将对以下关键字使用场合和特点进行讲解，引用自《C 语言深度剖析》

auto	缺省，不用管
register	寄存器变量，适用于多次连续使用的变量，但修饰变量时不一定管用。 用的时候要求： 1. 变量为单个值，长度小于等于整型长度； 2. 对该变量无法取址，由于不再内存中； 3. 该变量只能是局部变量；
Static	静态全局变量 修饰变量时，变量生命周期和全局变量一样长，但此时只能在文件或者函数内使用；修饰函数时，函数仅能在本文件中使用。
Const	定义 const 只读变量，具有不可变性。 编译器通常不为普通 const 只读变量分配存储空间，而是将它们保存在符号表中，这使得它成为一个编译期间的值，没有了存储与读内存的操作，使得它的效率也很高。 const 并不能把变量变成常量，在一个符号前面加上 const 限定符只是表示这个符号不能被赋值，但也不能防止通过程序的内部方法来修改这个值，const 最有用之处就是用它来限定函数的形参，这样该函数将不会修改实参指针所指的数据，但其他的函数却可能会修改它，这是 const 最一般的用法。
volatile	C 语言关键字 volatile（注意它是用来修饰变量而不是上面介绍的 volatile）表明某个变量的值可能在外部被改变，因此对这些变量的存取不能缓存到寄存器，每次使用时需要重新存取。该关键字在多线程环境下经常使用，因为在编写多线程的程序时，同一个变量可能被多个线程修改，而程序通过该变量同步各个线程。 volatile 提醒编译器它后面所定义的变量随时都有可能改变，因此编译后的程序每次需要存储或读取这个变量的时候，都会直接从变量地址中读取数据。如果没有 volatile 关键字，则编译器可能优化读取和存储，可能暂时使用寄存器中的值，如果这个变量由别的程序更新了的话，将出现不一致的现象。
Extern	外部引用的变量或者函数

1.1，最宽恒大量的关键字----auto

auto：它很宽恒大量的，你就当它不存在吧。编译器在默认的缺省情况下，所有变量都是 auto 的。

1.2，最快的关键字---- register

register：这个关键字请求编译器**尽可能**的将变量存在 CPU 内部寄存器中而不是通过内存寻址访问以提高效率。注意是**尽可能，不是绝对**。你想想，一个 CPU 的寄存器也就那么几个或几十个，你要是定义了很多很多 register 变量，它累死也可能不能全部把这些变量放入寄存器吧，轮也可能轮不到你。

1.2.1，皇帝身边的小太监----寄存器

不知道什么是寄存器？那见过太监没有？没有？其实我也没有。没见过不要紧，见过就麻烦大了。^_^，大家都看过古装戏，那些皇帝们要阅读奏章的时候，大臣总是先将奏章交给皇帝旁边的小太监，小太监呢再交给皇帝同志处理。这个小太监只是个中转站，并无别的功能。

好，那我们再联想到我们的 CPU。CPU 不就是我们的皇帝同志么？大臣就相当于我们的内存，数据从他这拿出来。那小太监就是我们的寄存器了（这里先不考虑 CPU 的高速缓存区）。数据从内存里拿出来先放到寄存器，然后 CPU 再从寄存器里读取数据来处理，处理完后同样把数据通过寄存器存放到内存里，CPU 不直接和内存打交道。这里要说明的一点是:小太监是主动的从大臣手里接过奏章，然后主动的交给皇帝同志，但寄存器没这么自觉，它从不主动干什么事。一个皇帝可能有好些小太监，那么一个 CPU 也可以有很多寄存器，不同型号的 CPU 拥有寄存器的数量不一样。

为啥要这么麻烦啊？速度！就是因为速度。寄存器其实就是一块一块小的存储空间，只不过其存取速度要比内存快得多。进水楼台先得月嘛，它离 CPU 很近，CPU 一伸手就拿到数据了，比在那么大的一块内存里去寻找某个地址上的数据是不是快多了？那有人问既然它速度那么快，那我们的内存硬盘都改成寄存器得了呗。我要说的是：你真有钱！

