1. **stdin、stdout 、stderr(I/O操作):系统自动打开的流文件，不需要fopen**
2. stdin是标准输入，一般指由键盘输入到缓冲区里的内容，由perl语言实现。

**stdin(键盘到内存)、stdout(内存到屏幕)**是标准输入输出设备的文件句柄宏定义，可以理解为**它们就是文件**，而这个文件和标准输入设备(键盘)、标准输出设备(屏幕)建立了某种关联。如stdout:当数据写到这个文件里面的时候，屏幕就会通过既定的方式把你写进去的东西显示出来.

stdin是[标准输入流](https://www.baidu.com/s?wd=%E6%A0%87%E5%87%86%E8%BE%93%E5%85%A5%E6%B5%81&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank), 本质上跟文件流的概念差不多：只不过 (1)**文件流是你自己用fopen建立**的, (2)**而stdin是系统帮你内建好的,并不需要fopen**，你从键盘输入的字符都被放进stdin, 用scanf等函数就可以从stdin里面取出字符.对比一下就是, 我们用的**scanf就是对stdin在操作； 而fscanf是对文件流操作**。

(2) **访问stdin**，可用stdio.h中提供的以下几个函数：

1)scanf、getchar来从stdin流中获取指定的数据。scanf根据第一个参数指定的格式符将数据读入后，存储到参数指定的变量内存地址中；

getchar从stdin流中读取一个字符并返回。

1. 可以使用一般的访问输入流的函数来访问stdin，比如：fscanf、fgetc，使用方法和scanf、getchar类似，只是多了一个参数指定文件流。当然，**scanf、getchar、gets的具体实现就是通过调用fscanf、fgetc、fgets**，这样可以提高代码利用率。
2. 可以用fflush和rewind这两个函数来刷新stdin，来清空缓冲区中已有的内容，不过这种方法不是标准方法，当stdin被[重定向](https://www.baidu.com/s?wd=%E9%87%8D%E5%AE%9A%E5%90%91&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)到文件时，这会导致指针重新指向文件开头。

二、从**stdin缓冲区读取一行字符串：gets、getline、fgets(读文件)**

1)gets(),原型为**char \*gets(char \*s);**

gets从stdin中读入一行内容到s指定的buffer中，当遇到换行符或EOF时读取结束。读取成功时，返回值为字符串的起始地址；失败时返回NULL。 注意： **gets会将行末尾的'\n'字符(即回车)或EOF替换成字符串结束符'\0'**，这样，**gets读取的内容中就不会包括'\n'字符了**。（**区别fgets:末尾添加‘\0’,但不会替换‘\n’**）

2)fgets(),原型为**char \*fgets(char \*s, int size, FILE\*stream);**

fgets从stream中读取最多size-1大小的内容到s指定的buffer中，当遇到换行符或EOF时读取结束。读取成功时，返回s地址；失败时返回NULL：即**当n<=0 时返回NULL，即空指针； 当n=1 时，返回空串""；如果读入成功，则返回缓冲区的地址；如果读入错误或遇到文件结尾(EOF)，则返回NULL**。 需要注意的是，**fgets会在所读取的内容后面添加'\0'，这样，fgets读取的内容中会包括行末尾的'\n'字符**。

3)getline(),原型为：**ssize\_t getline(char \*\*lineptr,size\_t \*n, FILE \*stream);**

getline从stream中读取一行内容到\*lineptr所指定的buffer中，当遇到换行符或EOF时读取结束。＊n是\*lineptr所指定的buffer的大小，如果＊n小于读入的内容长度，getline会自动扩展buffer长度，并更新\*lineptr和\*n的值。读取成功时，返回读入的字符个数；失败时返回-1。需要注意的是，**getline读取的内容中会包括行末尾的'\n'字符**。

**getline不是C库函数，而是gcc的扩展定义或者C++库函数**。 C有 fgets(), gets() 函数，gcc编译器扩展定义了getline()函数.用于读取一行[字符](https://baike.baidu.com/item/%E5%AD%97%E7%AC%A6" \t "https://baike.baidu.com/item/getline%E5%87%BD%E6%95%B0/_blank)直到换行符,包括换行符。它会生成一个包含一串从输入流读入的字符的字符串，直到以下情况发生会导致生成的此字符串结束。1）到文件结束，2）遇到函数的定界符，3）输入达到最大限度。

三、从**stdin读取一个字符：getchar、getch**

(1)getchar:原型：**int getchar(void);** 从标准输入 stdin 获取一个字符（一个无符号字符）。getchar函数的返回值是用户输入的第一个字符的[ASCII码](https://baike.baidu.com/item/ASCII%E7%A0%81" \t "https://baike.baidu.com/item/getchar%28%29/_blank),如出错返回-1,且将用户输入的字符回显到屏幕

