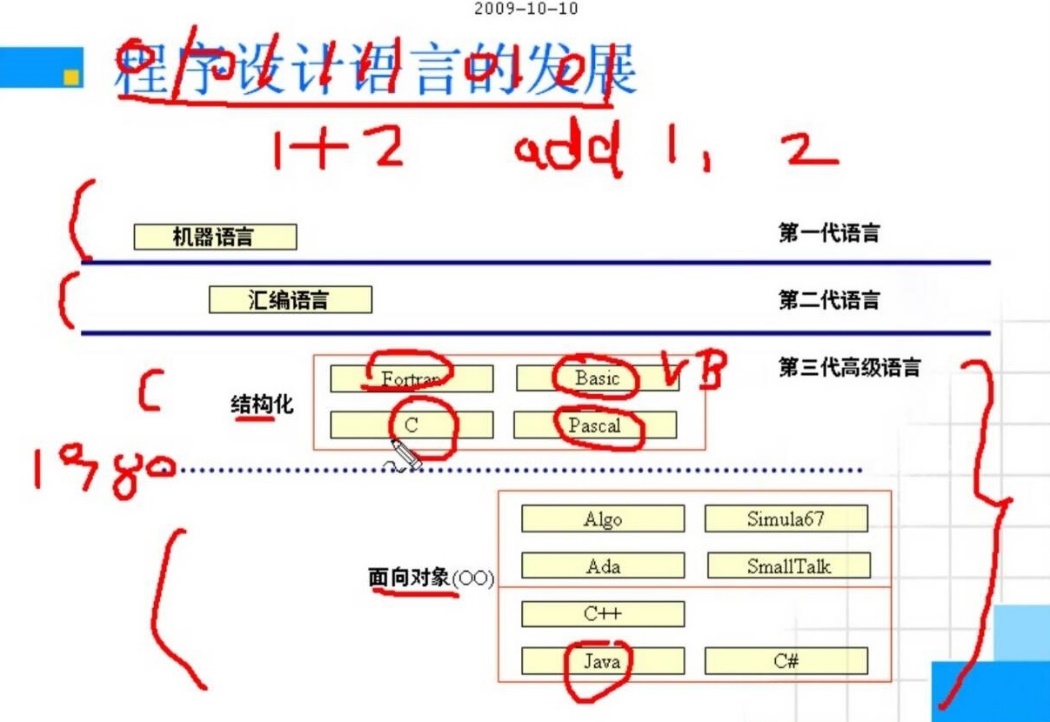
**C程序设计语言**

**C语言概述及发展史**

****

**C++脱变于c ,java (sun公司，被甲骨文公司收购)，c# (微软) 蜕变于c++，c#和java 非常类似(可二选一进行学习)  
结构化语言最重要的是c语言，面向对象语言中c++最难懂**  
**c语言特点**  
**优点：  
代码量小  
速度快（主要是因为指针可以直接访问内存）  
功能强  
缺点：  
危险性高  
可移植性不强（同样的程序放在别的平台可能结果不同）  
开发周期长**  
**c语言预备知识**  
**1.cpu，内存条，硬盘之间的关系  
cpu不能直接处理硬盘里的数据，必须要借助内存条  
2. 程序是如何运行起来的？  
程序输入完后，点编译再点链接最后点执行，编程序的软件会生成一个.exe的可执行性文件，注意这个可执行文件不能被软件直接运行,而是软件请求操作系统,然后操作系统在控制CPU来执行这个可执行性文件,而操作系统又是通过驱动程序控制电脑硬件**  
  
**c语言中的数据类型**  
**c语言中没有一个数据类型可存放字符串，必须借用字符数组，字符指针变量或字符指针数组存放，在java中可以用数据类型string 定义个变量存放  
eg: int i=0;  
变量本质是内存中一块存储空间**

**可以理解为系统让内存留出一块存储空间,把这个存储空间命名为i，即让这个存储空间与变量i 产生某种关联  
也可以将变量 i理解成一个指针，指向内存中一块地址，再向地址中存放东西,且变量必须初始化,不初始化直接输出，会输出一个垃圾值（随机值）**

**实数在C语言中又称浮点数**

**浮点数在C语言中有两种表达形式**

**一：小数形式：由小数点和数字组成 eg：0.2 .256 12. ………**

**二：指数形式: eg: 0.23e2 0.23E001 0.23E00 0.1e+6 .e0 … 注意:e前面后面必须有数,且后面的数必须为整数，e大小写无所谓 , 注: 0.23e2 ==0.23\*10^2**

**单精度浮点型默认保留小数点后6位**

**双精度浮点型默认保留小数点后15位**

**可以看出双精度浮点型比单精度浮点型精度更高**

**C语言默认实数(浮点数)为双精度浮点型**

**Eg: float a=0.2356;**

**编译后不会报错,但会出现警告,运行后的结果会丢失精度**

**在0.2356后加f或F，警告就会消失**

**字符及字符串**

**单个字符用单引号括起来**

**‘A’ 表示单个字符A**

**‘AB’ 错误,并且单引号里只能有一个字符**

**字符串用双引号括起来**

**“abc” 正确**

**‘abc’ 错误**

**“A” 正确, 因为“A” 代表 ‘A’ 和 ‘\0’的组合(字符串结束标志)**

**在c语言中定义一个字符串，系统默认会在字符串后加一个字符串结束标志 ‘\0’**

**并且字符串是由>=0个字符组成**

**注: 转义字符属于单个字符不属于字符串,也是用单引号引起来**

**转义字符: 将 \ 后的字符转换成另外的意义**

**Eg:**

**\o 或 \oo 或 \ooo 其中o表示一个八进制数字,最多3位**

**‘\101’ 代表八进制数101对应的ASCII字符，即 ’A’(八进制数101就是十进制数65，字符A对应的ASCII码是65)**

**换句话说就是将八进制数101转化成它对应的ASCII字符**

**‘\012’ 代表八进制数12对应的ASCII字符，即 换行符**

**\xh [h...] 其中h代表一个十六进制数字**

**‘\x41’ 代表十六进制数41对应的ASCII字符，即 ‘A’ (十六进制数41就是十进制数65，字符A对应的ASCII码是65)**

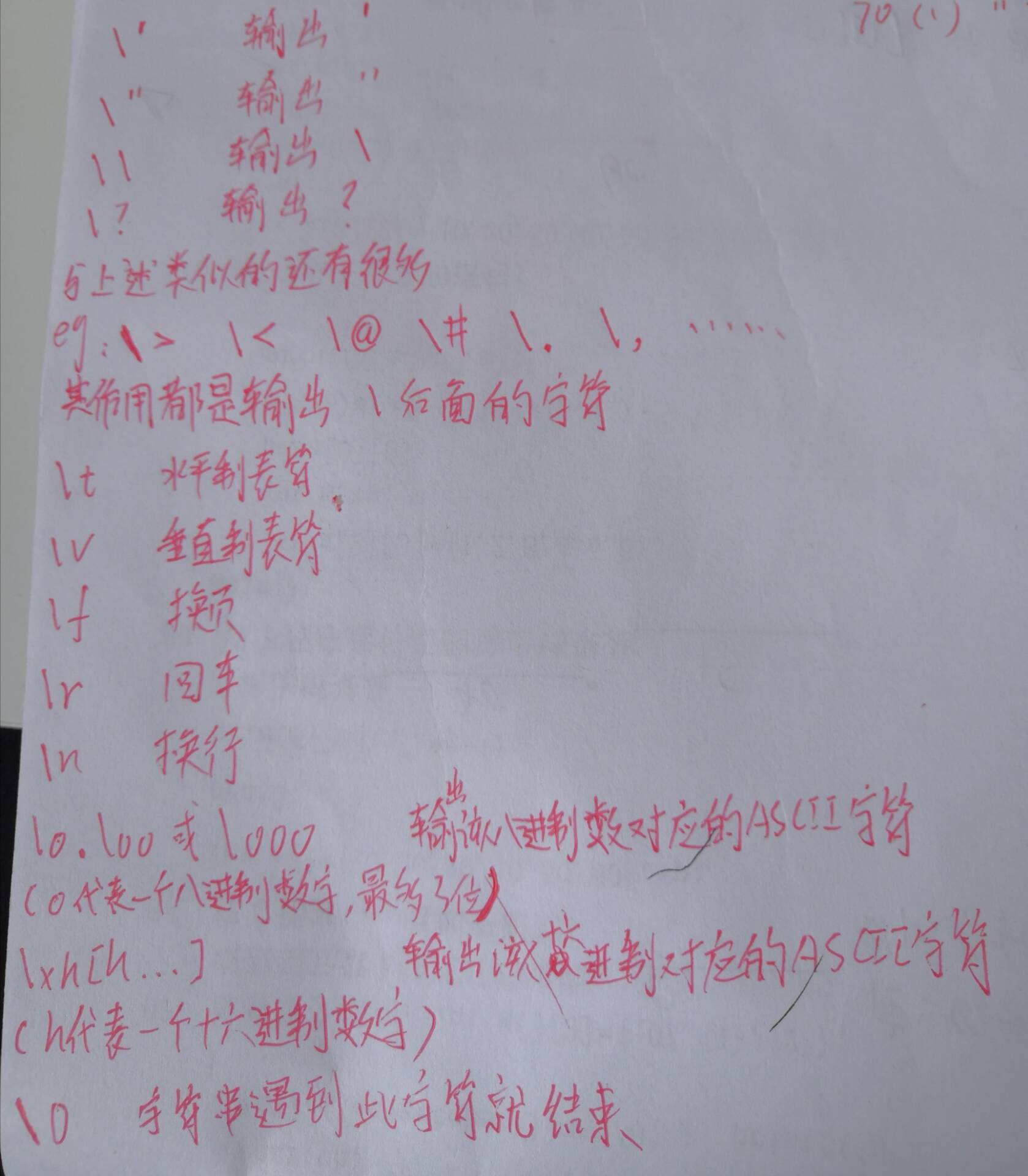
**换句话说就是将十六进制数41转化成它对应的ASCII字符**

**‘\101’ 和 ‘\x41’ 都代表ASCII码为65的字符，即字符 A**

**‘\033’ 和 ‘\x1B’ 都代表ASCII码为27的字符，即ESC控制符**

**‘\0’ 或 ‘\000’ 是代表ASCII码为0的字符，即空操作字符，常用于字符串**

**常见的转义字符**

****

**拓展:**

**float 和double 都不能保证可以精确的存储每一个小数（浮点型数据）**

**eg:**

**double i = 0.2355696;**

**printf(“%lf”, i); // 输出结果为0.235570**

**总结:存储某些小数时,用float和double 不能精确存储它，所以存储一个与它非常非常相近的数,这个非常相近的数可能比它大一点，也可能小一点，因此循环语句中值会随时更新的变量不能定义成浮点型（准确的说这个更新的变量一定也要是能够控制循环结束的变量）**