1.2.2，使用 register 修饰符的注意点

虽然寄存器的速度非常快，但是使用 register 修饰符也有些限制的：register 变量必须是能被 CPU 寄存器所接受的类型。意味着 register 变量必须是一个单个的值，并且其长度应小于或等于整型的长度。 而且 register 变量可能不存放在内存中，所以不能用取址运算符“&”来获取 register 变量的地址。

1.3，最名不符实的关键字----static

不要误以为关键字 static 很安静，其实它一点也不安静。这个关键字在 C 语言里主要有两个作用，C++对它进行了扩展。

1.3.1，修饰变量

第一个作用：修饰变量。变量又分为局部和全局变量，但它们都存在内存的静态区。

静态全局变量，作用域**仅限于变量被定义的文件中**，其他文件即使用 extern 声明也没法使用他。准确地说作用域是从定义之处开始，到文件结尾处结束，在定义之处前面的那些代码行也不能使用它。想要使用就得在前面再加 extern ***。恶心吧？要想不恶心，很简单，直接在文件顶端定义不就得了。

静态局部变量，在函数体里面定义的，就只能在这个函数里用了，同一个文档中的其他函数也用不了。由于被 static 修饰的变量总是存在内存的静态区，所以即使这个函数运行结束，这个静态变量的值还是不会被销毁，函数下次使用时仍然能用到这个值。

1.12，最易变的关键字----volatile

volatile 是易变的、不稳定的意思。很多人根本就没见过这个关键字，不知道它的存在。也有很多程序员知道它的存在，但从来没用过它。我对它有种“杨家有女初长成,养在深闺人未识” 的感觉。

volatile 关键字和 const 一样是一种类型修饰符，用它修饰的变量表示可以被某些编译器未知的因素更改，比如操作系统、硬件或者其它线程等。遇到这个关键字声明的变量，编译器对访问该变量的代码就不再进行优化，从而可以提供对特殊地址的稳定访问。

先看看下面的例子：

```
int i=10;
int j = i；//(1)语句
int k = i；//(2)语句
```

这时候编译器对代码进行优化，因为在（1）、（2）两条语句中，i 没有被用作左值。这时候编译器认为 i 的值没有发生改变，所以在（1）语句时从内存中取出 i 的值赋给 j 之后，这个值并没有被丢掉，而是在（2）语句时继续用这个值给 k 赋值。编译器不会生成出汇编代码重新从内存里取 i 的值，这样提高了效率。但要注意：（1）、（2）语句之间 i 没有被用作左值才行。

再看另一个例子：

```
volatile   int i=10;
int j = i；//(3)语句
int k = i；//(4)语句
```

volatile 关键字告诉编译器 i 是随时可能发生变化的，每次使用它的时候必须从内存中取出 i 的值，因而编译器生成的汇编代码会重新从 i 的地址处读取数据放在 k 中。

这样看来，如果 i 是一个寄存器变量或者表示一个端口数据或者是多个线程的共享数据，就容易出错，所以说 volatile 可以保证对特殊地址的稳定访问。

但是注意：在 VC++6.0 中，一般 Debug 模式没有进行代码优化，所以这个关键字的作用有可能看不出来。你可以同时生成 Debug 版和 Release 版的程序做个测试。

留一个问题：const volatile int i=10；这行代码有没有问题？如果没有，那 i 到底是什么属性？

1.13，最会带帽子的关键字----extern

extern，外面的、外来的意思。那它有什么作用呢？举个例子：假设你在大街上看到一个黑皮肤绿眼睛红头发的美女（外星人？）或者帅哥。你的第一反应就是这人不是国产的。extern 就相当于他们的这些区别于中国人的特性。extern 可以置于变量或者函数前，以标示变量或者函数的定义在别的文件中，下面的代码用到的这些变量或函数是外来的，不是本文件定义的，提示编译器遇到此变量和函数时在其他模块中寻找其定义。就好比在本文件中给这些外来的变量或函数带了顶帽子，告诉本文件中所有代码，这些家伙不是土著。那你想想 extern 修饰的变量或函数是定义还是声明？