**用户输入的字符被存放在键盘缓冲区中,直到用户按回车为止(回车字符也放在缓冲区中).当用户键入回车之后,getchar才开始从stdio流中每次读入一个字符。如果用户在按回车之前输入了不止一个字符,其他字符会保留在键盘缓存区中,等待后续getchar调用读取.也就是说,后续的getchar调用不会等待用户按键,而是直接读取缓冲区中的字符,直到缓冲区中的字符读完后,才等待用户按键.**

统计从键盘输入一行字符的个数。  
#include <stdio.h>  
main(){  
int n=0;  
printf("input a string:\n");  
while(getchar()!='\n') n++;  
printf("%d",n);  
}

getchar()是到缓冲区读取第一个字符。首先,从键盘输入许多字符,最后回车后,所有字符包括回车放到缓冲区，这时才开始循环执行getchar(),读入一个字符就判断是否回车,不是就继续读,是回车就退出循环(之所以说是顺序接受一行字符,是因为**getchar取的始终是缓冲区中第一个字符**,每取一个缓冲区的字符少一个,也就是顺序读取了)

注意：**getchar 函数的返回值类型是 int 型。**当这些函数读取出错或者读完文件后，会返回 EOF.EOF 是一个宏，标准规定EOF的值必须是一个 int 型的负数常量,通常编译器都会把 EOF 定义为 -1。 若把getchar返回的int赋值给char c的时候，会发生截断，和EOF比较时又会升级为int，可能会发生如下错误：

1) 某些合法的字符被“截断”了以后，恰好等于-1，导致程序在复制的过程中发生了中断。

2) 前面的C不可能取值为EOF，导致程序产生了一个死循环。

1. getch:原型：**int getch(void);** 所在头文件：[conio.h](https://baike.baidu.com/item/conio.h" \t "https://blog.csdn.net/eric_e/article/details/_blank)，从控制台**读取一个字符，但不显示在屏幕上(非C的库函数)**

getch();并非标准C中的函数，所以在使用的时候要注意程序的可移植性。国内C语言新手常常使用getch();来暂停程序且不知道此函数来源，建议使用getchar()或者使用<stdlib.h>头文件中的system("PAUSE")调用系统命令暂停;（如果情况允许）代替此功能或更换一款编译器。

[getch](https://baike.baidu.com/item/getch)与getchar基本功能相同,差别是**getch直接从键盘获取键值,不等待用户按回车,只要用户按一个键,getch就立刻返回**, getch返回值是用户输入的ASCII码,出错返回-1.输入的字符不会回显在屏幕上.getch函数常用于程序调试中,在调试时,在关键位置显示有关的结果以待查看,然后用getch函数暂停程序运行,当按任意键后程序继续运行.

四、**scanf： %d时忽略先导空白符**

(1)scanf:从stdin流读数据时遇到空白符就停止的输入函数（空白符：指空格符、制表符、回车符)。 scanf的返回值：成功赋值的的变量个数；发生错误时返回EOF，即返回值为-1, 如： 1）ret为scanf成功输入的数值的个数：有几个格式符，就成功几个数,若成功输入,则ret=scanf("%d%d",&a,&b)=2。

2) while ((ret = scanf("%d", &i)) != EOF) //ret个数为EOF即-1 时，scanf才会退出（又因为scanf输入不成功其返回值为0， 即ret为0，不会等于-1，故会不断地循环打印，只有ctrl+z才停止）

3) fflush/rewind : 清空缓冲区数据

(2) scanf时系统把空格字符作为输入字符串之间的分隔符,scanf函数在读取字符串时遇到空格，就会认为读取结束。如：

char str[13];

scanf(“%s”,str);

若输入以下12个字符： How Are You？（输入时，之间用了空格）

系统只会将How送到str数组中，此时str数组中为How其余10位全为\0

(3)如果scanf中%d是连着写的如**“%d%d%d”,在输入数据时，数据之间不可以用逗号分隔，只能用空白字符（空格、tab键、回车键）分隔**，例如“2 （空格）3（tab） 4” 或 “2（tab）3（回车）4”等。若是“**%d,%d,%d”，则在输入数据时需要加’,’** 如“2,3,4”,要和前面格式一一对应。