**强制类型转换**

**格式:**

**(数据类型)(表达式)**

**功能：**

**把表达式的值强制转换为前面所执行的数据类型**

**eg:**

**(int)(4.5+2.2) 结果为：6**

**(float)(5) 结果为：5.000000**

**一般情况下**

**short 占2个字节**

**int 4**

**long int 4**

**long long int 8 (C99新增类型,有的C编译器尚未支持)**

**char 1 (在C语言中占1个字节，在java中占2字节)**

**float 4**

**double 8**

**注:**

**C标准没有具体规定各种类型数据所占用的存储单元长度,这是由各编译系统决定的**

**C标准只要求long型数据长度不短于int型,short型不长于int型**

**即: sizeof(short)<=sizeof(int)<=sizeof(long)<=sizeof(long long)**

**c语言中的进制**

**在c语言中十进制正常表示**

**八进制数前面要加数字 “0”，eg：016 注意:八进制数中不能有8，每位数的范围是0-7**

**十六进制数前面要加 “0x”或者“0X”，eg：0x88 注意:十六进制数中不能有16（G）**

**进制转换 每位数的范围是0-F**

**二进制，八进制，十六进制转化为十进制只需按权位系数和展开就行**

**eg：二进制数1001 ->十进制**

**1001=1\*2^0+0\*2^1+0\*2^2+1\*2^3=9**

**八进制数16->十进制**

**16=6\*8^0+1\*8^1=14**

**十六进制数 7A ->十进制**

**7A=10\*16^0+7\*16^1=122**

**十进制转化为二进制，八进制，十六进制采用除留余数法**

**十进制转化为八进制，十六进制，还可以先将其转换成二进制数,再将二进制数转化成对应的八进制，十六进制**

**拓展:**

**1011B 为二进制数1011，也记为 (1011) 2**

**1357O 为八进制数1357，也记为 (1357) 8**

**2049D 为十进制数2049，也记为 (2049) 10**

**3FB9H 为十六进制数3FB9，也记为 (3FB9) 16**



**数据在内存中的存储形式**

**整数: 以补码的形式存放**

**实数（浮点数）:以指数形式存储**

**字符：以整数形式(该字符对应的ASCII码)存放**

**拓展:ASCII码是整数,本质上字符在内存的存储方式与整数一样**

**ASCII码规定了不同字符使用哪一个整数值表示**

**什么是字节**

**字节就是存储数据的单位,并且是硬件所能访问的最小单位**

**1字节(byte)=8位(bit)**

**1K=1024字节**

**1M=1024K;**

**1G=1024M**

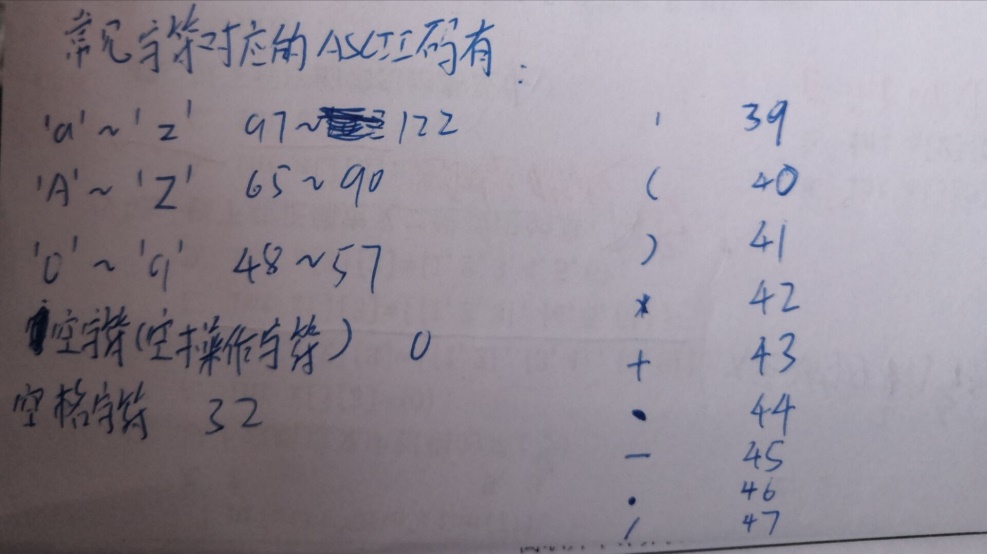
**什么是ASCII码**   
**ASCII码不是一个值，而是一种规定（规范）  
它规定了不同字符使用哪一个整数值表示**

**英文：American Standard Code for Information 美国信息交换标准码 发音:/ ‘aski /**

**Eg: ‘A’ --- 65 理解: 字符A在ASCII编码下以整数65表示**

**书本上理解是:字符A的ASCII码是65,这样理解不准确**

**‘a’---97 理解同上**



**Printf语句的用法**

**用法: printf (“输出控制符”, 输出参数);**

**功能: 规定要输出的数据以什么格式输出**

**eg:**

**Int i =100，j =52；**

**printf(“%d”, i, j); //error 只会输出100, 不输出52 注意:输出控制符和输出参数的个数必须一一对应**

**printf(“%d %d”, i, j); //正确 输出100 和52 将i，j以十进制格式输出**

**printf(“%x %x”, i, j); //正确 输出 64和 34 将i，j以十六进制格式输出**

**printf(“%#x %#x”, i, j); //正确 输出 0x64和 0x34 将i，j以十六进制格式输出**

**注意: 输出控制符规定要输出的参数以什么格式输出**

**拓展: %d or %i 输出有符号的十进制整数**

**即: 以带符号的十进制形式输出整数(正整数不输出符号)**

**%ld 输出有符号的长整型十进制整数**

**%u 输出无符号的十进制整数，即只能输出十进制的正整数**

**%x or %X 输出无符号的整型/短整型/长整形十六进制整数**

**(这里的输出无符号是指不输出前导符0x)**

**即：以十六进制无符号形式输出整数(不输出前导符0x)**

**%o 输出无符号的整型/短整型/长整形八进制整数**

**(这里的输出无符号是指不输出前导符0)**

**%c 输出单个字符**

**%s 输出字符串**

**%f 以小数形式输出单/双精度浮点数(实数)**

**%lf 以小数形式输出双精度浮点数(实数)**

**%e or %E 以指数形式输出单/双精度浮点数(实数)**

**%g or %G 用来输出单/双精度浮点数(实数),系统自动选f或e格式输出,选择其中长度较短的格式,不输出无意义的0**

**注:**

**除了X, E, G外，其他格式符必须为小写**

**Scanf语句的用法**

**用法: scanf (“输入控制符”, 输入参数);**

**功能: 将从键盘输入的字符转化输入控制符规定格式的数据，然后存入以输入参数的值为地址的变量中（简单来说，就是通过键盘将数据输入到变量中）**

**eg:**

**Int i;**

**scanf(“%d”,&i );**

**printf(“%d”, i );**

**关键问题:**

**为什么用scanf语句从键盘向变量中输入一个数据,需要加输入控制符?**

**答:这里我假设从键盘向变量 i 中输入数据 “123” 通过上面的程序会输出十进制整数123，输入时加输入控制符”%d”，是为了规定数据 “123” 以十进制整数的形式存放到变量 i 中 ,这里输入的数据 “123”,并非是我们认为的整型数字123，而是字符1 2 3,因为操作系统接收从键盘输入的数据时,是把它当做字符接收,所以当把它存入变量中时,需要加输入控制符,规定它以什么形式输入,并且规定的形式必须要与存入的变量的数据类型一致**

**拓展:**

**(1)输入双精度浮点型数据时，输入控制符必须是 %lf**

**(2)也可以通过scanf语句向多个变量输入数据**

**eg:**

**int i, j, k;**

**scanf(“%d,%d,%d”,&i, &j, &k );**

**printf(“%d”, i , j, k);**

**输入 1 2 3 错误**

**输入 1 ,2 ,3 错误**

**输入 1, 2, 3 正确**

**输入 1,2,3 正确 //注意:输入数据时必须严格按照输入的格式输入(输入一个数后,紧随而之必须输入一个逗号分隔,再输入另一个数)**

**C语言中的运算符**

**运算符的优先级:**

**逻辑非,自加自减运算符**

**! ++ --**

**算术运算符**

**\* / % + -**

**关系运算符**

**> < >= <= == !=**

**逻辑运算符**

**&& ||**

**条件运算符(又叫三目(元)运算符)**

**？ ：**

**赋值运算符**

**= += -= \*= /= %= (后面6个又叫复合赋值运算符)**

**逗号运算符**

**,**

**拓展:**

1. **逗号表达式：**

**格式: (A，B，C，D)**

**功能: 从左到右执行，最终表达式的值是最后一项的值**

**eg:**

**int i = (3，5，7，8)；**

**int j = 2；**

**printf（“%d\n”， i）； //输出的结果为 8**

1. **(2) (3) (4)**

**i = (j++，++j，j+2，j-3)；**

**printf（“%d\n”， i）； //输出结果为 1**

**注意:**

**先执行（1）j的**

**值为3，再执行（2）j的值为4，当执行到（3）时 j的值并没有改变，仍是4因为j+2不等于 j=j+2,所以j的值并没有被更新，最后执行（4）时，将j-3，即4-3赋予i，输出i = 1；**

1. **赋值运算结合性为由右向左结合，赋值运算符左值为变量，右值为变量或常量**

**eg:**

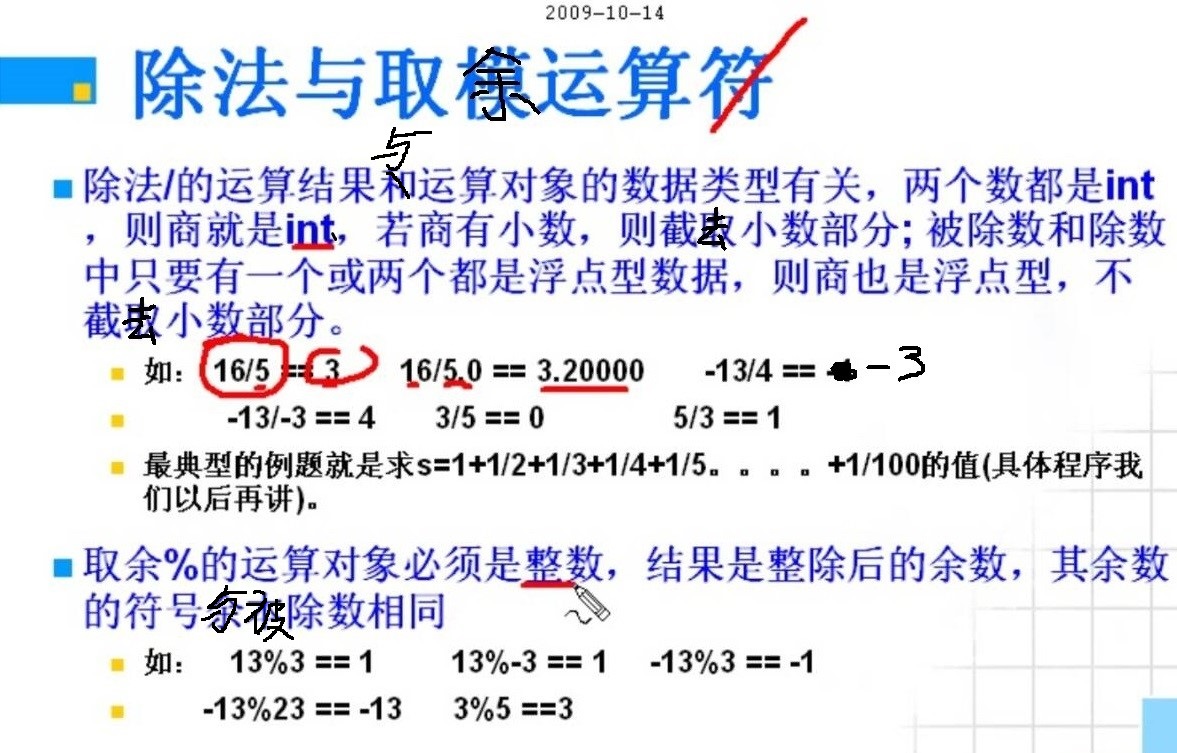
a=（b=3）=1；//这是一个错误的赋值语句 (1)

a=（b==3）=1；//这是一个错误的赋值语句 (2)

