I/O（标准IO，文件IO）

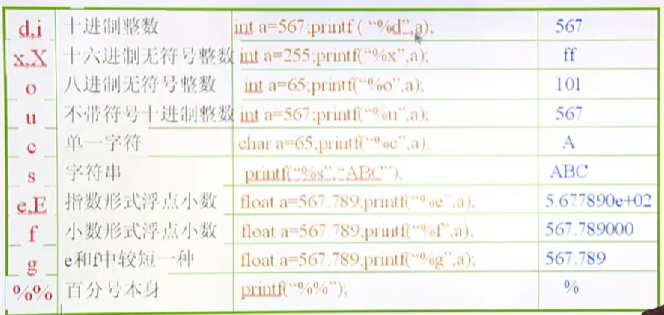
**格式化输入输出函数：scanf，printf**

printf函数原型，int printf(const char \*format, …);

format：”%[修饰符]格式字符“

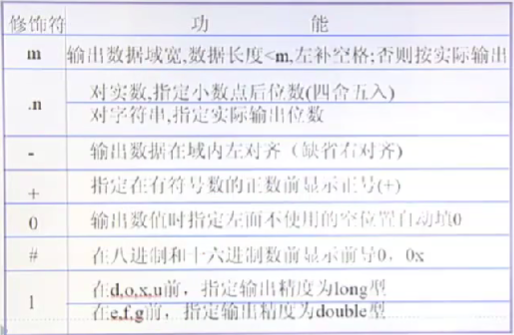
实际使用时，printf(“%[修饰符]格式字符”, 输出表项);

格式字符（感觉落下了%p，它表示打印的是地址）



char型虽然可以看出短小精悍的整型，也可当作整型来用，但一般不会用它传参，因为它有点窄，表示的数有点少，所以传参一般不用它，包括后面要讲的字符输入输出函数，getchar和putchar，全都是用整型来对待的，转换成整型数来操作

修饰符



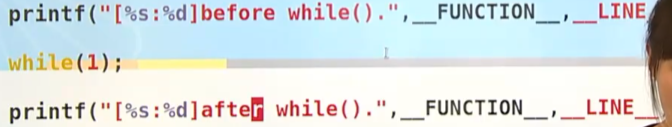
没有单位的数值在计算机的世界当中是没有意义的

面试题：定义一个宏，表示一年有多少秒

#define SEC\_YEAR (365\*24\*60\*60)，这样做，数据会溢出，所以应该加L，不放心可以加LL

#define SEC\_YEAR (365LL\*24 LL\*60 LL\*60 LL)

如果不是格式要求一定不能换行（后面会讲强制刷新缓冲区函数来解决这个问题）的话，每句printf最好顺手带上换行\n，因为用终端做输入输出时，默认的不再是全缓冲模式，而是行缓冲模式，printf遇到\n才去刷新缓冲区的，或者等缓冲区满了才自动刷新，这在调试时会出大问题，比如有个死循环，然后你搞了两句测试代码，



你没有加\n，所以before这句话就被放到了输出缓冲区，什么时候刷新呢，要么等程序结束，自动刷新IO，要么遇到一些强制刷新函数，比如flush这种，要么等你缓冲区满了，它一次性给你刷新，现在什么情况都不是，所以你的before while()这句话输不出来，你就会认为死循环出现在前面，其实不是

scanf函数原型int scanf(const char \*format, …);

format跟上面差不多，只不过一般不会用到修饰符，这会限制你的输出

实际使用时，scanf(“%格式字符”, 地址表);

scanf(“%f%f%f”, &a, &b, &c)，多个%f之间不用空，你在终端输入时，你可以用空格，用tab，用回车都可以，它们都可以视为有效的分割符，当然，%f之间加空格分开也行，不影响