(4)输入设备（键盘）连接着一个叫“缓冲”的东西，把缓冲认为是一个字符数组。当你的程序执行到scanf时，会从你的缓冲区读东西，如果缓冲区是空的，就阻塞住，等待你从键盘输入。现在假设缓冲区里有：abcd\n1234\n (其中\n是回车符)执行：scanf("%s",name);的时候，由于scanf是读数据直到看见空白符（空白符：指空格符、制表符、回车符）就停止的输入函数。所以执行后，把abcd存到了name中。缓冲区于是变成了 ： \n1234\n

接下来的执行就有问题了，如果遇到了：scanf("%d",&number);怎么办？因为遇到了回车符，它并不是一个数字，所以**scanf还有一个特性，就是忽略先导的空白符**。不管是有几百个回车也好，几万个空格也罢，**只要它们连续地出现在缓冲区的开头，就统统忽略他们**。然后再读有意义的字符。于是1234被读入number。即用scanf()读数据，**只要不是%c就行**。然后输入的时候，随便输入回车、空格、制表符，然后“回车”确认。会发现程序依然提示等待你输入。就是因为它忽略掉所有前导空白符之后发现缓冲区是空的！于是仍在阻塞，等待你输入。

回到刚刚，当缓冲区还是：\n1234\n的时候，如果遇到了：scanf("%c",&sex);应该怎么办呢？你说，那好办呀，不是说了忽略前导空白符吗？跳过回车读'1'呀！想法是好的，可这只针对你的程序这一种情况。如果我编写的程序就是统计用户输入了多少个回车呢？所以**对scanf来讲跳过前导空白符有个例外，当参数是%c的时候，就把缓冲区的第一个字符返回回去，不管是什么字符。**

当执行scanf("%s",name)时，要求从键盘输入，于是你输入了"abc"，然后“回车”。缓冲区里此时是：abc\n ,scanf把abc拿走了，留下了\n，缓冲区里现在就剩下\n 于是，则下一个scanf ("%c",&sex)时， 便会先读取之前的\n。

程序里如果我们想一次读入一个英文句子:I am a student.如果你用scanf读的话，只能读出"I"，想读出后面的东西要不断调scanf。此时需要用gets，这个函数不管是什么一律读进来，直到遇到回车符才停下。

(5)重点！！！ **scanf、gets区别** :

1) scanf不能接受空格、制表符Tab、回车等，即**scanf函数只能读取到空格、制表符Tab、回车的前面**，（scanf函数在读取时遇到空格，就认为读取结束），而**gets函数可以读取空格、制表符Tab** **。 但仅仅只会将回车替换为‘\0’。**

2) scanf函数能对各种类型进行输入，而gets函数只能对字符串进行输入。

3) 针对于**字符串**：使用**scanf和gets输入字符串时都会在结束后自动添加’\0’**

scanf ：当遇到回车，空格和tab键时，会自动在字符串后面添加’\0’，再一起放到数组中，但是**回车，空格和tab键仍会留在输入的缓冲区中**。   
gets：可接受回车键及之前输入的所有字符(会将行末尾的'\n'字符或EOF替换成字符串结束符'\0')， 并**自动换行,回车键不会留在输入缓冲区中**。

(6)

scanf给变量i赋值时要用取地址符&i：因为要在被调函数scanf()中改变主函数中i的值，故要取地址传过去。

五、**输出 printf、puts、putchar**

(1)**int printf ( const char \* format, ... );**

printf()是C语言标准库函数，用于将格式化后的字符串输出到标准输出。标准输出，即标准输出文件，对应终端的屏幕。正确则返回输出的字符总数，错误返回负值。

%c:字符型。可以把输入的数字按照ASCII码相应转换为对应的字符

%s:字符串。输出字符串中的字符直至字符串中的空字符（字符串以空字符’\0’结尾）

%p:以16进制形式输出指针

%d:输出十进制有符号32bits整数

1. **int puts(const char \*string);**

puts()函数用来向标准输出设备（屏幕）输出字符串并换行，即送一字符串到流stdout中,具体为：把字符串输出到标准输出设备，**将'\0'转换为回车换行**。若字符串成功输出，则返回值为非负数（通常为0）；若字符串输出失败，则返回值为EOF（通常为-1）。

其调用方式为，puts(s)；其中s为字符串字符（字符串[数组](https://baike.baidu.com/item/%E6%95%B0%E7%BB%84/3794097" \t "https://baike.baidu.com/item/puts/_blank)名或字符串[指针](https://baike.baidu.com/item/%E6%8C%87%E9%92%88/2878304)）。

