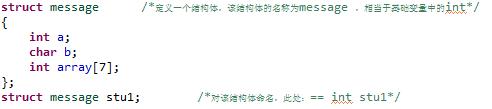
C学习笔记

一：关于typedef struct n和struct n结构体的区别于使用:

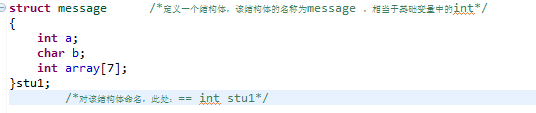
结构体的定义一般有以下两种方式：

1：

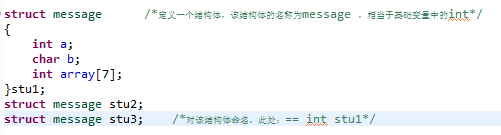


Struct作为关键字存在，说明是一个用户自定义的结构体变量，而message则是该结构体的具体名称。先定义某个结构体的成员变量，然后再对该结构体进行变量声明。

一般为了省事，都是选择直接在定义结构体变量的时候进行变量声明：

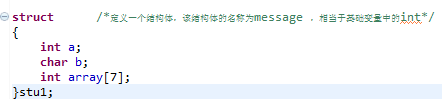


这样就相当于在定义了一个新的数据类型的时候就直接对其进行了变量声明。当然如果后续需要再对其进行其他的变量声明，可以如下：



以上声明的变量stu1、stu2、stu3都具有一致的结构体成员。

关于message：messgae作为结构体的名称，当只需要进行一次变量声明的时候可以省去，即：



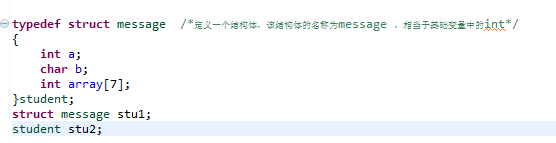
但是这样则无法再进行其他该结构体类型的变量声明了。

关于typedef：typedef为C语言的关键字，作用是为一种数据类型定义一个新名字。这里的数据类型包括内部数据类型（int,char等）和自定义的数据类型（struct等）。

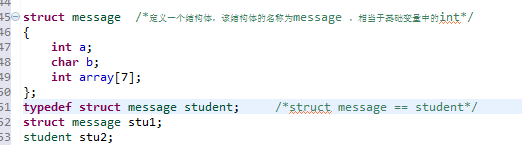
在编程中使用typedef目的一般有两个，一个是给变量一个易记且意义明确的新名字，另一个是简化一些比较复杂的类型声明。例如：



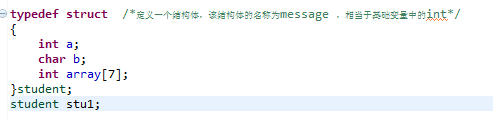
当typedef和struct一起在结构体中出现时，就可以使得结构体的使用更加的方便简单：



以上代码等效于：

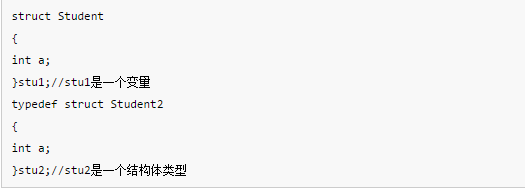


同样，当该结构体只需要进行一次变量声明的话，同样可以省去message：

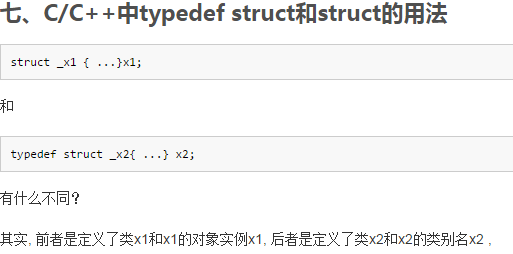


结构体使用的顺序：定义一个结构体并命名、利用结构体声明一个变量。在此，对于结构体的名称可以省略，但是结构体是必须要定义的；变量可以在定义结构体的时候直接声明（省去了结构体名称）或者先定义命名结构体，再声明变量（结构体名称存在）。

用了typedef和没使用typedef的区别：



或者：





当只需要声明少数量的或者临时的变量时，可以直接用struct来完成，但是当需要声明多个变量或者在后面的程序中还要在进行其他的声明时，为了方便起见，就会使用typedef 来用

