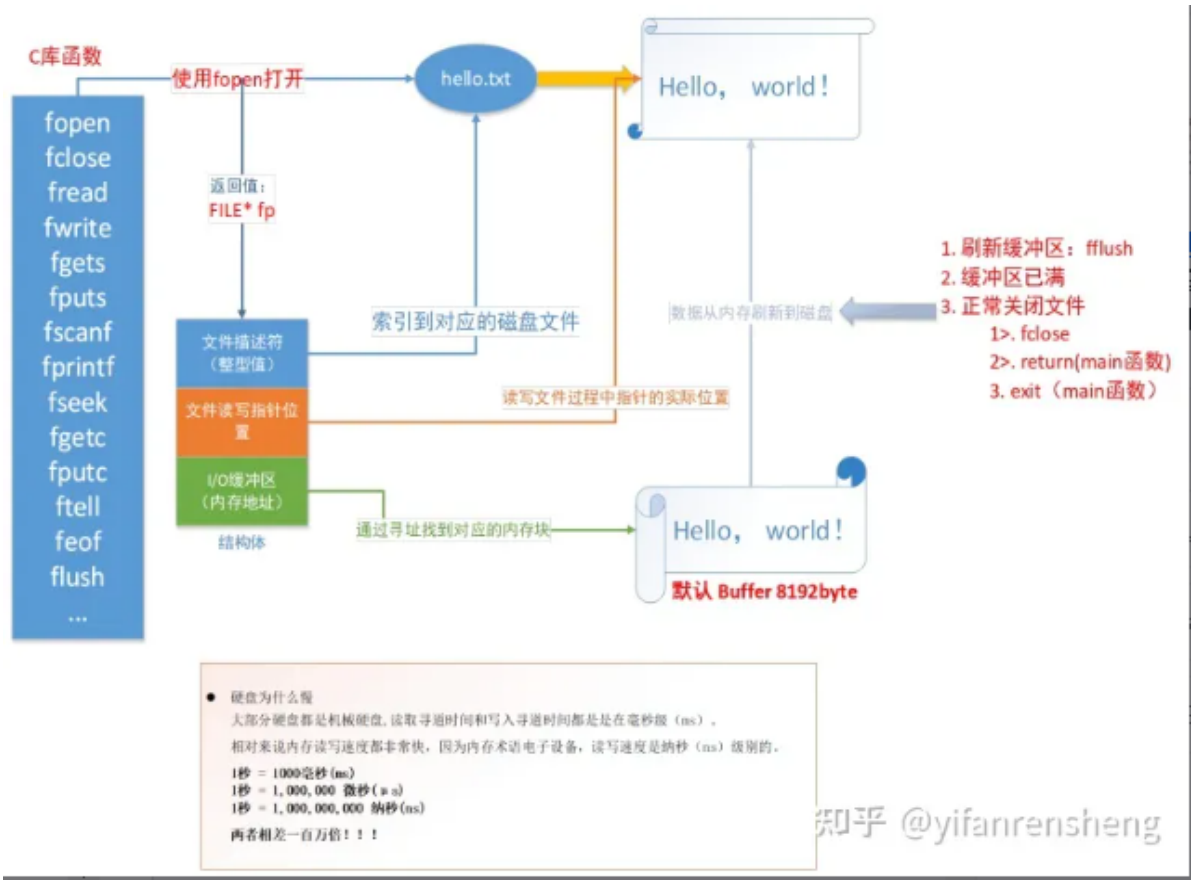


总结：IO流包含输入流和输出流，本身是一个抽象概念只是用于方便描述数据的流向，输入流指的是将设备或文件里的数据流向程序。输出流就相反。io流是抽象概念

在程序内部我们是通过文件流指针（指向的是FILE结构体：包含文件的相关信息）表示要操作的某个文件，作用类似于文件描述符。可以通过调用标准io函数以及给函数传入文件流指针来完成输入流或输出流的功能。

学习标准IO函数：利用文件流指针以及标准io函数实现**数据的输入和输出**（程序->文件和设备（终端），输出流：文件和设备->程序）

在编程语境中，使用IO流可以无需关心数据的来源和去向，提供了一种统一的方法来读取或写入数据，使得程序代码可以通用于不同的数据源和目的地。例如，无论是读取来自硬盘上的文件还是网络上的消息，都可以使用相同的代码结构，这是因为都被抽象成了流的形式。这样的抽象屏蔽了背后不同硬件和协议的复杂性，让程序员可以通过统一的接口与数据交互。也就是说不同的编程语言针对于IO流都有不同的实现，但对于输入和输出都有自己的实现。比如c语言中我们就可以通过文件指针以及fopen等io函数来使用io流。比如fopen函数就是打开一个文件并创建一个该文件对应的流，该函数返回的FILE指针其实就可以理解为该文件的一个io流，通过这个io流就可以使用其他的函数来对文件进行处理。