极度注意，scanf函数里，用什么格式都好用，唯独用%s比较危险，因为你输入的字符串即使越界了，它也不会报错，即使已经错了

还有就是在scanf中你的%s实际上不能获得带有分割符的一个串，像串里包含空格，tab键，回车等分割符，它都不能获得，比如你通过scanf输入hello world到str中，再printf(“%s”, str);，那在终端上只会打印hello，而不会打印world，但是如果你初始化时直接使将str初始化为“hello world” ;，再printf(“%s”, str);，那在终端上就会打印hello world

scanf的返回值非常重要，关于scanf的返回值，它是返回它能正常接收的数据个数，比如里边只有%d，而且后来正确接收了一个十进制整型，那scanf的返回值就为1，否则就为0，如果里面有%d%f，后来全部正确接收则返回2，只正确接收1个就返回1，都没正确接收就返回0

敢把scanf放到循环语句中，一定要利用scanf的返回值来加上校验语句，保证scanf正确接收数据，这样的话，如果没有正确接收，则可以利用break来跳出循环，不校验的话非常容易出现把一个输入错误的项放在输入缓冲区当中，没有办法取出来，构成了一定的死循环

关于printf的返回值，是能够正常的输出的字符个数，但不包括尾0（‘\0’）

如果是scanf(“%d”, &i); scanf(“%c”, &ch); 你在终端跑起来时，先输一个整型赋给i，比如是5，然后在回车之后，输入char型给ch，比如是a，期望输出得到5和97两个输出，但实际情况是，你只能得到5和10，因为第二个scanf把回车接收进去了，回车的ascii码就是10，可以解决它，第一种方法，在后一个scanf语句中加入抑制符\*，在本例中，格式如%\*c表示你想要吃掉一个ch型的数据，以此来吃掉回车，如scanf(“%\*c%c, &ch”); 如果你不用抑制符的话，你可以在两个scanf语句中加入getchar()，也可以吃掉回车，和抑制的效果一样

**字符输入输出函数：getchar，putchar**

getchar函数原型int getchar(void);

正常返回一个int类型，返回的就是你输入的字符的ascii码值（后续如果想要以字符的形式输出就用%c，想看它的ascii码值就用%d），如果出错或者读到文件尾或者文件出错的话，返回EOF值，EOF是一个宏，宏值一般是-1，但你别拿-1跟它相对比，因为未必所有的EOF定义都是-1，一般来讲应该是-1

putchar函数原型int putchar(int c);

正常返回你写出去的字符的整型值形式，如果出错返回EOF

**字符串输入输出函数：gets(非常危险)，puts**

gets函数原型char \*gets(char \*s);

传入的是一个地址，返回值也是一个地址，它实际上是从标准输入stdin读进来一行内容，放到一块buffer中去，哪块buffer呢，由s指针所指向的那块buffer，直到碰到换行符或者EOF，并且会在尾部追加一个\0，它比较危险，因为它对缓冲区的溢出不做任何检查，这里指的缓冲区不是指进程间的buffer，而是我这里所定义的s指向的那个buffer，而且溢出不报错，只报警告，所以最好不要用gets，如果非要用，就用fgets来代替，既然有了替代品，为什么不将gets从标准库中移除呢，因为gets发布这么久了，如果要移除，那很多用到gets的代码都需要重构，牵一发而动全身，代价太大

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

第一个参数跟gets一样，第二个给了一个size值，最多能读到size-1个，因为最后要补尾0，第三个把这个流给你放灵活了，不一定从标准输入来的，可以从任意一个打开的file流中取内容过来，其实这个替代品只是治标不治本，它只能保证，你s指向的这块空间是size大小，你读进来的内容最多是size-1个，但是它不能保证你所要拿到的这个字符串，也就是说它只能保证你不出错，但未必能达到你想要达到的那个效果，要想完美解决问题，有一个方言，叫getline

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

getline可以帮助你完整地拿到一个串，多大都行，它使用的是动态内存的实现，为什么说它是方言呢，它既不是C89也不是C99，它只存在于GNU libc库当中

puts函数原型int puts(const char \*s);

是把s指针指向的这个串往外写，写到标准输出stdout上，并且以一个换行符进行终止