区别printf：**puts函数在输出时，会自动换行，而printf不会。**

(4)printf**打印百分号**要使用 **%%** 转义

(5) **int putchar(int c)**:该函数的功能是在stdout上(向终端)**输出单个字符**。

用 法: putchar(c);其中c可以是被单引号（英文状态下）引起来的一个字符，也可以是介于0~127之间的一个十进制整型数（包含0和127），还可以是事先用char定义好的一个字符型变量。

1)当c为一个被单引号（英文状态下）引起来的字符时，输出该字符（注：该字符也可为转义字符）；

2)当c为一个介于0~127（包括0及127）之间的十进制整型数时，它会被视为对应字符的ASCII代码，输出该ASCII代码对应的字符；

3)当c为一个事先用char定义好的字符型变量时，输出该变量所指向的字符。

4)该函数将指定的表达式的值所对应的字符输出到标准输出终端上。表达式可以是字符型或整型，它每次只能输出一个字符。  例如：“putchar('#')”输出字符“#”。

5)返回值: （1）当输出正确的时候，返回输出字符转换为的unsigned int 值；

（2）当输出错误的时候，返回 EOF（End of file）文件结束符

if(putchar(c)==EOF)

　　 {

printf("output error:%m\n");

exit(0);

}

六、文件操作

1.文件包括：

(1)程序文件(Windows中后缀为.c的源文件，后缀为.obj的目标文件， 后缀为.exe的可执行文件)：文件内容是程序代码。

(2)数据文件(如程序运行过程中**输出到磁盘或其他外部设备**的数据)：**文件内容是供程序运行时读写的数据**。文件操作即针对于数据文件。

2.什么是文件？

(1)操作系统把各种设备都统一作为文件来处理，即每一个与主机相连的输入输出设备都可看作1个文件， 如键盘: 输入文件 ； 屏幕、打印机: 输出文件

(2)文件：指**存储在外部介质**上数据的集合。

(3)**文件名：也叫文件标识，包含3部分：文件路径、文件名主干、文件后缀**

如：D:\计算机科学与技术\C前期作业\file1.txt

文件路径：D:\计算机科学与技术\C前期作业； 文件主干：file1

3.所有I/O操作均只是从程序移进或移出字节：**流也称为字节流**

4. **FILE是结构体类型**，由系统typedef的名称，并在**#include<stdio.h>头文件中声明**了FILE的数据结构

每一个FILE类型的变量对应一个文件信息区，在文件信息区中存放的是该文件的相关信息。一般通过声明一个**FILE类型的指针变量来指向此文件(指向的是内存缓冲区中文件信息区的开头)。**

5.用fopen函数打开数据文件：给文件**在内存中建立**相应的**文件信息区**(存放文件的相关信息) 和 **文件缓冲区**(暂时存放输入输出的数据)

用fclose函数关闭数据文件：即**撤销文件信息区和文件缓冲区**，使文件指针不再指向该文件，无法通过该指针对原来的文件进 行读写操作。

6. 文件缓冲区：系统在内存为每一个正在使用的文件开辟一个文件缓冲区

**缓冲区即为内核中的一段内存，由内核进行维护、管理。**

内存向磁盘输出的数据要先放到缓冲区：磁盘速度慢，文件先放到缓冲区。

标准输入(stdin) ， 标准输出(stdout)：也均是用缓冲区原理

缓冲区什么时候会把文件写到磁盘里？

(1)**用户主动刷新 fflush(stdin); rewind(stdin);** 刷新输入流时

(2)**关闭文件**（关闭前会写进磁盘，否则文件会丢失）

(3)**缓冲区写满**时，会一起送到磁盘去

7. **缓冲区大小是4KB，即4096字节**

一个文件哪怕只有1个字节也要占用磁盘大小4KB

一个文件至少占用磁盘空间是4KB，即4096字节（**便于管理，否则碎片太多了**）

Windows打开txt文档，光标会默认在文件头。

8.

VS是用绝对路径来启用程序

VS找文件的路径是和main.c在同一个路径的

VS中向main函数传参，对项目右键点击-属性-调试-命令参数

在VS的调试命令中写：file.txt 123 abc（即传了三个参数，用空格隔开）

若传入的参数本身就包含空格，故不能当成引入了多个参数：故要用双引号引起来

“file.txt 123 abc”：此时整体为一个参数

argv[0] 指向自身,此时argc为1（即没有传参时）：此时显示绝对路径

当传入一个参数，argv[1]指向这个参数，此时argc为2

\是转义字符，在VS中传参时，若**写绝对路径时**，如E:\file.txt，则fopen要写成fopen(“E:**\\**file.txt”,”r+”); 即**路径处要写两个反斜杠**，只写一个\会当成转义字符，即会把\f当成一个字符。

9.

