gcc编译器拓展了c的语法，有许多c标准并没有规定的功能。

默认情况下，gcc是开启gnu拓展的。我们也可以控制gcc使用哪种标准

只需使用编译选项 -std= 即可 =后面可选很多内容，例如

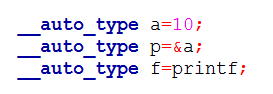
-std=c99 表示令gcc遵守c99标准

常用的选项有

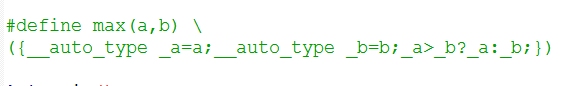
-std=c89 -std=c99 -std=c11 -std=gnu11 -std=gnu99 等

类型推导

\_\_auto\_type 拓展关键字功能与c++11的auto的类似，用于自动类型推导，起占位符的作用，与c++11不同的是，该关键字只能作用于一个变量的定义，使用如下

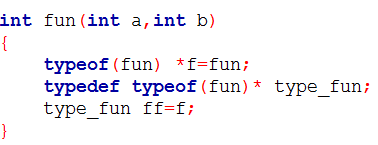


该关键字一般用于宏中，如



这样可以消除例如max(i++,++j)的影响，关于({})查阅内嵌表达式

另一个关键字为typeof ，使用该关键字可以获取表达式的类型，例如



该关键字一般也使用在宏中，在获取复杂类型是也非常有用，比如上例中定义一个指向某函数的指针类型

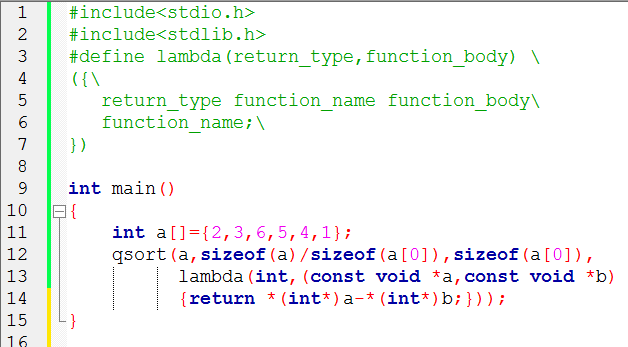
内嵌表达式与嵌套函数

如上面的max宏，内嵌表达式用于将若干语句整合成一个表达式，并返回最后一条语句的值。如果最后的语句为if while for等，则返回void

内嵌表达式一般也用在宏中，其中它还要去其他的特性结合才会更加方便

通过内嵌表达式与宏，配合类型推导，我们可以模拟一部分的泛型编程。

这里有一个有趣的例子，我们甚至可以通过内嵌表达式来模拟一个lambda表达式



该例中，我们定义了一个lambda宏，该宏的作用是创建一个函数，并返回他。我们能如此定义全依赖于gcc支持嵌套函数，即在一个函数中可以定义另一个函数。由内嵌表达式，我们返回了创建的函数的函数指针。由于作用域，我们也不必担心lambda的使用会造成名字冲突。

本地标签

通常来说，我们可能需要在某一条件下即返回某一值，例如我们定义一个函数如果条件1成立，则返回某个值，如果条件2成立，则返回另一个值。这样在内嵌表达式中会有限制。该例子当然有其他方法解决，但是总会遇到无法解决的时候。