二、标准IO函数使用

1、标准IO函数大全

类别	名称	作用
标准输入	`scanf`	从标准输入流读取格式化输入。
	`getchar`	从标准输入读取下一个字符。
	`gets`	从标准输入读取一个字符串，直到遇到换行符或指定字符数。
	`*fscanf`	当使用stdin作为参数时，从标准输入读取格式化输入。
	`getc`	当使用stdin作为参数时，从标准输入读取下一个字符。
	`fgetc`	当使用stdin作为参数时，从标准输入读取一个字符。
标准输出	`printf`	向标准输出流写入格式化输出。
	`putchar`	向标准输出写入一个字符。
	`*puts`	向标准输出写入一个字符串，并自动追加换行符。
	`*fprintf`	当使用stdout作为参数时，向标准输出写入格式化输出。
	`putc`	当使用stdout作为参数时，向标准输出写入一个字符。
	`*sprintf`	将格式化的数据写入字符串中。
	`fputc`	当使用stdout作为参数时，向标准输出写入一个字符。
标准错误	`perror`	打印错误消息，并将上一个函数错误原因输出到标准错误流。

类别	名称	作用
文件处理	`fopen`	打开文件，返回 `FILE` 指针。
	`freopen`	重新打开文件，与已有的 `FILE` 指针关联。
	`*fclose`	关闭一个打开的文件流。
读	`fread`	从文件流读取数据到数组中。
	`fwrite`	将数组中的数据写入文件流。
	`*fscanf`	从文件流中读取格式化输入。
	`*fprintf`	向文件流中写入格式化输出。
	`getc`	从文件流中读取下一个字符。
	`fgetc`	从文件流中读取一个字符。
写	`putc`	向指定的文件流写入一个字符。
	`fputc`	向指定的文件流写入一个字符。
	`*fgets`	从文件流中读取一行。
	`*fputs`	向指定的文件流写入一个字符串。
	`printf`	格式化数据并写入到标准输出。
	`*sprintf`	格式化数据并写入到字符串中。
	`snprintf`	格式化数据并写入到指定大小的缓冲区。
文件缓存	`fflush`	刷新输出缓冲区。
	`setbuf`	设置文件流的缓冲区。
	`setvbuf`	设置文件流的缓冲区策略。
文件定位	`*ftell`	返回当前文件位置指示器的值。

类别	名称	作用
	`*fseek`	设置文件位置指示器到指定位置。
	`fgetpos`	获取当前文件位置指示器的值。
	`fsetpos`	设置当前文件位置指示器的值。

- 学习顺序
  - 文件读写
    - fopen
    - fclose
    - fputc
    - fputs
    - fwrite
    - fgetc
    - fgets
    - fread
    - fscanf
    - fprintf
  - 标准输入和输出
    - sprintf
    - snprintf
    - gets：不推荐使用、建议使用fgets来代替
    - puts
    - getchar
    - putchar

1、fopen

（1）概念

用于尝试打开一个文件并返回该文件的FILE类型的指针(文件流指针)，方便用其他函数进行文件处理

（2）语法

```
FILE * fopen(const char *filename, const char *mode);
```

- pathname: 要打开的文件的路径及名称
- mode: 打开文件的方式

	具体解释	open() flag
r	以只读的方式打开一个文件，文件必须存在	O_RDONLY
r+	以读写的方式打开这个文件，文件必须存在。写入操作在读取之后的位置执行	O_RDWR
w	如果文件存在则清空再写，如果文件不存在则创建写入	O_WRONLY|O_CREAT|O_TRUNC
w+	如果文件存在则清空读写，如果文件不存在则创建读写	O_RDWR|O_CREAT|O_TRUNC
a	如果文件存在则追加写入，如果文件不存在则先创建在写入	O_WEONLY|O_CREAT|O_APPEND
a+	如果文件不存在则创建文件并追加写入和读取。 读取时从文件开头开始读取，写入时从文件的末尾开始写入 lseek偏移后不能进行写入操作，如果进入写入操作，则追加在末尾写入（不管写之前进行了	