(1)perror函数原型: **void perror (const char  \*str);** 能侦测出错的原因

(2)fopen函数原型：**FILE \* fopen(const char \* path, const char \* mode);**

成功，返回值：文件的起始地址；失败：返回NULL

(3)fclose函数原型：**int fclose(FILE \*fp);** 关闭数据文件

成功，返回值为0；失败：返回EOF

**(读、写1个字符，并把位置标识符往前移动)**

(4) fgetc函数原型：**int fgetc(FILE \*fp)；**从fp指向的文件 读入一个字符，并把位置指针标识符往前移动。成功，返回值为所读的字符 ； 失败：返回EOF

fputc函数原型：**int fputc（char ch ,FILE \*fp）;**把1个字符ch写到fp所指向的文件，并把位置指针标识符往前移动。成功，返回值就是输出的字符； 失败：返回EOF

**(读、写1个字符串，一次读一行，且每行读到换行\n时停止读取，并在行尾赋结束符’\0’)**

(5)fgets函数原型：**char \*fgets(char \*buf ,int bufsize ,FILE \*fp)；**每次最多读取(bufsize-1)个字符，并在最后加一个’\0’字符，再把这bufsize个字符放到buf所指的字符数组str中。 注意：**如果文件中的该行，不足bufsize-1个字符，则读完该行就结束；如果该行（包括最后一个换行符）的字符数超过bufsize-1，则fgets只返回一个不完整的行(即返回只读了bufsize-1个字符的行)**，但是，缓冲区总是以结束符‘\0’结尾，故对fgets的下一次调用时会继续读该行。若成功，返回地址str；失败，返回NULL。fgets读取的内容中会包括行末尾的'\n'字符,且fgets会在所读取的内容最后面添加'\0'。

fputs函数原型：**int fputs(const char \*str ,FILE \*fp)；**把str所指向的字符串写到fp所指向的文件中。成功，返回非负值； 失败：返回EOF

**(读写1组数据,如数组或结构体变量所包含的值):fread、fwrite以二进制形式进行读写**

(6)fread函数：**size\_t fread(void \*buffer,size\_t size,size\_t count,FILE \*fp);**

从文件中读一个数据块。**返回值:实际读取到**的项的个数（小于或等于count），如果不成功或读到文件末尾则返回 0。注意：fread不会赋结束符。

fwrite函数:**size\_t fwrite(const void \*buffer,size\_t size,size\_t count,FILE \*fp);**

向文件写一个数据块。返回值：返回实际写入的数据块个数（小于或等于count）

**(对文件进行格式化输入输出)：读写对象不是终端，而是文件**

(7) fprintf函数:**int fprintf (FILE \*fp,const char \*format, [argument...])；**

输入到文件中，所有格式均按字符串存到文件中。

fscanf函数: **int fscanf(FILE \*fp,const char \*format,[argument...]);**

把文件中字符串按照各自的格式(整型、字符型、浮点型等)读取出来存到对应的变量中。

(8)**重点区别：fprintf、fscanf 与fread、fwrite：**

1)**fscanf、fprintf是文本方式读写**：能看见：输入时要将文件的**ASCII码转换成二进制再保存到内存**变量中；输出时又要将**内存中的二进制转换成字符**，故会花费较多时间。

2) **fread、fwrite是二进制方式读写**：看不见是乱码：向磁盘写数据时可直接将内存中的数据原封不动、不加转换地复制到磁盘中，读取时也是可直接将磁盘数据读入内存中。

**故若在内存与磁盘频繁交换数据的情况下，最好不要用fscanf、fprintf，而是用fread、fwrite来进行二进制的读写。**

(9)fseek函数原型：**int fseek(FILE \*fp, long offset, int fromwhere);**

即为fseek(文件类型指针，位移量，起始点)：改变文件的位置指针（即光标）。

成功返回值为0 ；失败则不改变fp指向的位置，并返回非0值

(打开txt文档，光标会默认在文件头)

注意fseek是以起始点为基点，向后移动的字节数。 若要向前偏移，则加负号，如：以文件当前位置向前偏移2位：fseek(fp , -2 , SEEK\_CUR)；

文件 写和读、读和写之间都要有fseek刷新位置指针（即要把光标偏移到文件开头）

(10) 格式化读写有：printf、scanf、fprintf、fscanf、sprintf、sscanf

其中**sprintf和sscanf：并不是文件操作**，因为参数中没有文件指针

**int sprintf(char \*str, const char \*format, ...);**

**int sscanf(char \*str, const char \*format, …);**

**sprintf、sscanf：用于字符串和其他类型之间的相互转换！！！！！**

1)**sprintf常用于把不同类型的数据变成字符串，再存到buf数组中：数据的字符串拼接。区别于strcat只能拼接2个字符串，sprintf不限制拼接的个数。**