简洁明了的结构体别名替代。

二：关于.c文件和.h文件：

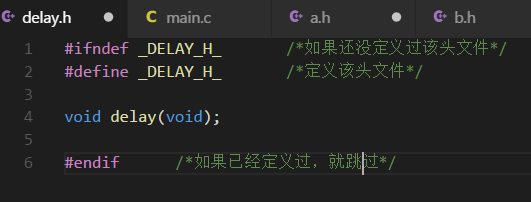
工程都是有主要文件（main.c）和各大功能模块文件（delay延时）组合而成的。而每一个功能模块负责得具体功能实现，一般都是通过函数来实现。一个功能模块中某一功能函数的具体工作流程为：被声明（出生）>被定义（成长）>被调用（建设国家）。但是这些功能函数亦或是功能模块不是直接写在主要文件main.c中的，因为有些模块属于基本的、项目多处公用的，因此一般作为独立于main.c而存在，当工程中某一处需要用到时就可以单独调用，从而实现多处、多次的使用。功能模块是如何存在并被如上所调用的呢？某一独立的功能模块，分为两个文件，一个是.h，另一个是.c，模块的功能由一到多个函数完成，这些函数则又是，将其声明放在.h中，而在.c中进行具体的定义与实现。就相当于.h的作用是展示该模块“能实现什么”，而.c的作用则是“该怎么实现”，这样一来，分工明确，各自负责。当需要调用的时候，就不需要全部把实现代码也搬进去main.c了，而只需要引用（#include）所需模块的头文件便可。这样，当需要某一函数的时候，通过引用的头文件，直接找到该函数并进行调用。

一般每一个模块中，相应的.c文件中都需要包含同名的.h文件。

但是新的问题又来了，对于一个工程，存在多个.c文件和多个子功能模块，但是有些功能模块是属于基本功能，会被其他的多个功能模块互相多次、多处调用，在最后汇总成main.c时，就会出现，在同一个.c文件中某个功能模块被重复调用（某个头文件被多次重复调用）的错误。为避免以上错误，就要用到#ifndef、#define、#endif了：

三：关于#ifndef #define #endif的用法

在每一个头文件的初始处写明#ifndef用法，可以避免该头文件由于被多次引用而报错。具体使用举例：（以delay.h文件为例）



这样在后面的任何一个地方被多次调用时就都可以避免错误了

一般最好每一个头文件都这么申明一下。

四：关于编译器的工作过程。

1：预处理。对于一个工程，会有很多个c文件和h文件。编译器工作是以c文件为单元的，因此在预处理阶段，对c文件中的伪指令和特殊符号进行替换，得到一个没有头文件，没有宏定义，没有特殊符号的c文件，该c文件和源c文件含义相同但是内容不同。将替换的内容有以下几种：

1.1：宏定义指令#define #undef

1.2：条件编译指令#ifdef #else 等

1.3：头文件包含指令#include

1.4：特殊提示符

这样，就得到了一个个“中间c文件”。（生成.c）

2：编译。（中间c文件>汇编文件.s）通过词法分析和语法分析对所有语句进行语法规则的检测，在满足其语法和词法规则的基础上，将其所有的“中间c文件”翻译成等价的独立汇编文件。（生成.s文件）

3：汇编。（汇编.s文件>目标文件.o）汇编是将所有的汇编文件转换成机器代码，也就是可执行文件。目标文件由段组成，一般分为代码段和数据段，代码段用来存放指令，数据段用来存放各种变量和常量。（生成.o文件）