	具体解释	open() flag
b	`"b"`表示“二进制（binary）”：在打开文件时，加上这个字符标志，表示文件以二进制模式打开，不对文件内容进行任何新行转换。在处理文本文件时，通常不需要使用`"b"`，但在处理二进制文件（比如图像、声音、编译好的程序等）时，必须使用`"b"`来确保数据正确读写.比如`rb`或`wb`	

- 返回值:
  - 成功: 返回一个文件的流指针
  - 失败: 返回NULL

（3）例子

```
FILE *fp;
fp = fopen("example.txt", "r"); // 打开名为"example.txt"的文件进行读取
if (fp == NULL) {
    // 出错处理
}
```

2、fclose

（1）概念

根据文件流指针进行刷新缓冲区以及关闭该文件的io流，释放对应资源。

（2）语法

```
int fclose(FILE *stream);
```

（3）例子

```
fclose(fp); // 关闭文件流
if (fclose(fp) != 0) {
    // 出错处理
}
```

- 成功:返回 0
- 失败:返回 EOF == -1

3、freopen

（1）概念

用于打开一个新的文件或重新打开当前文件

（2）语法

```
FILE *freopen(const char *filename, const char *mode,
FILE *stream);
```

（3）参数

- filename:想要打开的文件路径字符串地址。如果是打开当前文件，则传入NULL
- mode：打开模式。如 `w+` 、 `r` 等
- stream：指向 一个旧的 `FILE` 结构的指针，表示要重新打开的文件流

（4）返回值

- 新的`FILE`结构的指针，当freopen执行成功后，**原文件流指针将会失效**，应使用新的指针来操作文件

3、fgetc

（1）概念

从指定文件流指针中获取一个字符

（2）语法

```
int fgetc(FILE *stream);
```

- 参数
  - 文件流指针
- 返回值:

  成功: 返回获取到的字符转化成的int类型的数据。char转int可以参考ASCII码

  失败: 返回 EOF 到达文件末尾 EOF。EOF表示到达文件末尾，可以直接用在判断逻辑语句中
- 注意
  - 这个函数读取一个字符（无论是文本字符还是其他如换行符等特殊字符），并将文件位置指针向前移动一个字符。如果读取成功，返回的是读取到的字符的ASCII值；如果读取失败或者到达文件末尾，则返回EOF（通常是-1）

（3）例子

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

int main() {
```

```
    FILE *fp;
    int ch;

    fp = fopen("example.txt", "r"); // 打开文件用于读取
    if (fp == NULL) {
        perror("Error opening file");
        return(-1);
    }

    while((ch = fgetc(fp)) != EOF) { // 循环读取每个字符直到文件末尾
        putchar(ch); // 输出读取到的字符
    }

    fclose(fp); // 关闭文件
    return 0;
}
```

4、fgets

（1）概念

读取文件里的一行内容

（2）语法

```
char *fgets(char *s, int size, FILE *stream);
```

- 参数
  - s:读取到的数据存放的地址
  - size:预期读取的字节个数

        - 当'\n'之前的字节个数小于size，遇到换行符结束读取，但是换行符也会被当做一个字符被读到内存之中，并且会在最后一个字符后添加'\0'.
            - **实际**：如果读取换行符时不足size，那么后续的字符就是null。
        - 当'\n'之前的字节个数大于size,会读到size个大小的字节时停止读取。会在末尾加上一个'\0'。会将最后一个字节的字符给覆盖掉。机制会在读取完之后读写指针向前偏移一个字节。
            - 例子：给的size为5，实际文件一行有10个。那么会实际读取4个数据，第5个为\0.下一次读取会继续从文件的第5个字节继续读
            - **实际**：如果读取的内容大于size，那么保存　字符串的最后一个会是\0，
        - stream：文件流指针
- 返回值
    - 成功:返回读到的字符串的首地址
    - 失败:返回NULL(读到文件末尾也返回NULL)

（3）例子

```
char str[60];
if(fgets(str, 60, fp) != NULL) {
    // 使用str处理读取的内容
}
```

5、fputs

（1）概念

向一个文件中写入一个字符串

(2) 语法

```
int fputs(const char *str, FILE *stream);
```

- 参数
  - str: 想要写入的数据的地址。如果需要写入后换行需要字符串里有 `\n`
  - stream: 想要写入哪个文件
- 返回值
  - 成功: 返回一个非负数
  - 失败: 返回EOF（-1）

(3) 例子