**因为(b=3)是一个赋值表达式，不是一个变量，不能将1赋予它**

**同理（b==3）是一个逻辑表达式，不是一个变量，不能将1赋予它**

**算术运算符中需要注意的：**

****

**注意：关系表达式和逻辑表达式关于值的问题？  
   关系表达式返回的是逻辑值的真假，即返回真或返回假  
   逻辑表达式返回的是逻辑量的真假，若逻辑量为非0即为真，为0即为假  
    在c语言中真用1表示，假用0表示  
  
表达式和语句的区分  
没有以 ；结尾的语句都可以理解为表达式  
以 ；结尾的是语句**

**拓展：(C语言中的位运算符)**

C**语言中，位运算的对象只能是整型或字符型数据，不能是其他类型的数据。**

C语言提供的六种位运算符，如下表所示，

|  |  |
| --- | --- |
| 运算符 | 含义 |
| ~ | 按位求反 (单目运算符) |
| << | 左移 (单目运算符) |
| >> | 右移 (单目运算符) |
| & | 按位与 (双目目运算符) |
| ^ | 按位异或 (双目目运算符) |
| | | 按位或 (双目目运算符) |

**优先级: ~ > (<<, >>) > & > ^ > |**

**先说一下布尔值的与、或、异或、反运算：  
与运算，两者都为真计算结果为真，反之为假：  
1 & 1 = 1，1 & 0 = 0，0 & 1 = 0，0 & 0 = 0；  
或运算，至少其一为真计算结果为真，反之为假：  
1 | 1 = 1，1 | 0 = 1，0 | 1 = 1，0 | 0 = 0；  
异或运算，两者不同为真，两者相同为假：  
1 ^ 1 = 0，1 ^ 0 = 1，0 ^ 1 = 1，0 ^ 0 = 0；  
取反运算，单目运算符：  
~1 = 0，~0 = 1；  
以上例子 1 代表真，0 代表假。**

**按位运算就是将二进制位的每一位进行如上计算。**

**(1)按位或运算  
如 2 | 5，转换为二进制（以 1 字节 8 位为例）：  
00000010 | 00000101，每个二进制位分别进行或运算后得：  
00000111 即 7。**

**所以：2 | 5 = 7**

**注意：7 = 5 + 2 但不代表 2 | 5 = 2 + 5。  
如：2 | 6 = 6，5 | 6 = 7，2 | 4 | 5 | 6 | 7 = 7。**

**(2)按位与运算**

**如：2 & 5 的结果为 0：  
00000010 & 00000101 = 00000000  
按位与运算 2 & 6 的结果为 2（二进制 00000010）：  
00000010 & 00000110 = 00000010。**

**(3)按位异或运算**

**2 ^ 6 的结果为 4（二进制 00000101）：  
00000010 ^ 00000110 = 00000100。**

**(4)按位取反运算**

**如： ~00000110 = 11111001**

**流程控制(重点)**

**流程控制结构包括3种基本结构：**

**顺序结构，选择结构，循环结构，每种结构又有相关的语句**  
**重要性：  
        一：c语言中的流程控制原封不动的被照搬到c++,java, c# 中。  
        二：循环结构、选择结构、顺序结构，它们组成的算法可以解决任何复杂的问 题**

**拓展：自学编程最关键的是能看懂程序  
            如何看懂程序  
           一：先看懂流程  
    （程序执行的顺序，先执行哪个语句，再执行哪个语句）  
           二：再看懂每个语句的功能  
           三：最后在试数  
  
什么是流程控制？  
程序代码执行的顺序  
  
流程控制的分类  
顺序  
程序从上往下执行**

**选择（分支）  
 定义：有选择的执行某些语句，即某些语句可能执行，也可能不执行  
 分类：**

**if (双分支选择语句)**

1. **if 语句的作用范围:** **if语句默认只能控制一个语句,且该语句必须是if语句的下一个语句若要控制多个语句，必须加 { } , while for 循环语句作用范围和if 一样**
2. **if ...else if …else语句的用法:**

**else语句必须匹配对应的if语句不能单独用,**

**因为else语句是if语句的子句，是if语句的一 部分**

**if语句可以匹配else语句,也可以不匹配单独使用**

1. **if 语句常见的错误:**

**if (3>2)；**

**等价于**

**if (3>2)**

**；//这是一个空语句,理论上没错但输出为空**

**switch（多分支选择语句）**

**格式：**

**switch（表达式）**

**{**

**case 常量1：语句A；break； (1)**

**case 常量2：语句B；break； (2)**

**…………**

**default：语句C；(3)**

**}**

**重点需要注意的几个地方**

**(1) switch括号中的表达式的值应为整型或字符型**

**(2) case后跟一个常量或者常量表达式**

**(3) case 和 default都是标号,相当于入口，起标记作用,不起条件判断作用**

**eg: 当表达式的值等于常量1，执行语句A**

**当表达式的值等于常量2，执行语句B**

**当表达式的值既不等于常量2，也不等于常量3，执行语句D**

**(4) case 子句后必须加break语句,用来跳出switch语句**

**若（1）（2）中没有break，结果会依次将语句A，B，C输出**

**(5)default后可以不加break语句**

**因为switch语句执行到default就已结束,流程会自动跳到switch语句的下一个语句执行，所以加不加无所谓**

**(6)** **各个case标号出现次序 不影响执行结果**

**eg: 可以先出现default ,再出现 case 1 , 再出现 case 3...**

**(7) 多个case标号可以共用一组执行语句**

**eg: case ‘A’:**

**case ‘B’:**

**case ‘C’: printf (“A”);break;**

**循环**

**定义: 某些语句可能会重复执行**

**分类:**

**for**

1. **单个for 循环的使用**

**格式: for(循环变量赋初值；循环条件；循环变量增值)**

**//循环变量是用来控制循环结束的变量**

**eg:**

**for(表达式1；表达式2；表达式3)**

**语句4；**

**注: 执行顺序 1 2 4 3 ，表达式3执行完代表一次循环结束**

1. **多个for循环的嵌套使用**

**eg:  for (表达式1；表达式2；表达式3)  
               for(表达式4；表达式5；表达式6)  
                           语句7；  
                                                语句8；  
  
                            执行顺序：**

**1 2 4 5，若5成立执行7 6，不成立时说明内部的for循环结束，然后跳出该for循环执行3，然后执行2，若2成立，继续执行内部for 循环里的语句，不成立执行8，说明内外层for循环都已执行完毕**

**while**

**用while语句解决的问题，for都能解决，for语句比while语句功能更强，用法更灵活，更简练**

**do while**

**do while 较while，for的区别**

**while for 都是先判断循环条件是否成立再执行循环体**

**do while 与之相反**

**while for 里的循环体可以一次都不执行，但do while至少要执行一次**

**拓展:**

**为什说for语句功能更强**

**eg:**

**for (表达式1；表达式2；表达式3)**

**{ 循环体 }**

1. **表达式1语句也可以写在for语句外面**
2. **表达式3也可以写在循环体里**
3. **表达式1 2 3都可以省略,但省略1或者3都有可能导致无限循环，使这个for循环语句失去意义**
4. **甚至循环体内的语句都可以放在达式3的位置**
5. **表达式1 3 可以是逗号表达式，即包含多个语句**
6. **………**

**break 和 continue**

**break**

**Break如果用于循环，其作用是终止循环**

**Break如果用于switch语句，其作用是终止switch语句**

**Break不能直接用于if语句中，除非if语句是循环内部的一个子句**

**在多层循环中，break只能终止距离它最近的那层循环(最里面包裹着它的那层循环)**

**在多层switch嵌套中，break只能终止距离它最近的switch语句**

**(最里面包裹着它的switch语句)**

**continue 用来结束本次循环，开始执行下次循环**

**数组**

**一维数组**

**int a[4]; //a是数组名，4是数组中元素的个数，也是数组的长度，4个元素（4个变量）分别是a[0] a[1] a[2] a[3] ,没有a[5]这个元素，int 定义了该数组是整型数组,即数组中的4个元素都为整型，一维数组就就好比坐标系中只有横坐标，就像一条直线**

**二维数组**

**int a[3][4]; //该二维数组中有12个元素，二维数组实际上就是个矩阵（3行4列的矩阵）,就好比坐标系中只有横纵坐标，就像一个平面**

**注:二维数组实际上是数组的数组**

**二维数组a包含3个行元素，每个行元素又是包含4个元素的一维数组, 分别是 a[0], a[1],a[2] ,这里 a[0], a[1],a[2] ,是一维数组名**

**多维数组**

**int a[2][3][4]； //定义了三位数组a，它有2页3行4列**

**字符数组**

**char a[N]；//a是一个字符型数组，可以用来存放字符串，数组名a存放了首字符的地址**

**eg：char a[ ] = “abcde”；**

**注：**

**1. 可以通过数组名加下标引用字符串中的一个字符，并输出**

**a[0] = ‘a’ a[1] = ‘b’ a[2] = ‘c’ ……**

**printf(“%c%c%c”，a[0]，a[1]，a[2])；**

**2. 还可以通过数组名输出该字符串**

**printf(“%s“，a)；**

**3. 字符数组a的长度为6，不是5，因为字符串存放在内存中，系统会自动在字符串末尾加一个字符串结束标志 ( ‘\0’ )**

**拓展:**

1. **c语言中规定数组从下标0开始，数组元素以行序为主序优先存放(按行优先存放) ，数组在c语言中是种特殊的数据类型，把它当作指针类型来处理，在数据结构中是一种存储结构**