2)**sscanf是把buf中的字符串，根据格式读取为不同的数据类型。**

3) 其中**sprintf和sscanf：并不是文件操作**，因为参数中没有文件指针

区别：fprintf 是将格式化后的字符串写入到文件流stream 中

而sprintf 是将格式化后的字符串写入到数组buf中。

(11)ftell原型: **long ftell(FILE \*stream);** //用来获得文件流当前的读写位置

返回值打印要用ld：指距离开头有多少个字节(当然位置到开头的距离)。

10.

getchar要进入缓冲区，按回车才停止读取（故每次要刷新缓冲区rewind)，遇到\n换行

getch不经过缓冲区，直接读取并立刻返回，遇到 \r\n才换行，输入的字符不会显示在屏幕上（故适合设置成输入密码，因为不会显示出密码是什么，保密）

11.

区别**r+(文本文件)** 和 **rb+(二进制文件)**：

1. **读写整型数时：**

r+： 整型数按字符ASCII码处理

rb+： 当写整型数时，要用二进制的可读可写rb+，按整型数处理

1. **处理换行时(txt文本要2个字符:\r\n,才能解析为换行**)：

r+: 写入\n,实际存储的是\r\n，读取时，\r\n又会变成\n

rb+：写入\n,实际存储的就是\n，读取时，\n读取出来就是\n

12. 重点！！！qsort 的函数原型：快速排序(快排的非递归实现！！)

**void qsort(void \*base, size\_t num， size\_t size, int (\*compar)(const void \*, const void\*))**；

用于**对数组(包括结构体数组)进行排序**，该函数不返回任何值。

各参数：1.待排序数组首地址 2.数组中待排序元素数量 3.各元素的占用空间大小

4.指向函数的指针, 用于确定排序的顺序 (此函数自己来写如何排序)

**compare函数原型 ：compare( (void \*) & elem1, (void \*) & elem2 );**

|  |  |
| --- | --- |
| Compare 函数的返回值 | 描述 |
| < 0 | elem1将被排在elem2前面 |
| 0 | elem1 等于 elem2 |
| > 0 | elem1 将被排在elem2后面 |

(2)**对结构体数组中的元素进行排序(通过数组元素的某个成员确定排序顺序，如num成员)**

struct student

{

int num;

char name[20];

float score[3];

}stu[N];

qsort(stu, N, sizeof(stu[0]), compar);//待排结构体数组首地址为stu；有N个待排元素；每个元素所占大小为sizeof(stu[0])；compar比较两个元素来确定如何对两元素排序

int compar(const void \* a, const void \* b)

{

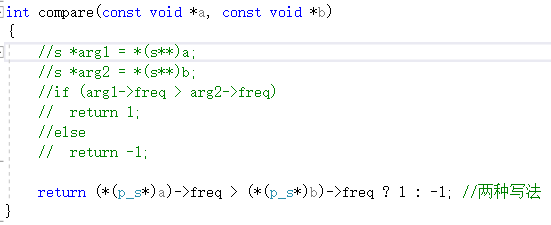
return (\*(struct student\*)a).num > (\*(struct student\*)b).num ? 1 : -1;

}

**有两点注意事项：**

**注意(1)：因为qsort()要改变排序，故被调函数中要用二级指针才能改变主调函数中的一级指针，所以compar函数的参数a、b均是二级指针，每次都要先解引用再写排序算法。**

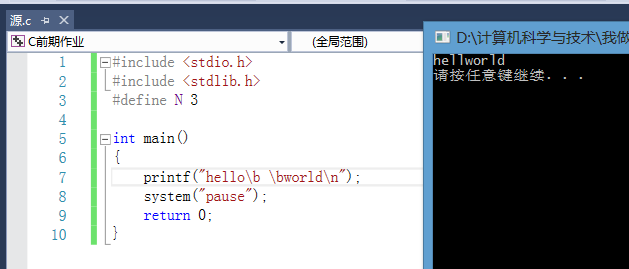
**注意(2)： compar函数要写在main函数的外面。**

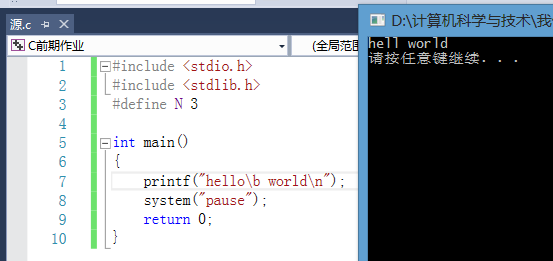


13.