4：链接。（目标.o文件>可执行文件.hex）链接是将上面生成的所有独立汇编文件按照一定的顺序连接成可执行的目标文件。（生成.exe/.hex）

五：关于sizeof（）与strlen()的区别；

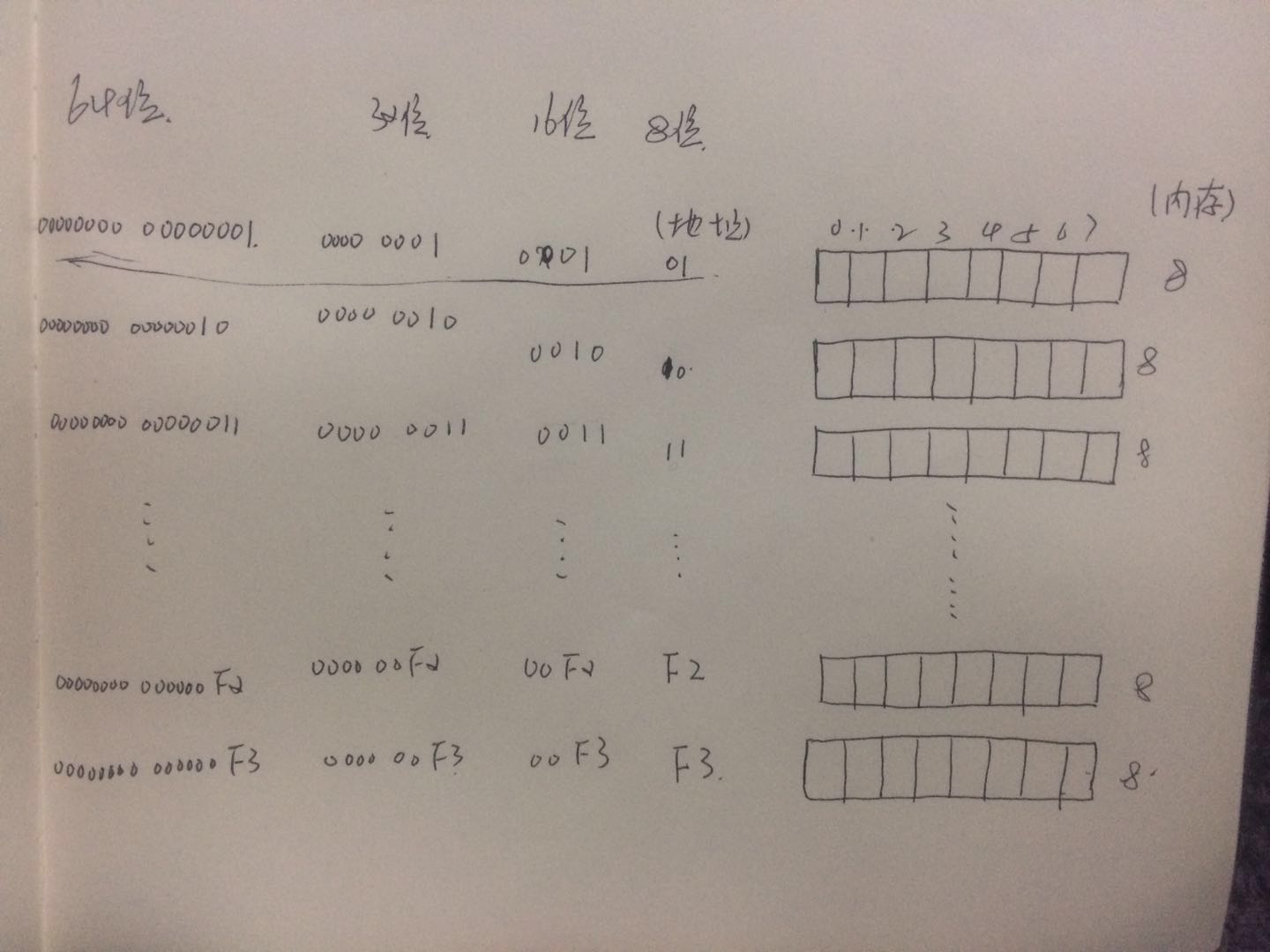
01：sizeof是运算符，旨在计算该变量、数组、函数在声明好之后固定分配的内存空间容量，一旦变量被声明，该大小就是固定的，因此在编译时计算；而strlen则是一个函数，它用来计算字符指针类型的字符串的具体输入长度，由于具体字符在运行时才出现，因此在运行时才计算。

02：sizeof的参数可以是数组、指针、类型、变量、函数；但是strlen的参数只能是字符指针。数组做sizeof的参数不退化，但是当做strlen的参数的时候会退化为指针。

六：关于数据结构与指针；

指针：首先，指针也是一个变量，也有类型和它自己的地址以及数值，它本身的类型决定了编译器将它所指向的地址内的变量当做什么类型的数据来处理。它本身地址的长度由其声明的变量类型决定，它本身地址内部的数值，是另一个变量的地址，由于处理器一般都是32位的，因此该数值必定是32位的一个整数，该数值占用一个字节长度，但是该指针本身的内存大小却可能是int（2字节）、char(1字节)或者其他长度。用sizeof()测得，在32位的系统里，一个指针本身占4个字节的长度，但是它里面却装了一个32位的整数。