**函数（重点）**

**什么叫函数**

**能够完成特定功能的代码块**

**也可以理解为函数是一个工具，它是为了解决大量类似问题而设计的**

**为什么需要函数(也是函数的优点)**

**(1) 避免了重复性操作，提高代码利用率**

**(2) 有利于程序的模块化,每个函数用来完成一个特定的功能**

**函数的分类:**

**从用户角度分两种**

1. **库函数:系统提供的函数，用户不必定义和声明，直接调用，**

**因为库函数及其声明已经包含在其对应的头文件中，使用库函数时必须加上其对应头文件**

**eg：<stdio.h> 里包含了printf 和scanf 函数及其声明，**

**所以可以直接使用，无需再定义和声明**

1. **用户自己定义的函数，用来完成指定的某个功能**

**从函数的形式上分两种**

1. **无参函数：用来完成指定的一组操作，当执行完毕时可以带回或不带回函数值(函数返回值)**
2. **有参函数：主调函数通过参数向被调函数传递数据，当被调函数执行完毕，一般会返回一个函数值(函数返回值)，供主调函数使用**

**注意：**

1. **C程序是从main函数开始，到main函数结束，中途可能会调用其他函数，main函数是由系统调用的**
2. **C程序的基本单位就是函数**
3. **所有函数都是平行的，不存在谁属于谁，定义函数时也是分别定义，相互独立，即函数不能嵌套定义但能嵌套调用**
4. **return 语句用来终止函数,同时返回一个函数值，也可以不反回**
5. **函数返回值的数据类型需与函数的数据类型一致，如果不一致，则返回值以函数的数据类型为准**
6. **函数体内可以有多个return语句， 但函数最终只有一个返回值**
7. **void类型的函数不可以有带返回值的return语句，但可以有不带返回值的return语句**
8. **无论函数是否带有return语句，当函数执行完后都会返回到调用处。**
9. **定义函数时没有声明其类型，默认为int类型**

**函数声明**

**作用：告诉编译器，自己定义了一个函数**

**注意：函数声明可以放在主调函数内部的开头部分**

**也可以放在函数外部，但必须要所有函数的前面**

**函数调用的过程**

1. **主函数中执行到被调函数时，为被调函数临时分配内存单元（主要是为被调函数内部定义的一些变量分配内存单元，其中就包括形参）**
2. **在主调函数中将实参的值传递给被调函数的形参(在主调函数中通过实参向形参传递数据)**
3. **开始执行被调函数 (即开始执行被调函数内部的语句)**
4. **用return语句将被调函数中的函数值返回给主调函数使用，如果不需要返回值，就不需要用return语句，该步可不执行**
5. **函数调用结束，释放被调函数的内存单元（主要是释放被调函数内部定义的一些变量的内存单元，其中就包括形参）**

**注：**

**若f函数调用g函数，当g函数调用结束，返回到f函数中调用g函数的位置，若该位置后还有语句未执行，就继续执行后面的语句**

**实参和形参**

**个数必须相同，位置必须一一对应，数据类型必须互相兼容**

**拓展**

**<stdio.h> 是什么?**

**stdio 是 standard input/output 标准输入输出 的缩写**

**.h 是 .header file 头文件 的缩写**

**这个头文件里包含了printf 和 scanf 库函数及其声明**

**简单来说stdio．h文件中包含标准输入输出函数的函数说明，通过引用此文件以便能正确使用printf、scanf等函数**

**还有个与其功能类似的头文件叫 <iostream.h>**

**Iostream 输入输出流 其里面包含了 cin (输入) cout (输出)**

**这两个函数**

**cin cout 和 scanf printf 的区别**

1. **使用的格式不同**
2. **使用cin cout 输入/输出参数时，无需加输入/输出控制符**
3. **在输入输出流中 endl 表示换行**

**注:iostream 是c++中用来输入输出的, c99标准时，才将其引入c语言**

**局部变量和全局变量的作用域，生存期及存储类别**

**预备知识:**

**(1)内存中给用户使用的存储空间可以分为3个部分**

**程序区，静态存储区，动态存储区**

**系统给变量分配的存储空间是在内存中的静态存储区，则该变量为静态存储方式**

**系统给变量分配的存储空间是在内存中的动态存储区，则该变量为动态存储方式**

**(2)在c语言中，变量和函数有两个属性：数据类型和数据的存储类别**

**定义一个变量必须指定其数据类型，定义函数时，没有指定其数据类型，默认int型，但存储类别可以指定也可以不指定，因为c语言规定了如果在定义时不指定变量和函数的存储类别，会默认使用一个特定的存储类别**

**局部变量**

**什么是局部变量?**

**函数内部定义的变量或者函数内部的复合语句中定义的变量。**

**eg:**

**for (int i=0；i<5；i++)**

**循环变量i就是在函数内部的复合语句中定义的变量，它的作用域只限于循环内,即变量i只能在for循环内部使用,在for循环外部不能使用**

**局部变量的作用域**

**只能在函数内部使用**

**局部变量有哪些**

1. **自动变量，又叫动态局部变量（以关键字auto声明）**

**Eg: auto int a；**

**注:**

1. **定义时关键字auto可以省略,即 auto int a；等价于 int a;**
2. **在函数内部定义变量时如果不指定其存储类别,默认为自动变量**
3. **自动变量属于动态存储方式，即在内存中的动态存储区分配存储空间，函数调用结束后其存储空间被释放**
4. **定义自动变量时不赋初值的话，其里面是个垃圾值（随机值）**
5. **静态局部变量（以statis关键字声明）**

**Eg: static int a；**

**注:**

1. **定义静态局部变量时 static不可省**
2. **静态局部变量属于静态存储方式，即在内存中的静态存储区分配存储空间，静态局部变量的值在函数调用结束后不消失继续保留原值，即其所占的存储空间在函数调用结束后不会被释放，它所占的存储空间在整个程序彻底结束后才会释放**
3. **定义静态局部变量/全局变量时不赋初值的话，编译时自动赋初值为0(对数值型变量)，或’\0’(对字符变量)**

1. **寄存器变量（以关键字register声明）**

**注：**

1. **cpu中有个部件叫寄存器，寄存器中有块空间可以用来临时存储数据，如果定义了一个变量，系统给它分配的存储空间在cpu的寄存器中，这个变量就是寄存器变量**

**eg：register int a；**

**变量a是寄存器变量，其存储空间不在内存中的静态或动态存储区，而是在cpu中的寄存器中**

1. **寄存器的存取速度比内存快**
2. **形式参数**

**形式参数也是局部变量，存储在内存中动态存储区**

**全局变量 (又叫外部变量，全程变量)**

**什么是全局变量?**

**函数外部定义的变量。**

**全局变量的作用域**

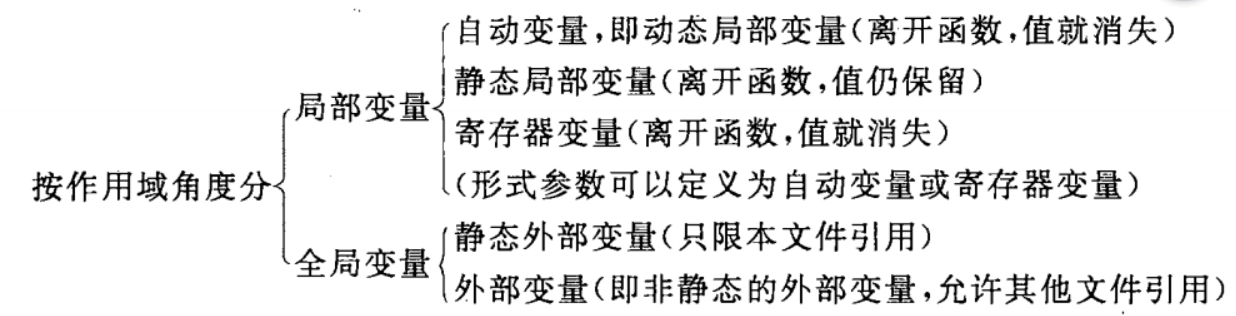
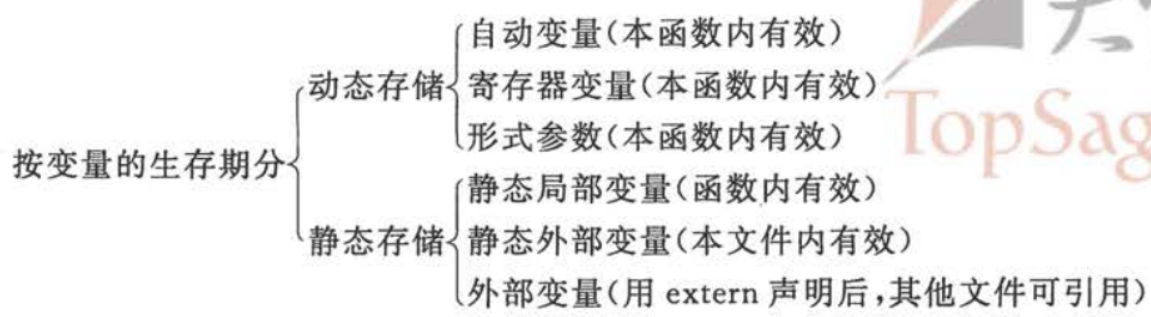
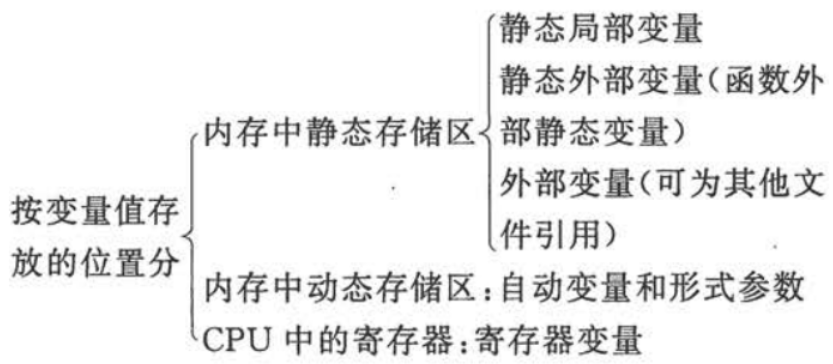
**从定义处到本源文件的末尾，即从定义处到整个程序结束**

**注：**

1. **全局变量都是存储在静态存储区，程序开始编译时为其分配存储空间，当整个程序结束时其所占存储空间才会被释放**
2. **使用extern可以扩大外部变量在一个文件中的作用域，也可以将外部变量的作用域从本文件扩展到其他文件。具体如何使用查书208 209 页**
3. **使用static可以将外部变量的作用域限制在本文件中，当确定其他文件不需要引用本文件中的外部变量时就可用static，将其变为静态外部变量，防止被误用。具体用法查书211页**