**printf("\b \b"); 删除**（中间的空格是覆盖掉下一个字符）：

即 \b把光标移到前面，空格给覆盖掉，再把光标移回去，达到删除目的

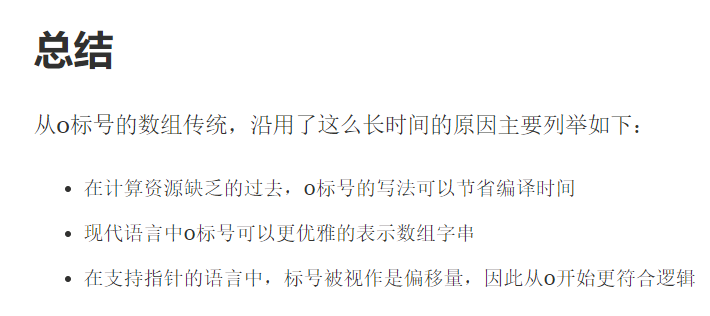




14.

**为什么数组标号是从0开始的？**

在C语言中有指针的概念，而**指针数组标号实际上是一个偏移量而不是计数作用**。例如对于指针p，第N个元素是\*(p+N)，指针指向数组的第一个元素就是\*(p+0)



七: Linux下的文件操作：

**Linux下基于文件描述符的文件操作(可实现非缓冲)**

1. 区别：之前学的文件操作的 标准C接口，都是有缓冲机制的，不能实现非缓冲。

Linux下的接口也都默认是有缓冲机制的，但也可以实现非缓冲 (flag参数为O\_SYNC时)

Linux下文件，不分文本模式r+和二进制模式rb+

1. **文件描述符fd(整型数)：Linux下的接口都是为文件描述符服务的，不能再用标准C的**

**标准输入0：**STDIN\_FILENO

**标准输出1：**STDOUT\_FILENO

**标准错误输出2：**STDERR\_FILENO

进程一启动起来，这三个内核文件描述符便会自动打开。

内核默认的一个进程可以打开的文件描述符范围：0 ~ 1023，即最多能打开1024个文件。

1. open()能打开和创建函数，原型为

#include <sys/types.h> #include <sys/stat.h> #include <fcntl.h>

**int open(const char \*pathname, int flags)**; //打开文件, **open函数是可变参数**

**int open(const char \*pathname, int flags, mode\_t mode)**; //创建文件

**int creat(const char \*pathname, mode\_t mode)**; //只能创建文件不能打开，不常用

**int close(int fd)**; //close 用于文件的关闭；fd 表示文件描述词,是先前由open 或creat 创建文件时的返回值。文件使用完毕后，应该调用close 关闭它，一旦调用close，则该进程对文件所加的锁全都被释放，并且使文件的打开引用计数减1，只有文件的打开引用计数变为0 以后，文件才会被真正的关闭。

1)其中参数： **flags 和mode 都是一组掩码的合成值(即按位或)**。 下面掩码处所显示的宏，代表整型数flag的某一位为true真。 (整型数共有32位)：Linux的权限管理都是位操作

2)flags 表示打开或创建的方式，mode 表示文件的访问权限。open通常采用**直接赋数值**的形式,如下：

int fd = open(“1.txt”,O\_WRONLY | O\_CREAT,**0755**); //表示给755 的权限

3)creat 函数等价于open(pathname,O\_CREAT|O\_TRUNC|O\_WRONLY,mode);

open()函数出错时返回-1，相关参数如下图：

4)下图：



1. 读写文件函数原型：一般是**异步，要先写到缓冲区再写到磁盘** （同步是直接写到磁盘）

**ssize\_t read(int fd, void \*buf, size\_t count)**; //参数：文件描述符、缓冲区长度

**ssize\_t write(int fd, const void \*buf, size\_t count)**;

read 、write： 1)已到达文件尾或无可读取的数据(即参数nbyte为0)：返回0

2)成功：返回读取/写入的字节数； 3)失败错误：返回-1

(5)改变文件大小： #include <unistd.h>

**int ftruncate(int fd, off\_t length);**

//函数ftruncate 会将参数fd 指定的文件大小改为参数 length指定的大小。 参数fd为已打开的文件描述符，而且必须是以写入模式打开的文件。若原来的文件大小比参数length大，则超过的部分会被删去。返回值执行成功则返回0，失败返回-1。