不管计算机是多少位的，不变的是一个字节占用8个位，计算机的内存都是由一个个8位的单位字节累计起来的。当编译器是16位时，说明一次能处理最多两个字节长度的变量数值；当编译器是32位时，说明一次能处理长度最大是四个字节单位的量；而当是64位时，说明一次能处理长度最大为8个单位字节长度的量。也说明了为什么计算机的位数最低是8位。



以32位的处理器为例，其地址是32位的，占用了四个字节，而指针变量的值，就是用来表示某一变量的地址的，因此该为指针分配的空间大小也必须是四个字节，只有这样才能表示得了一个32位的地址。

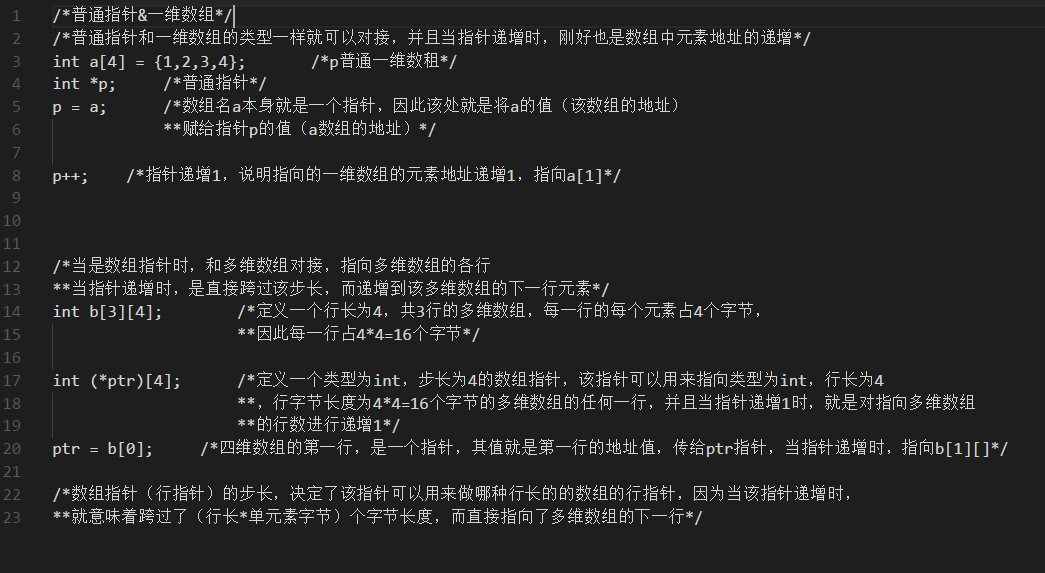
02：指针数组与数组指针。

指针数组：

int \*p[4] ：由指针组成的数组，该数组内的四个元素都是指针。该四个元素的值都是某变量的地址值。指针数组，即，指针的数组。指针数组就是一堆同种类型的指针的数组形式的集合,本质还是数组。

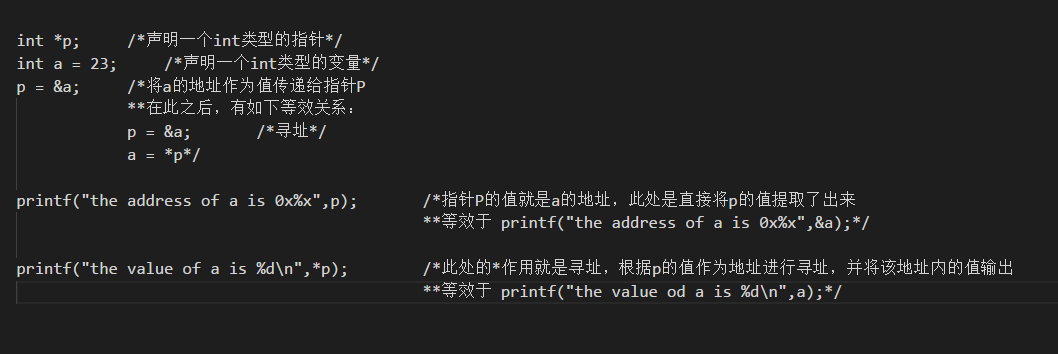
数组指针：

int （\*p）[4] ：指向一个含四个普通int类型变量元素的数组的指针。数组指针，即，数组的指针，本质是指针，只不过用来指向特定行长的多维数组而已。由于优先级的原因，（）比[]优先级更高，因此，先是一个指针p，然后才是一个步长为4的数组。当指向的数组是一维数组时：