```
int main(int argc, const char *argv[]) {
    FILE * fp = fopen("./1.txt", "w");
    if (NULL ==fp) {
        perror("fopen");
        return -1;
    }
    // 找到一个想要写入的数据的地址
    char buf[123] = "hello world";

    if (EOF == fputs(buf, fp)) {
        perror("fputs");
        return -1;
    }
    return 0;
}
```

6、fputc

（1）概念

写入一个字符到文件中

（2）语法

```
int fputc(int char, FILE *stream);
```

- 参数
  - `int char`：要写入的字符。尽管参数名是 `char`，但传递的参数应该是一个 `int` 类型的值，以便能够传递所有可能的字符和 `EOF`。
  - `FILE *stream`：目标文件指针，该指针应指向你想要写入字符的文件。
- 返回值
  - **成功**：返回写入的字符。
  - **失败**：返回 `EOF`。如果发生错误，还可以设置错误指示器，可以用 `ferror

7、fscanf

（1）概念

用于在指定文件流中查找满足指定格式的数据，并将数据保存在对应的变量中。**即能够从文件中提取需要的数据并保存到变量中**

（2）语法（fgets+strtok）

```
int fscanf(FILE *stream, const char *format, ...);
```

- 参数
  - `stream`：指向 `FILE` 对象的指针，该 `FILE` 对象代表一个输入流。
  - `format`：一个 字符串，包含了一个或多个以下项：普通字符 (除了 `%`), 和格式化指令。格式化指令识别输入项，并按照给定的形式将其转换和存储到提供的参数中。
  - `...`：额外的参数，每个参数都是一个指针，指向用来存储转换后的值的变量。
- 返回值
  - **成功读取的项数**：返回成功匹配并赋值的输入项数。
  - **文件结束(EOF)**：如果在读取任何数据之前遇到文件结尾，返回 `EOF` 。
  - **错误**：如果读取过程中发生错误，也返回 `EOF` 。

8、fread

（1）概念

以字节方式读取文件数据

（2）语法

```
size_t fread(void *ptr, size_t size, size_t count, FILE *stream);
```

- 参数
  - `ptr`：指向一块内存的指针，用来存储读取的数据。
  - `size`：每个数据项的大小，以字节为单位。
  - `count`：要读取的数据项个数。
  - `stream`：指向 `FILE` 对象的指针，代表一个打开的文件。

- 返回值
    - 返回实际读取的数据项个数，如果这个数小于`count`，可能是发生了错误或者达到了文件末尾。

9、fwrite

（1）概念

用字节方式写入数据到文件

（2）语法

```
size_t fwrite(const void *ptr, size_t size, size_t count,
FILE *stream);
```

- 参数
    - `ptr`：指向一块内存的指针，里面包含了要写入的数据。
    - `size`：每个数据项的大小，以字节为单位。
    - `count`：要写入的数据项个数。
    - `stream`：指向`FILE`对象的指针，代表一个打开的文件。
- 返回值
    - 返回实际写入的数据项个数，如果这个数小于`count`，可能是发生了错误或者达到了文件末尾。

（3）例子

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

int main() {
    FILE *infile, *outfile;
    char buffer[512];
```

```
    size_t bytesRead, bytesWritten;

    // 打开源文件并检查是否成功
    infile = fopen("source.bin", "rb");
    if (infile == NULL) {
        perror("Error opening source file");
        return -1;
    }

    // 打开目标文件并检查是否成功
    outfile = fopen("destination.bin", "wb");
    if (outfile == NULL) {
        perror("Error opening destination file");
        fclose(infile);
        return -1;
    }

    // 读取输入文件并写入输出文件
    while ((bytesRead = fread(buffer, 1, sizeof(buffer),
infile)) > 0) {
        bytesWritten = fwrite(buffer, 1, bytesRead,
outfile);
        if (bytesWritten != bytesRead) {
            perror("Error writing to destination file");
            break;
        }
    }

    // 关闭文件
    fclose(infile);
    fclose(outfile);

    return 0;
}
```

8、fprintf

（1）概念

用于将按照指定格式书写的字符串写入到文件中

（2）语法

```
int fprintf(FILE *stream, const char *format, ...);
```

- 参数
  - `stream`：指向 `FILE` 对象的指针，该 `FILE` 对象代表一个输出流，用于接收格式化的输出内容。
  - `format`：一个以 null 结尾的字符串，提供了输出文本的格式。字符串由普通字符组成，以及格式说明符。格式说明符会被后续的附加参数所取代。
  - `...`：根据格式字符串中的格式说明符，依次提供相应的附加参数，它们的数目和类型必须与格式字符串中的格式说明符一一对应。
- 返回值
  - **成功写入的字符数**：返回成功写入到流中的字符总数。
  - **错误**：如果发生输出错误，返回一个负数。