**归纳**

**从不同角度区分变量**

****  

**拓展**

**动态存储区的变量，临时分配的存储空间，在函数调用结束时其存储空间就会释放。**

**静态存储区的变量，在程序开始编译时(对全局变量而言)或函数调用开始时(对静态局部变量而言)，为其分配存储空间，直至整个程序结束时才会被释放**

**当一个函数内部定义的局部变量的名字和全局变量一样时，在该局部变量的作用范围内会屏蔽全局变量，即全局变量不起作用**

**用static声明局部变量和外部变量的区别**

1. **对局部变量而言，用static声明，是将它变为静态局部变量，在静态存储区为其分配存储空间，涉及的时生存期的问题**
2. **对外部变量而言，用static声明，是将它的作用域限制在本文件中，涉及的是作用域的问题**

**内部函数和外部函数**

**内部函数**

**定义: 只能被本文件中的其他函数调用**

**格式：static 函数的类型 函数名（形参表）**

**eg:**

**static int fun(int a，int b)**

**//定义内部函数时必须再函数的数据类型前加static关键字**

**外部函数**

**定义：不仅被本文件中的其它函数调用，还可以被其它文件中的函数调用**

**格式：extern 函数的类型 函数名（形参表）**

**eg:**

**extern int fun(int a，int b) 等价于 int fun(int a，int b)**

**//关键字extern可省略**

**注：其它文件要调用本文件中的函数，在其它文件中需要对此函数声明，声时需要加关键字extern**

**eg：**

**extern void enter\_string(char str[])；**

**//若在定义函数时没指定函数是内部还是外部函数，默认为外部函数**

**结构体**

**什么是结构体?**

1. **结构体是用户自己定义的数据类型，也叫结构体类型**
2. **它是复合型数据类型，因为它里面的成员可以是其他基本数据类型的变量，也可以是另一个结构体类型的变量**

**换句话说：结构体是把一些基本数据类型组合在一起形成的新的复合型数据类型**

**如何定义结构体类型**

**格式：struct 结构体名**

**{**

**成员列表；**

**}；**

**eg：**

**struct student**

**{**

**int age；**

**char sex；**

**………**

**}；**

**注：**

1. **定义结构体类型必须加关键字struct**
2. **student是结构体名，struct student 是结构体类型名**
3. **age，sex是结构体中的成员，前面也说了成员可以是其他基本数据类型的数据，也可以是另一个结构体类型的数据**
4. **也可以定义没有结构体名的结构体类型**

**eg：**

**struct**

**{**

**int age；**

**char sex;**

**}；**

**//该结构体类型没有结构体名，这样的用法少用**

**如何定义结构体类型的变量**

1. **定义结构体类型的同时也定义该类型的变量**

**Eg：**

**struct student**

**{**

**int age；**

**char sex；**

**} student1，student2；**

**// student1，student2是结构体类型的变量**

1. **先定义结构体类型，再单独定义该类型的变量**

**eg：**

**struct student**

**{**

**int age；**

**char sex；**

**}；**

**struct student student1，student2；**

**如何给结构体变量赋值**

1. **定义结构体变量的同时给整体赋值**

**eg:**

**struct student student1 = {13，‘男’}；//类似于数组**

1. **定义结构体变量后再给变量赋值**

**eg：**

**struct student student1；**

**student.age = 13；**

**student.sex = ’男’；**

**// “ . ” 是成员运算符，优先级在运算符中最高**

**如何引用结构体变量中成员**

1. **结构体变量名.成员名**

**eg：**

**struct student stu；**

**stu.age = 13；**

1. **指针变量名->成员名**

**eg：**

**struct student student1；**

**struct student \* st = &student；**

**st ->age = 13；**

**注：**

1. **st是该结构体类型的指针变量可以存放该类型数据的地址**
2. **“ -> ”是指向运算符**
3. **st ->age 等价于 (\*st) .age 等价于 stu.age，即3者相互等价**

**解释:**

**因为st 指向了 stu，即\*st就是stu 所以(\*st) ->age 等价于 stu.age 不难理解**

**为什么st ->age 也会等价于 stu.age？**

**因为 st->age在计算机内部会转化成(\*st).age，这是 ->的含义，也是一种硬性规定，所以st ->age 也等价于 stu.age**

**总结：**

**我们之所以会知道st ->age 等价于 stu.age，是因为st ->age被计算机转化成(\*st) .age来执行了，这里要区分与其他类型指针的区别**

**注：(\*st) .age中的 () 不能省略，因为” . ”优先级高于” \* ” ，去掉（）就变成\*(st.age)，错误**

1. **st->age的含义：**

**st所指向的结构体变量中的age这个成员**

**结构体数组**

**结构体数组与其他数组不同的是，结构体数组中的每个元素都是结构体变量，并且每个元素都又包含自己的成员**

**eg:**

**struct student a[3]；**

**//定义了一个结构体数组a，数组中有3个元素 a[0] a[1] a[2]，这3个元素都是struct student 类型的结构体变量，并且每个元素都又包含自己的成员**

**拓展：**

**结构体变量不能进行加减乘除  
但结构体变量能相互赋值，前提是这两个结构体变量的结构体类型必须相同**

**指针(c语言的灵魂)**

**牢记: 指针就是地址,地址就是指针，指针和地址是同一个概念**

**地址是内存单元编号(从0开始的非负整数)**

**什么是内存单元编号?**

**内存实际上就是一个个小格子,每个格子里可以存放数据,并且每个格子有唯一的编号来标识,这个编号也就是内存单元编号,也叫地址**

**内存单元编号的范围:**

**内存单元编号的范围和内存有关, 假如电脑的内存大小是4G，范围就是 0 - 4G-1，是8G，就是 0 – 8G-1**

**什么是指针变量?**

**指针变量也是变量,与其它类型的变量不同的是指针变量内只能存放地址(内存单元编号)，并且指针变量也有自己的地址(内存单元编号)，简而言之，指针变量是用来存放地址的变量**

**习惯上喜欢把指针变量简称指针，但要注意区分指针与指针变量是两个完全不同的概念**

**指针变量的大小**

**无论指针变量的基类型是什么，或者指针变量指向的对象占多大字节, 指针变量**

**就只占4个字节大小，简单来说所有类型的指针变量都是地址，所占字节数均为4**

**指针变量包含的两个含义**

1. **存放以内存单元编号表示的地址**
2. **指向地址对应的数据类型（指向存储单元的数据类型）**

**eg:**

**int a = 1；**

**int \*p； // (1) 该指针变量p只能存放整型数据的地址**

**// (2)指向地址对应的数据类型,这里是int 类型**

**p = &a；//此行代码有3个含义**

**p存放了a的地址**

**p指向了a**

**\*p就是a**

**printf(“%d”, \*p)； //输出结果为1**

**注: \* 表示变量p 是一个指针变量, int 是该指针变量p的基类型,也可以说 int \* 是指针变量p的数据类型**

**什么是地址对应的数据类型？**

**该地址中存放的数据的数据类型,就是地址对应的数据类型**

**什么是基类型?**

**首先定义指针变量必须指定其基类型, 前面我们介绍过基本的数据类型（如:int char float...）,既然有这些类型的变量，就有指向这些类型的变量的指针， 所以指针变量的基类型是用规定此指针变量可以指向什么类型的变量，或者也可以说规定此指针变量可以存放什么类型数据的地址**