不管是普通指针也好，还是数组指针也好，在使用的时候都是，先声明一个指针，然后对其赋值（也就是，让该指针hi走向一个具体指）

在指针中，有如下关系：



03：指针函数与函数指针。

指针函数：int\* read();

本质是一个函数，只不过其返回值是一个指针，其类型就是该函数声明时的返回值类型，用于需要地址的情况。

函数指针：int (\*f) (int a, char b);

本质是一个指针，只不过只能指向符合要求“形参为int a 和char b”，“返回值为int”的函数，在声明函数之后，就可以将符合条件的函数的地址作为值传给该指针，这样后面就可以直接通过调用该指针来调用其指向的函数了。函数指针可以作为某个函数的形参。

七：关于计算机的位数，内存长度与变量字节长度等的相关；



八：关于嵌入式软件的单元测试；

九：makelist、makefile、cmake的用法与练习；

Makefile的作用：

1：告诉编译器要生成哪些单独的可执行的.o文件，省去了大型项目中的手动生成繁琐；

2：告诉编译器，生成最终的目标文件需要哪些模块执行文件以及如何寻找链接；

十：关于IDE的Link。

每一个集成开发环境（如eclipse）都是集文本编辑、编译器、调试于一体的。用户只需要按照自己的需求写好每一个.c和.h文件，然后点击IDE软件上的“编译”或是“调试”等按钮，就可以使的软件自动进行文件的编译、链接直至生成最终的可执行文件。那么该连接功能具体都做些什么呢？

由以上的学习笔记可知，一个工程若想生成最终的可执行文件，一般都需要经过：

预编译>编译>汇编>链接这四步，即，先以每一个独立的.c文件为单元，将其全部转换成独立的目标文件，然后再将生成最终的可执行文件所需的模块目标文件、库等，进行链接，生成可执行文件。而Link的作用就是由IDE软件一键自动完成以上工作。可以用以下示意图演示其工作过程：

假如该工程有三个模块.c文件，第一步，生成目标代码：

1.c -o 1.o /\*生成1.c文件的目标代码1.o\*/

2.c -o 2.o /\*生成2.c文件的目标代码2.o\*/

3.c -o 3.o /\*生成3.c文件到目标代码3.o\*/

第二步，链接需要的目标代码来生成可执行文件：

link 1.o 2.o 3.o -o a.out /\*生成最终需要的可执行文件a.out\*/

而ide的一键式工作就是上面的两步。

十一：shell脚本和makefile的区别。

共同点：都是Linux下shell指令的合集，通过脚本，可以一键完成某些繁琐操作，语法规则都很像。

不同点：shell只能完成简单的指令集合，而makefile则可以完成复杂的工程编译工作；二者的语法有一些细节不同；

十二：关于makefile。

Makefile的好处：

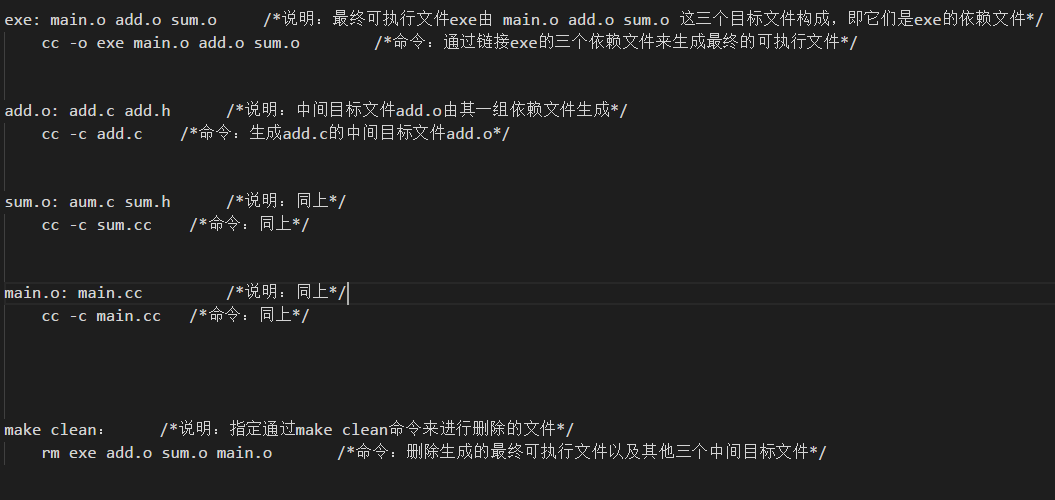
01：当确定好整个工程的依赖关系以后，当之后需要对某一个源文件进行更改时，便可以直接利用make进行更新，而不需要重复的操作，相当于为该工程制定了一套编译规则，之后的每一次重复性的编译都只用按照该规则进行就行了。