10、fseek

（1）概念

设置文件光标进行读写的位置

（2）语法

```
int fseek(FILE *stream, long offset, int whence);
```

- 参数
  - stream: 目标文件流指针
  - offset:
    - 该数为负数，向前进行偏移，如果偏移出了文件的开头，会报错返回如果
    - 该数为正数，向后进行偏移，如果偏移出了文件的末尾，会扩大文件，用'\0'来做填充。那么此类的文件被称为 空洞文件
    - 注意：如果偏移后没有对其进行任何写入操作，内核认为该偏移无效，不会扩大文件大小。
  - whence: 基准位置-----根据哪一个位置进行偏移
    - 0:即SEEK_SET 根据文件开头进行偏移
    - 1:即SEEK_CUR: 根据用户当前位置进行偏移
    - 2:即SEEK_END: 根据文件末尾进行偏移
- 返回值
  - 成功: 返回 0
  - 失败: 返回 -1

（3）例子

```
fseek(file, 6, 0);
```

11、fflush

（1）概念

它通常用于刷新输出缓冲区，确保所有缓存的输出数据被写入其对应的文件或终端中

（2）语法

```c
#include <stdio.h>
int fflush(FILE *stream);
```

- 返回值：成功返回0，失败为EOF，即-1

（3）例子

```c
FILE *file = fopen("example.txt", "w");
if (file != NULL) {
    fputs("这是一个测试字符串。", file);
    // 刷新文件缓冲区，确保内容已经被写入文件
    fflush(file);
    fclose(file);
}
```

12、ftell

（1）概念

获取当前读写光标的偏移量

（2）语法

```c
long ftell(FILE *stream);
```

- 参数：
  - stream 目标文件流指针
- 返回值：
  - 成功: 返回偏移量（是根据文件开头来计算的）
  - 失败: 返回-1

（3）例子

```
通常与fseek(fp, 0, SEEK_END)来配合使用测量文件当前的大小
//printf("测试函数fopen");
    //printf("测试函数fclose");
    FILE *file1 = fopen("data1.txt","r");
    FILE *file2 = fopen("data2.txt","r");
    //printf("%p\n",file);
    if(file1 ==NULL){
    printf("读取文件失败\n");
    }else{
    //fclose(file);
    }
   // ftell fseek
   fseek(file2,0,SEEK_END);
   long size = ftell(file2);
//printf("%ld\n",size);
   fclose(file1);
   fclose(file2);
```

注意：如果要测试文件大小，要注意，有可能测试的时候测试数据会比实际大小要多1，因为有个换行符(linux系统自动添加)

```
int data;
int count=0;
while((data= fgetc(file2))!=EOF){
    count++;
    printf("%c %d\n",(char)data,data);
}
printf("读取结束:读取次数:%d\n",count);
```

13、rewind

（1）概念

- 设置文件读写位置为开头

（2）语法

```
void rewind(FILE *filestream);
```

- 参数：需要设置位置的文件流指针
- 返回值：void没有返回值。可以通过ftell获取位置来判断是否设置成功，或者使用ferror来读取该文件流指针状态码是否为非0也可以查看是否设置成功。

（3）代码

```
FILE *fp = fopen();
rewind(fp);//设置位置
//判断是否设置成功
1. 通过ftell来判断
int pos = ftell(fp);
if(pos !=0){
    //没有设置成功
}
2.通过ferror () 来查看文件流指针是否状态码异常
int status = ferror(fp);
if(status!=0){
    有异常
}
```

13、puts

（1）概念

用于将**字符串**输出到标准输出(stdout)，并在末尾自动添加换行符。

（2）语法

```
int puts(const char *s);
```

- 参数
  - `s`：指向要输出字符串的指针
- 返回值
  - 成功：返回一个非负值。
  - 错误：返回EOF。

14、putchar

（1）概念

用于将单个字符输出到标准输出(stdout)