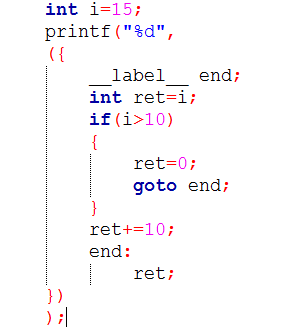
如果可以使用跳转，即goto直接跳转到最后一条语句，似乎可以解决这个问题。

但是内嵌表达式一般用于宏，在同一函数中可能多次使用，这样会造成标签冲突。

本地标签就用与解决这个问题，该标签只在该作用域内有效。

关键字 \_\_label\_\_ 用于定义本地标签。并且只能在内嵌表达式开头定义，即({的后面。

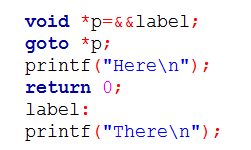
例如



即使该表达式定义为宏，也不会因为标签冲突而编译错误

标签值

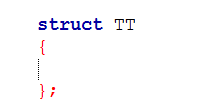
gcc支持将标签看成是一个值



也可以将其存在一个数组中，只需在标签名前加上&&

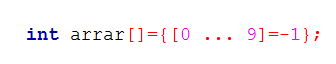
无成员结构体

gcc中可以定义一个没有成员的结构体，该结构体的字节长度为0



范围初始化

gcc中可以对一个数组进行范围初始化



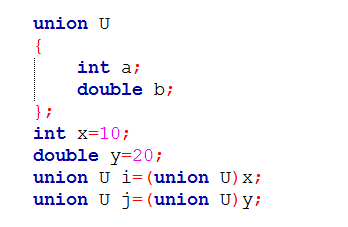
该表达式将0-9十个元素初始化为-1.

与之类似的，case也可以进行范围化 如 case 0 ... 9:

表示0-9均可匹配该项

union强制类型转换

可以将一个类型强制转换成union类型



0b前缀

类似于0x，0b表面该数字是二进制表示

内嵌汇编

gcc的内嵌汇编使用asm或者\_\_asm\_\_ 前者需要使用gnu标准

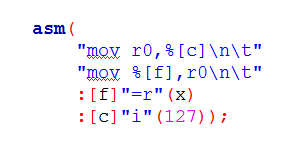
asm [volatile] ( AssemblerTemplate

: OutputOperands

: InputOperands

: Clobbers)

例如：（ARM汇编为例）



前面的字符串给出汇编模板，第一个冒号之后给出输出列表，汇编模板中的%[name] 由后面的 OutputOperands InputOperands 中的 [ ] 给出定义，例如%[f]即x ，“ ”之间的参数后续在讲述。

老式的规则中使用%0 ，%1 ...来表示参数 顺序按照汇编模板之后的 OutputOperands InputOperands 给出的顺序排列。

下面讲述””之间的参数

 i ： 立即数   例如： [changliang] "I" (120)， 表示把120作为立即数的方式传递进去。  
 m：任何有效内存 例如： [nc] "m" (a) , 表示把变量a的地址传递到内嵌汇编中，

str r0, %[nc], 就表示把r0的值存入变量a。  
 r： 通常的寄存器（r0-r15)。最常用。

上面这些限制性字符如果不带修饰符的话，表示是只读操作数，只能用于输入操作数列表中。对于输出操作数列表中的操作数我们需要加上相应的写权限。常用修饰符包括如下：

 = 只写操作数，通常用于所有的输出操作数。  
 + 读写操作。  
 & 只用作输出的寄存器。（表示不能与输入寄存器用同样一个寄存器）

最后给出Clobbers列表，即破坏列表

 一般就是一个“memory"，或者寄存器列表。如果在汇编代码中修改了内存中的值，我们就需要在破坏列表中加上"memory"字段，告诉编译器，我们修改了内存中的值。如果在汇编代码中修改了某些寄存器，那么我们就需要把在代码中所修改过的寄存器列在破坏列表中。

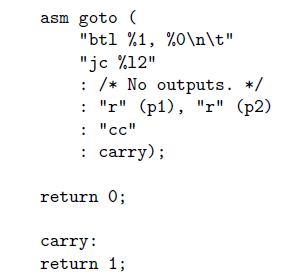
另外还有”cc” ，该参数表示汇编修改了标志寄存器，例如ARM中的CPSR寄存器

值得一提的是，如果某些列表为空，我们可以不写，但必须保留冒号 :

如果是后面的列表，则可以直接省略，如例子中就省掉了Clobbers列表，

如果省掉输出列表但是需要填写输入列表，那么输出列表之前的:需要保留

gcc还拓展了一个列表，加在破坏列表后面，用于跳转



具体查看gcc手册