02：makefile可以由开发者自己制定编译规则，可以规定编译哪些，不编译哪些，以及生成哪些目标文件，自己制定工程的依赖关系。

Makefile的语法规则(以下面的工程为例)：



新建makefile并编写如下：

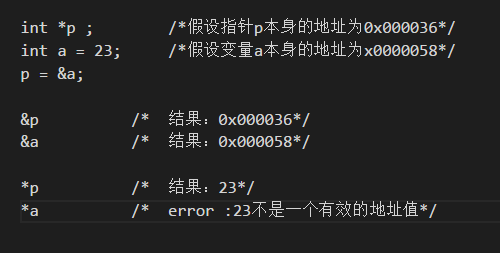


十三、关于运算符&和\*的区别。

&：取地址，即提取该变量的地址。

\*：寻址，即根据该地址，对该地址内的值进行提取

二者的运算参数和返回值斗都不一样的，如下：



十四、NULL ,0, \0,的区别。

int \*p = 0 /\*定义一个指针并使其值为0，说明该指针不指向任何变量，不具有任何

\*\*\*\*\*其他变量的地址值，但是该指针是真实存在的，它有它自己的内存地址\*/

在用来表示地址的时候，NULL 和0都可以表示空值，但是当表示数值的时候，NULL不等0；

15：堆和栈的区别与应用

16：进程与线程

17：二叉树

18：链表

2018.04.19

常量

常量可分为”数字常量、“字符常量”、“字符串常量” “枚举常量”。

数字常量：数字常量按照不同的形式又可以进行划分：

按照长度：短型：int a=8

长型：long int a=123456789L

按照是否整数：整型：int a= 12

浮点型：float a = 13.2f

有无符号：无符号：unsigned int a =3u(只能表示大于等于的数)

有符号：signed int a= -3(int -3) int 一般默认是有符号的 （正数、负数、0都可以表示）

按照进制：十进制：int a = 12

八进制：int a = 07;（八进制的数字以0开头）

十六进制：int a = 0x32 （十六进制以0x开头）

以上几种分类可以互相糅合，同一个数据可以包含多种表示方式，如：

Unsigned Long double float a = -0x12345678.999L

字符常量：

字符常量的作用：

1：一般用‘x’的形式来表示，其中x可以是任意值，字符常量的值是该字符在ASCII中的数值。

2、和一起构成特殊功能字符，比如\n，代表换行

3、’\0’用来强调表示其值为0（但是当‘0’时不是，‘0’表示ASCII中的48），这里表示了特殊的强调关系。

字符串常量：用来表示一串字符，格式为 “这是一串字符”，可以看做是字符数组，一般会在字符串末尾添加一个’\0’表示文本结束。因此一般存储字符互的物理长度要比实际给出的字符串长度大1。但是当用函数strlen（）扫描字符串长度时，却并不包括‘\0’。

字符常量与字符串常量的重要区别：字符常量只表示了在机器字符集中的一个数值（内部表示值），而字符串常量则是一串字符和一个‘\0’结束符的组合。

枚举常量：整数常量的列表 格式如下：

enum boolean {YES,NO};

默认第一个常量的值为0开始起算。如果对后面的常量的值没有特定值指定，那就将递增叠加，如果途中遇到了新的数值，那么就以该值为基础。再继续叠加。

同一个枚举中常量名称必须各不相同，但是各自的值可以相同。