（2）语法

```
int putchar(int char);
```

- 参数
  - `char`：要输出的字符。
- 返回值
  - 成功：返回输出的字符。
  - 错误：返回EOF

15、getchar

（1）概念

用于从标准输入(stdin)的缓冲区里读取并返回一个字符。

（2）语法

```
int getchar(void);
```

- 返回值
  - 成功：返回读取的字符,int类型数据。
  - 错误或文件结束：返回EOF。

16、getc

（1）概念

用于从指定的输入流读取并返回下一个字符

（2）语法

```
int getc(FILE *stream);
```

参数

- `stream`:要读取字符的输入文件流。

返回值

- 成功：返回读取的字符。
- 错误或文件结束：返回EOF。

跟fgetc的对比：推荐使用fgetc，getc不同的编译器可能是一个宏，也可能是一个函数，不稳定。

17、putc

（1）概念

用于将一个字符写入指定的输出流。

（2）语法

```
int putc(int char, FILE *stream);
```

参数

- `char`：要写入的字符。
- `stream`：要写入字符的输出文件流。

返回值

- 成功：返回写入的字符。
- 错误：返回EOF。

18、sprintf/snprintf

（1）概念

用于将格式化数据写入字符串。snprintf会额外的增加对输入字符串的长度限制，其他跟sprintf一样

（2）语法

```
int sprintf(char *str, const char *format, ...);
int snprintf(char *str, size_t size, const char *format,
...);
```

参数

- `str`：目标字符串的数组。
- `format`：格式化字符串。
- `...`：变长参数列表，对应于格式化字符串。
- `size`：要写入的最大字符数（包括终止空字符）。

返回值

- 成功：返回写入字符串的字符总数（不包括终止空字符）。如果是snprintf，那么如果这个值大于`size`，实际上不会将这么多的字符写入`str`
- 错误：如果编码错误则返回负数。

注意：

- 这两个函数都会自动填充'\0',所以建议在使用时尽量比预期的字节数+1，用于保存结束字符

2.1 IO函数的一般使用流程

- 1.使用fopen打开要操作的一个文件（得到其文件流指针）
- 2.使用文件流指针以及配套的函数来实现文件内容的读取或写入
- 3.操作结束后调用fclose关闭文件（通过文件流指针）

三、常用标准IO函数总结

类别	函数名	说明
文件读	**fgets**	一行一行的读
	fread	一般读取整个文件，适合二进制文件，比如音频、视频，文档等
	fgetc	一个一个字符的读
	fscanf	按照指定格式读取数据
文件写	**fputs**	一行一行的写
	fwrite	一次性写人到整个文件：适合二进制文件，比如音频、视频，文档等
	fputc	一个一个字符的写
	fprintf	按照指定格式将数据写入到文件中
标准输入	**scanf**	输入指定类型的数据
	getc	从标准输入获取一个字符
	getchar	从标准输入获取一个字符
标准输出	**puts**	输出一个字符串
	putchar	输出一个字符
	putc	输出一个字符
	printf	输出指定格式的数据

四、IO学习目标总结

- 什么是标准IO 和IO 流
  - 标准IO:是指c语言内置标准IO函数库，主要用于完成程序和设备之间数据的输入和输出。函数库里面提供了大量的函数用于完成数据的读写操作。设备可以是终端或文件。针对不同的设备会调用不同的内置函数
  - IO流：是指为了方便描述程序和设备之间数据流向而设计的一个抽象概念。包含输入流和输出流。
    - 输入流：指数据流向从设备流向到程序。我们调用相关函数的时候，系统就会讲设备里的数据通过主板IO线流向程序中
    - 输出流：指数据流向从程序流向到设备。我们调用相关函数的时候，系统就会讲程序里的数据通过主板IO线流向设备中
- IO有什么用，主要目的是什么
  - **能够将设备里输入的数据或文件里的数据读取到程序中**
  - **能够将程序里的数据输出到终端或文件以及反向操作。**
- 什么是文件流指针
  - **总结：文件流指针是一个文件在c语言程序里的表现形式，本质是一个结构体指针。每一个文件都有自己的文件流指针。开发者可以通过调用io函数（比如fopen）拿到文件流指针，并利用该指针和其他io函数完成数据的交互**
- IO的一般使用流程是什么
  - **1.利用fopen函数获取要操作文件的文件流指针**
  - **2.利用其他内置函数（比如fgets或fputs）完成文件数据的读和写**
  - **3.操作都完成之后调用fclose来关闭文件流指针，释放所占空间。即关闭文件。**
- IO提供的函数有哪些，哪些是常用函数
- 如何理解IO缓存机制

- IO应用场景有哪些，和数据库本质区别是什么

随堂练习

1. 记录用户的登录和二级菜单操作
2. 给文件的每个a字符后面追加一个helloworld