**\*p 该如何理解**

**(1) \*p可以理解为 取出p中存放的地址中的内容**

**解释: p中存放的地址就是a的地址, 因为p已指向a，取出p中存放的地址中的内容， 也就是取出a的地址中的内容,也就是1**

**(2)\*p是以p的内容为地址的变量，也就是变量a**

**(3)书本上的解释是: \*p 是p指向的对象，也就是a**

**指针的分类:**

**基本类型指针**

**eg: int \*p ，float \*p，double \*p，char \*p ………**

**指针与字符串(指向字符串的指针)**

**eg:**

**char \*p = “abcdef”；**

**// p是字符指针变量，存放了该字符串中首字符的地址，并指向首字符，在不引起误解的情况下，可以理解为p指向了整个字符串**

**注意:**

**char \*p；**

**scanf(“%s“，p)；**

**//错误，因为p是一个没有指向的野指针，不能向其里面存放字符串**

**char a[N]；**

**char \*p = a；**

**scanf(“%s”，p)；**

**//正确，向p指向的地址中存入一个字符串，实际就是往字符数组a中存放一个字符串**

**拓展：**

**系统是如何通过输出字符指针变量名从而达到输出整个字符串**

**过程：**

**系统会先输出字符指针变量名p所指向的字符串中的第一个字符，然后自动使p加1，使之指向下一个字符，再输出该字符……如此直到遇到字符串结束标志，字符串输出完毕**

**指针与数组(包括指向数组元素的指针,指向一维数组的指针, 指针数组)**

**eg：int a[3] = {1，2，3}，\*p；**

**char s[3][10]；**

**int b[2][4]；**

**int (\*pArr)[3]，(\*px)[4]；**

**char (\*pS)[10]；**

**p = &a[0]；//p是指向数组元素的指针，也可以写成p = a；**

**pArr = &a；**

**px =b； //px和pArr都是指向整型一维数组的指针**

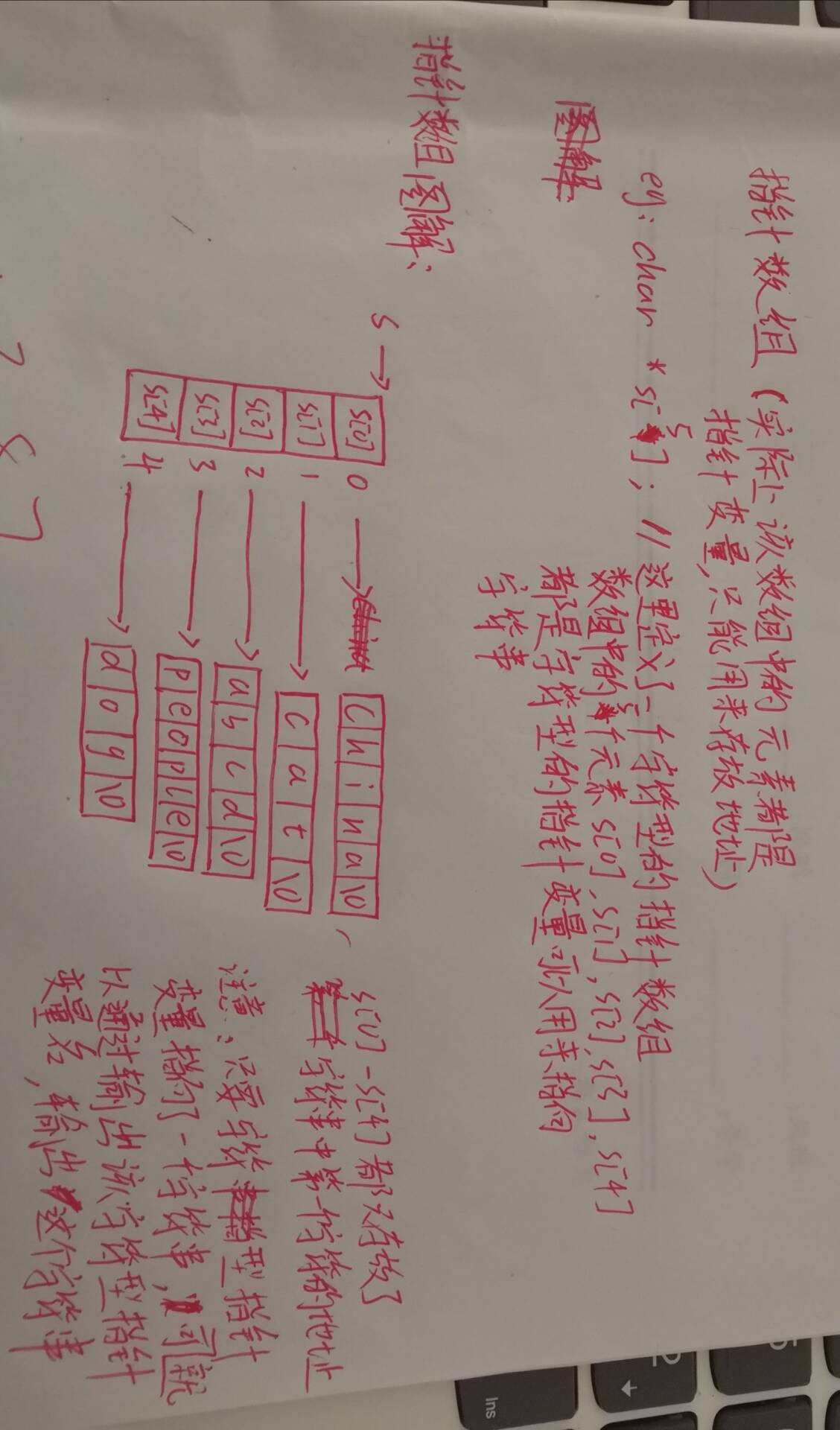
**pS = s； // pS 是指向字符型一维数组的指针**

**注意：int (\*pArr)[3]，(\*px)[4]；**

**char (\*pS)[10]；**

**的括号不能省，省了就变成了指针数组**

**指针数组**

****

**指针与函数(指向函数的指针)**

**指针与结构体(包括指向结构体数组的指针，指向结构体变量的指针)**

**指针与文件(指向文件的指针)**

**多级指针(指向指针的指针)**

**eg:**

**int a，\*b，\*\*c；**

**b = &a；**

**c = &b；**

**//指针变量c就是一个指向指针的指针，\*c就是指针变量b，b的值是整 型变量a的地址，\*b就是a，所以\*\*c 就是a**

**注：指针数组名，二维数组名就是指向指针的指针**

**文件**

**在程序设计中主要有两种文件**

1. **程序文件。比如源程序文件( .c)，目标文件(.obj)，可执行文件(.exe)，程序文件的内容是程序代码。**
2. **数据文件。其内容不是程序代码，而是供程序运行时读写的数据。比如运行一个求学生平均成绩的程序，需要输入该学生的每门课成绩的数据**

**注: 数据文件包括文本文件(又称ASCII文件)和二进制文件**

**文件类型指针(指向文件的指针)**

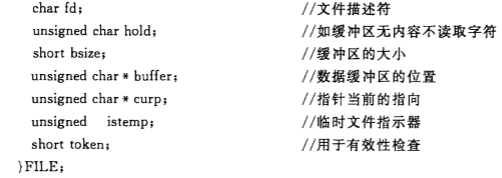
**每个被使用的文件都会在内存中开辟一个相应的文件信息区，用来存放文件的有关信息(比如该文件的名字，文件的状态，文件当前的位置...)**

**如何实现文件信息区?**

**用一个结构体变量来实现文件信息区，该结构体变量的结构类型为FILE，是由系统定义的，可直接使用该结构体类型来定义变量，并且用该结构体变量中的成员来保存文件的一些信息**

**eg:**





**注:FILE结构体类型的定义和声明包含在”stdio.h”中**

**FILE f1；**



**但一般不用此方法，而是定义一个指向FILE类型的指针(即指向文件的指针)**

**eg: FILE \*fp；**



**也就是说通过文件指针变量就能找到该指针变量指向的文件**

**文本文件的基本操作**

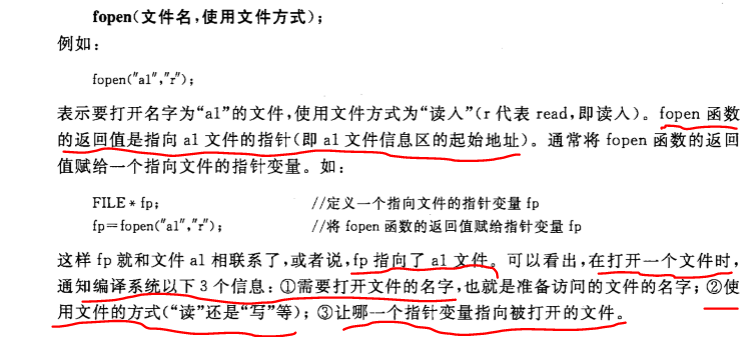
1. **打开和关闭文件**

**打开文本文件:**

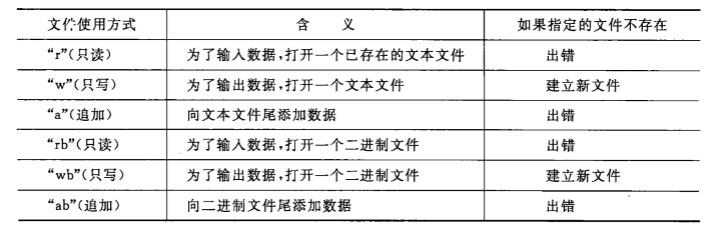
**为文件建立相应的信息区(存放有关文件的信息)，和文件缓冲区(用来暂时存放输入输出的数据)**

**用fopen 函数来打开数据文件**

**fopen函数的调用方式如下**



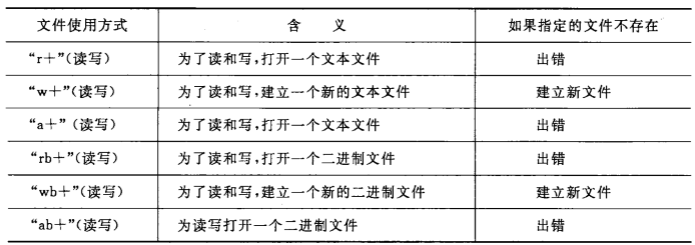
**使用文件的方式有:**



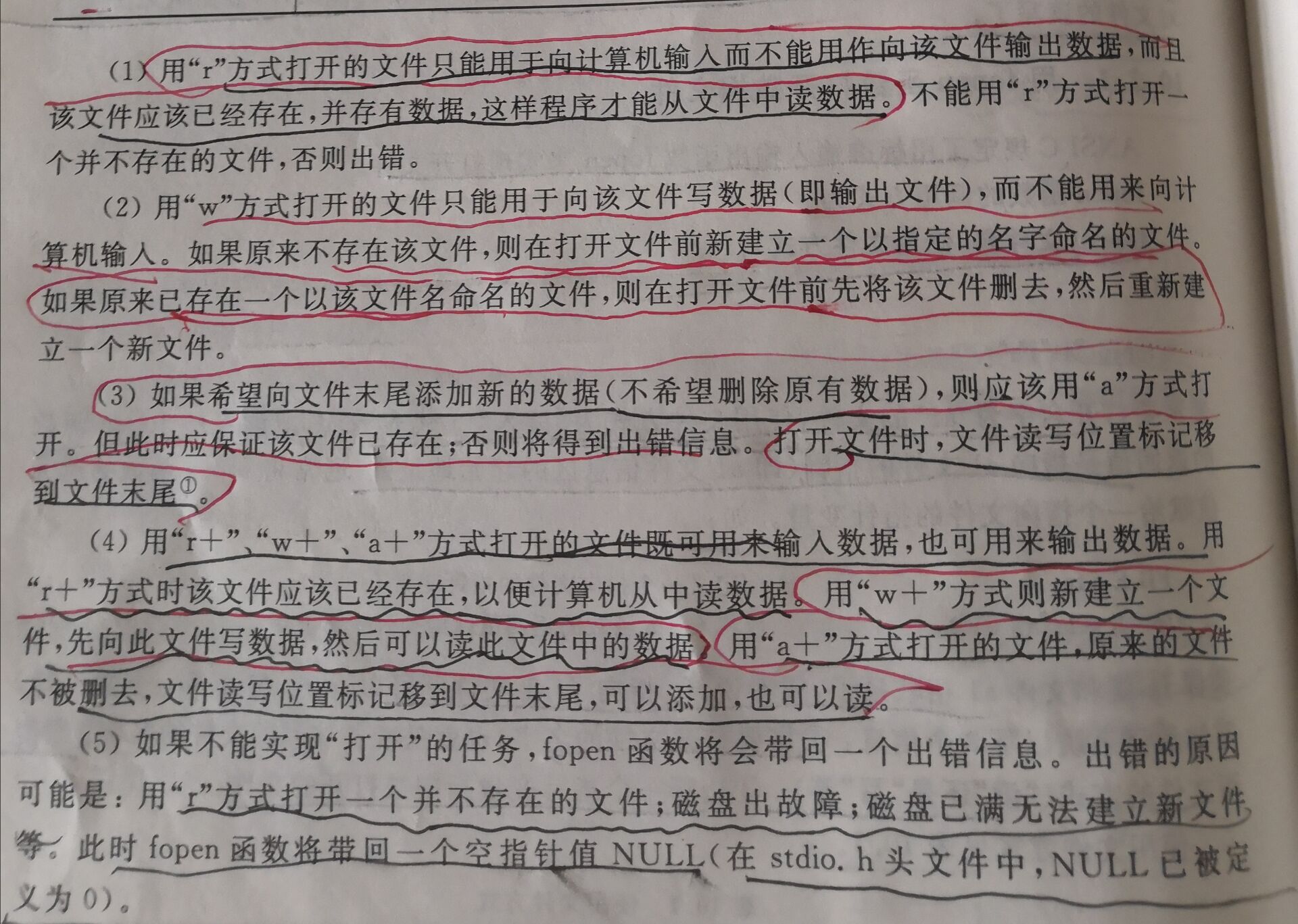
**这里的输入数据是指从文件中读取数据(即读取出这个文件中的数据，用来使用)**