当文件**大小被改大时，是假象lazy模式**(只是显示的数字变大了而已)，只有变大的部分要写磁盘时，才会真正分配空间。

(6)文件偏移：#include <sys/types.h> #include <unistd.h>

**off\_t lseek(int fd, off\_t offset, int whence)**;

//将文件指针设定到相对于whence，偏移值为offset 的位置。

//成功时返回**当前位置离文件开头的字节数**:相当于同时实现了C标准的fseek、ftell。

利用该函数可以实现**文件空洞**（对一个新建的空文件，可以定位到偏移文件开头1024 个字节的地方，在写入一个字符，则相当于给该文件分配了1025 个字节的空间，形成文件空洞）通常**用于多进程间通信的时候的共享内存**。

whence 可以是下面三个常量的一个：

**SEEK\_SET** 从文件头开始计算；

**SEEK\_CUR** 从当前指针开始计算；

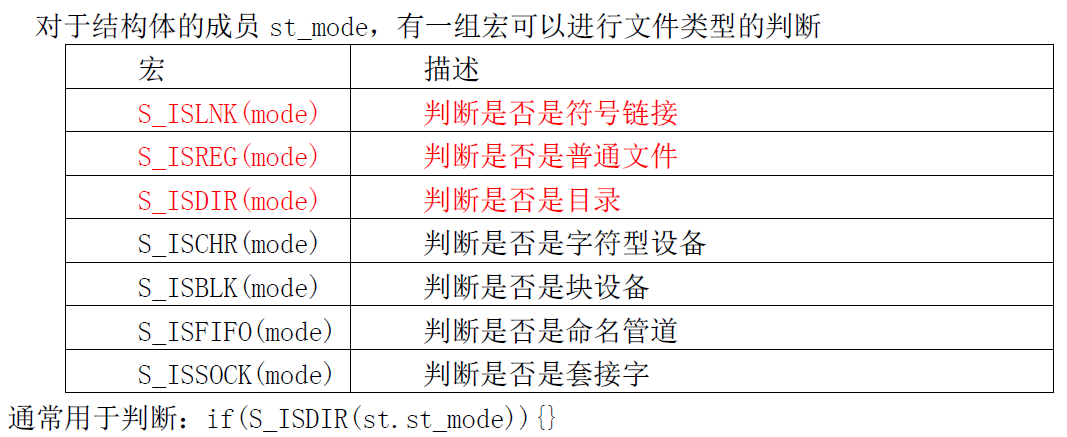
**SEEK\_END** 从文件尾开始计算

(7) 获取文件信息#include<sys/types.h> #include<sys/stat.h> #include<unistd.h>

**int stat(const char \*file\_name, struct stat \*buf); //文件名路径、stat结构体指针**

**int fstat(int fd, struct stat \*buf); //文件描述符、 stat结构体指针**

区别：stat什么文件都能操作； 而fstat不能操作目录文件。



(8) **文件描述符的复制: 用于重定向stdin 0、stdout 1、stderr 2；**

**fork时有dup机制，来继承父进程打开的文件描述符。**

**int dup(int oldfd);**

**int dup2(int oldfd, int newfd);**

1)区别复制、赋值：

**赋值：int fd2=fd;** //类似于C 语言的指针赋值，当释放掉一个得时候，另一个已经不能操作了。 在此情况下，**两个文件描述符的值相同，指向同一个打开的文件，但是内核的文件打开引用计数还是为1**，所以close(fd)或者close(fd2)都会导致文件立即关闭掉。

**复制：int fd2=dup(fd);**  //**文件描述符fd2的值为没有被使用的最小编号, 两个描述符共享同一个数据结构，且内核的文件打开的引用计数+1，变成2了**,故close(fd)后，文件不会被关闭，此时通过fd2 照样可以访问文件；只有再close(fd2)后 ，**内核的引用计数变为0，文件才会正式关闭。**

注意：两个共享**同一个结构体，只有同一个位置指针**，一个发生偏移，另一个也会偏移。

2) 重定向

printf就是封装的write()系统调用。

即：printf(“hello”); 等价于 write(1,”hello”,5); //1是标准输出

所以可通过dup()来实现重定向标准输出：即用**printf实现重定向输出** > (大于号)

如：关闭文件描述符1，再去复制文件描述符3，新的文件描述符便会为1了，和3指向同一个结构体，故标准输出就重定向到文件中了：此时printf打印便不会输出到屏幕，而是打印到描述符对应的文件中去了(即此文件充当了标准输出)。

(注意：**中间必须要打印换行刷新**)