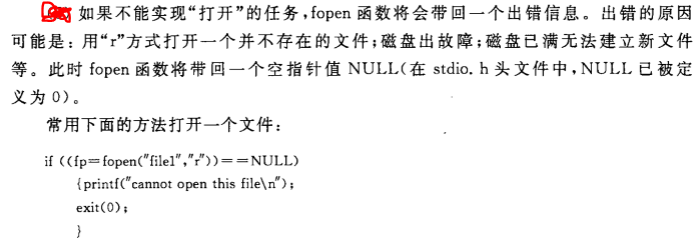
**这里的输出数据是指将数据输出到文件中（将数据写入到这个文件中）**

**对文件的输入与输出操作也就是对文件进行读与写**



**注：a+ :打开一个文本文件后，文件位置标记移到文本末尾，即可以在文本末尾添加数据，也可以从文本末尾开始读取数据，如果要打开的文件不存在，出错**





**打开不成功fopen函数返回空指针NULL**

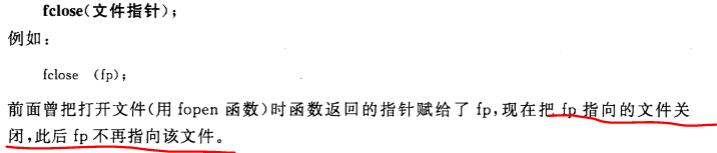
**exit是库函数，作用是使程序终止，使用该函数必须再程序开头包含stdlib.h头文件**

**关闭文本文件:**

**撤销文件信息区和文件缓冲区，使文件指针变量不再指向该文件**

**用fclose 函数来关闭数据文件**

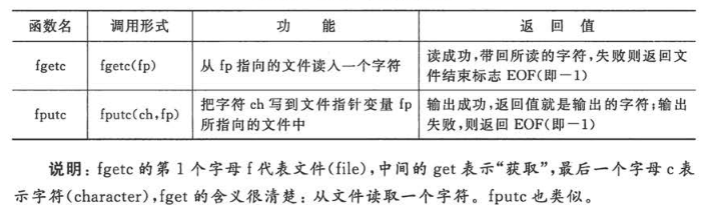
**fopen函数的调用方式如下**



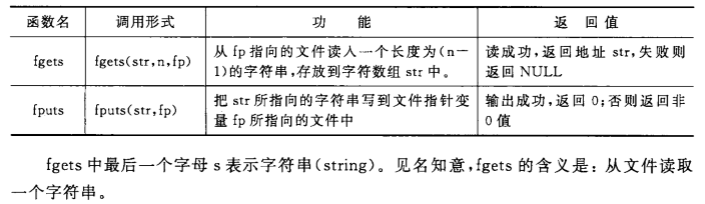


**EOF(-1)是文件结束标志**

1. **向文本文件中读取/写入字符**

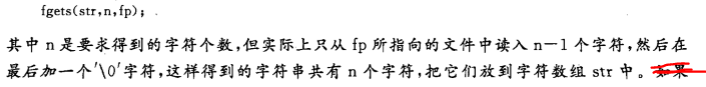


1. **向文件文件中读取/写入字符串**



**fgets函数的原型为:**

**char \*fgets (char \*str, int n, FILE \*fp)**



**fputs函数的原型为:**

**int fputs (char \*str, FILE \*fp);**

**eg:fputs(“China”,fp); //把字符串输出到fp所指的文件中**

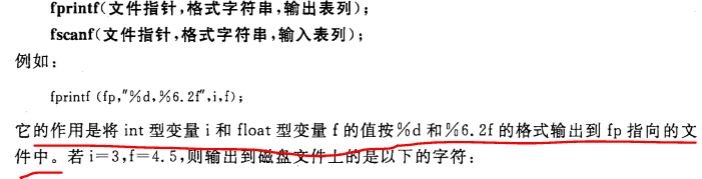
**fgets和fputs功能类似于gets和puts，前者以文本文件作为输入输出对象，后者以终端作为输入输出对象。fgetc和fputc同理**

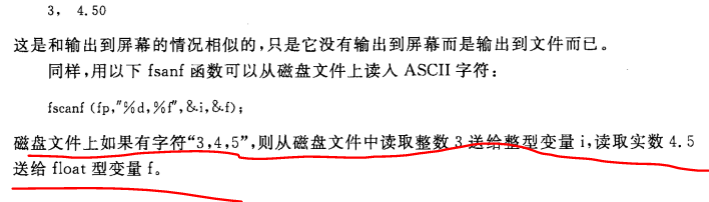
1. **用格式化的方式读写文件**

**即对文本文件进行格式化输入输出需要用到fscanf 和 fprintf 函数**

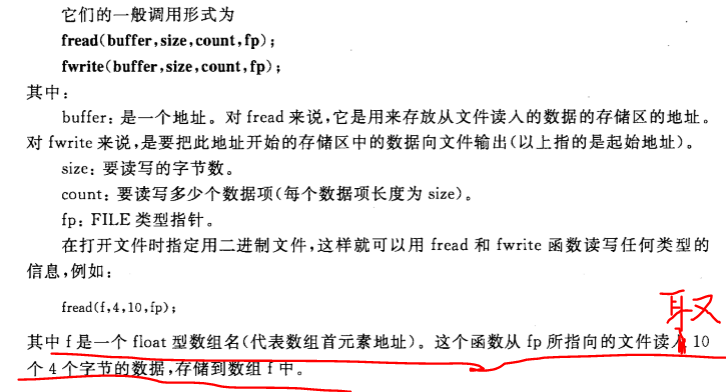
**fscanf 和 fprintf 与 scanf和 printf 相仿**

**前者读写对象是文本文件，后者是终端**





1. **用二进制方式向文件中读取/写入一组数据**



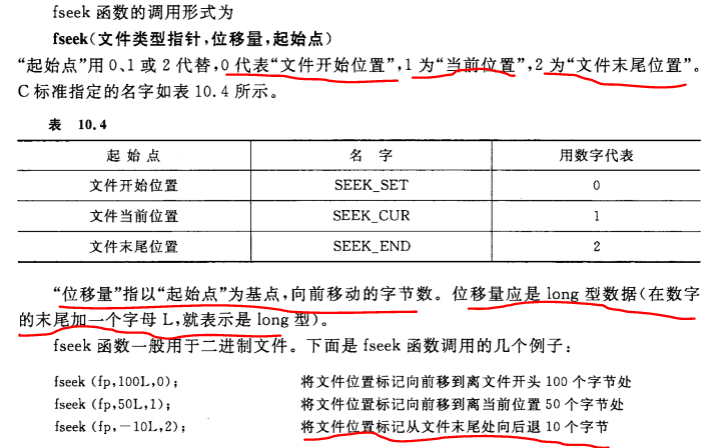
**buffer :是一个地址.对fread来说，是从二进制文件中读取的一组数据，存放到该地址中；对于fwrite来说,是将该地址中的一组数据，写入二进制文件中**

**fwrite(f,4,10,fp)**

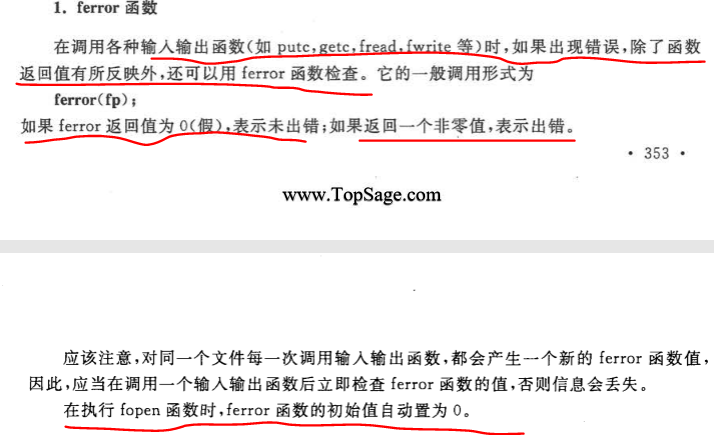
**将地址f中的10个，占4个字节的数据写入fp所指的文件中**

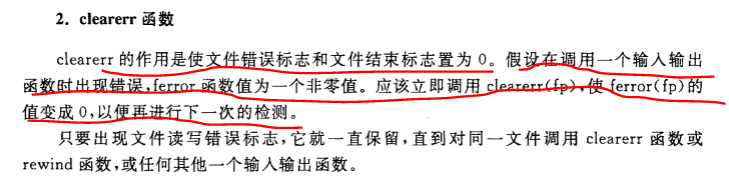
**文件位置标记的定位**

1. **用rewind函数使文件位置标记重新指向文件开头，此函数没有返回值**
2. **用feof函数用来检测当前文件位置标记是否已指向文件末尾，处于文件末尾返回真(一个非0值)，不处于返回假(0)**
3. **用ftell函数测定文件位置标记的当前位置**
4. **用fseek函数改变文件位置标记当前的位置**



**文件读写的出错检测**





**拓展：数据文件可以顺序读写，也可以随机读写**

**静态分配内存与动态分配内存**

**静态分配内存与动态分配内存的区别  
静态分配内存是由系统自动分配，由系统自动释放  
静态分配内存是在栈分配的，不可以跨函数使用  
动态分配内存是人为分配的，手动释放  
动态分配内存是在堆分配的，所以可以跨函数使用**

**预备知识**

**malloc函数**

**作用: 在内存的动态存储区分配一个长度为size的连续存储空间**

**函数原型是: void \* malloc(unsigned int size)  
//从函数原型可以看出malloc返回的是一个无类型指针，并且malloc函数 的形参来看，malloc函数只能接受一个参数，且该参数必须是整数**

**calloc函数**

**作用: 在内存的动态存储区分配n个长度为size的连续存储空间**

**函数原型是: void \*calloc(unsigned n, unsigned size)  
eg:  
     int p = (int \*)calloc(50, 4) ；   
     分配了50\*4个字节的动态存储空间，并且有指针变量p指向它  
 可以看出calloc返回的也是一个无类型指针**

**realloc函数**

**作用：如果已经通过malloc或calloc.函数获得了动态存储空间，想改变其大小，使用realloc函数重新分配**

**函数原型是 void \* realloc(void \*p，unsigned int size)  
可以看出realloc返回的也是一个无类型指针**

**eg: int \*p = (int \*)malloc(sizeof(int))；//p指向占4个字节大小的存储空间**

**int \*p = (int \*)realloc(p，100)；**

**//将p指向的存储空间由原来的4个字节扩大到100字节**

**free函数**

**作用：释放指针变量p所指向的动态存储空间**

**函数原型是 void free（void \*p）**

**注:**

**以上四个函数都包含在 <stdlib.h>头文件中，使用前需加上该头文件**

**什么是无类型指针？**

**void \*（无类型指针）是一种特殊的指针，不指向任何类型的数据，如果需要此地址指向某类型的数据，要先对此地址进行强制类型转换，无类型指针的基类型是void**  
**通过数组静态分配内存  
缺点：  
 1.不能手动释放。函数运行期间系统为数组分配内存，函数运行结束又由系统 释放  
 2.数组一旦定义，长度固定，不能在做修改  
 eg：int  a[5];  
 该数组长度为5，在函数运行期间不能在做修改  
 3.在一个函数里定义了个数组，在函数运行期间可以被其他函数调用，一旦该 函数运行结束，就不能在被调用  
 4.必须事先知道数组大小**

**通过malloc函数动态分配内存  
 1. malloc函数里只能有一个参数，且必须为无符号的整型，即 unsigned  int  
 eg: int \*p=(int \*)malloc(sizeof(int));  
           \*p=10；  
     p指向了一个占4个字节大小的存储空间  
    \*p 就等价于int 类型的整型变量  
   sizeof 用来测量数据类型，和变量的大小  
 2.malloc函数在malloc. h或stdlib头文件中，使用前需声明  
 3.malloc 函数返回的是void \*类型的地址，也是个无类型指针，所以需要强制 类型转换，具体转换成什么类型的地址，看具体情况  
 eg: int \*p=(int \*)malloc(4)；  
 //指针变量p指向了一个4字节大小的动态存储空间  
4.假设malloc 函数分配了大小为n个字节的内存空间，但只返回第一个字节的地址  
拓展  
   也可以通calloc函数动态分配存储空间  
 其作用是在内存的动态存储区分配n个长度为size的连续存储空间  
 eg:  
      int p = (int \*)calloc(50,4);  
     //分配了50\*4个字节的动态存储空间，并且有指针变量p指向它**

**静态数组与动态数组  
静态数组：**

**eg: int a[5]；**

**//该数组静态分配了20字节的内存空间，即该数组占20字节的 内存空间大小，且每4个字节被当做一个int变量使用  
数组名a是指针常量，存放了第一个元素的地址，即指向了20个字节中前4和字节的地址**

**动态数组**

**eg:**

**int \*pArr，len；  
printf("请输入存放元素的个数：")；  
scanf("%d",&len)； //假设len为5  
pArr=(int \*)malloc(sizeof(int)\*len);  
//动态分配了20字节的存储空间并且由指针变量p指向它，每4个字节被当做一个int变量使用，pArr指向前四个字节，pArr+1指向后四个字节，即\*pArr是第一个元素，\*（pArr+1）是第二元素，所以有pArr[0]，pArr[1] 等价于a[0]，a[1]**

**结论：**

**类似于动态的构造了一个一维数组：int pArr [len]；  
该数组名是pArr，数组长度是len，该数组中每个元素是int 类型**

**进一步操作有:**

**向数组中赋值  
for（i=0;i<len;i++）  
scanf("%d"，&pArr[i])；**

**在函数运行期间可以随时动态扩充或缩小该数组长度（区别于静态数组）  
 realloc(pArr，100)；**

**//将pArr所指向的动态存储空间扩大到100字节，将该数组长度扩充为100，注意不是在原来的基础上+100，即将原来的pArr[5]改为pArr[100]；**

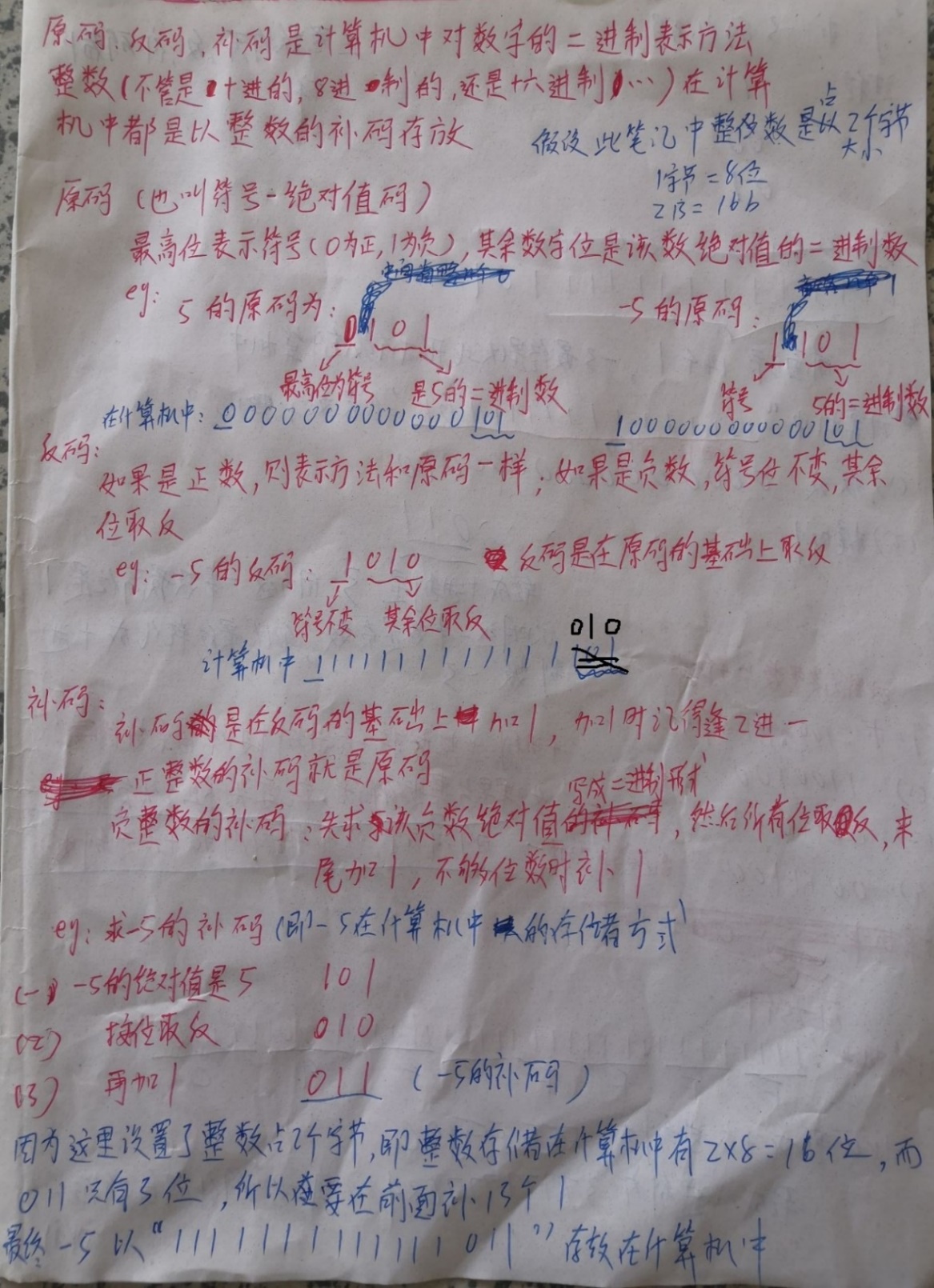
**在函数运行期间可以随时手动释放该数组  
free(pArr)；  
释放指针变量pArr指向的动态存储空间，即把该数组释放了**

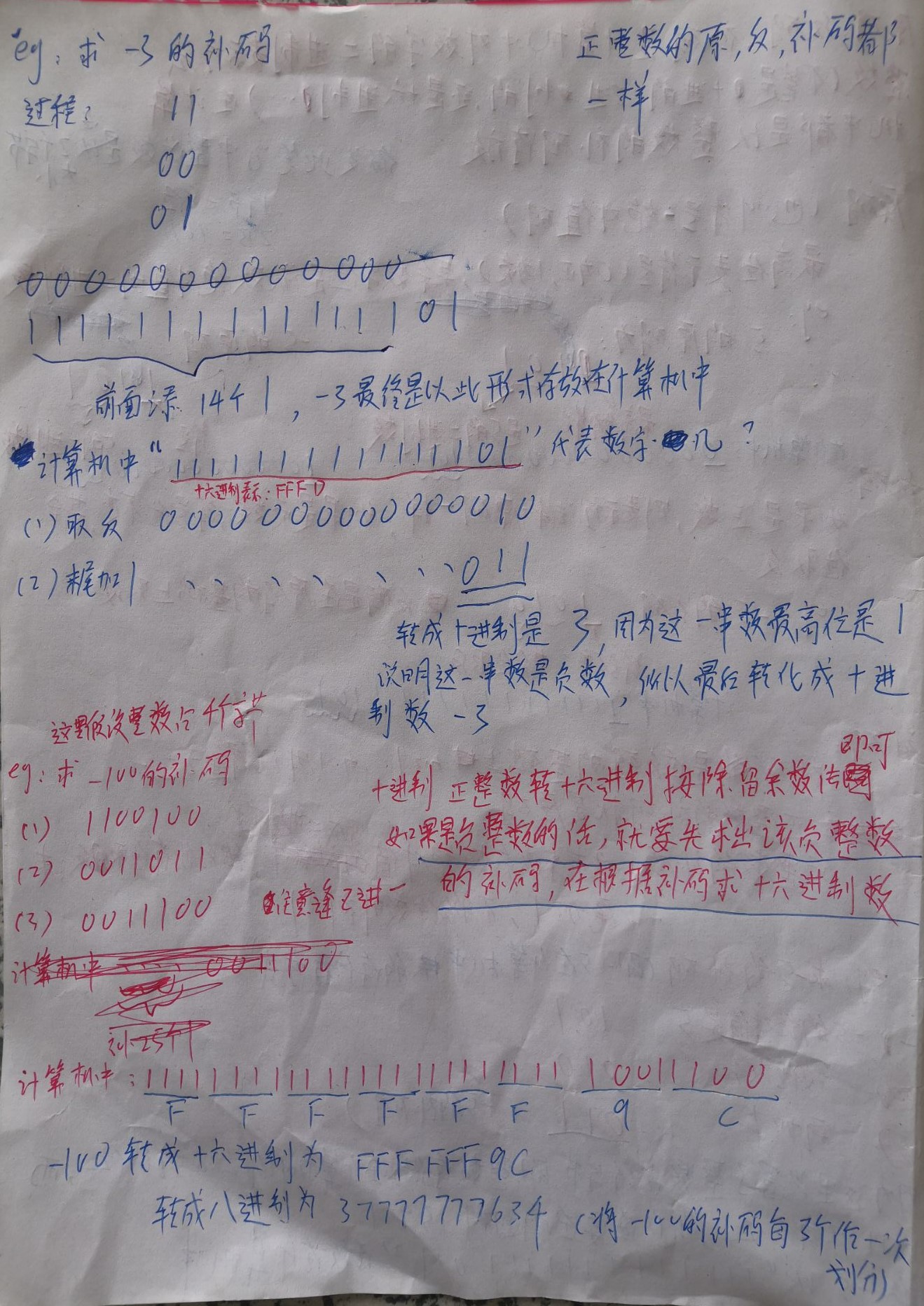
**动态数组较静态数组的三个优点**

**1. 在函数运行期间可以随时手动释放**

**2. 在函数运行期间可以随时动态扩充或缩小数组大小  
3. 不必事先知道数组大小（长度）**

**原码，反码，补码**

****

****

****