# 跟vczh看实例学编译原理——二：词法分析与前置后置条件

文章中引用的代码均来自<https://github.com/vczh/tinymoe>。

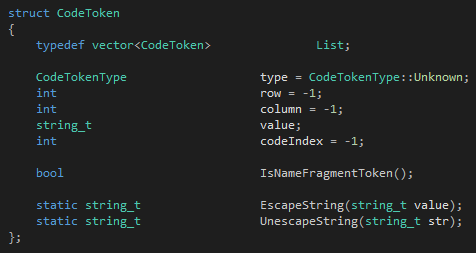
实现Tinymoe的第一步自然是一个词法分析器。词法分析其所作的事情很简单，就是把一份代码分割成若干个token，记录下他们所在文件的位置，以及丢掉不必要的信息。但是Tinymoe是一个按行分割的语言，自然token列表也就是二维的，第一维是行，第二维是每一行的token。在继续讲词法分析器之前，先看看Tinymoe包含多少token：

* 符号：(、)、,、:、&、+、-、\*、/、\、%、<、>、<=、>=、=、<>
* 关键字：module、using、phrase、sentence、block、symbol、type、cps、category、expression、argument、assignable、list、end、and、or、not
* 数字：123、456.789
* 字符串：”abc\r\ndef”
* 标识符：tinymoe
* 注释：-- this is a comment

Tinymoe对于token有一个特殊的规定，就是字符串和注释都是单行的。因此如果一个字符串在没有结束之前就遇到了换行，那么这种写法定义为你遇到了一个没有右双引号的字符串，需要报个错，然后下一行就不是这个字符串的内容了。

一个词法分析器所需要做的事情，就是把一个字符串分解成描述此法的数据结构。既然上面已经说到Tinymoe的token列表是二维的，因此数据结构肯定会体现这个观点。Tinymoe的词法分析器代码可以在这里找到：<https://github.com/vczh/tinymoe/blob/master/Development/Source/Compiler/TinymoeLexicalAnalyzer.h>。

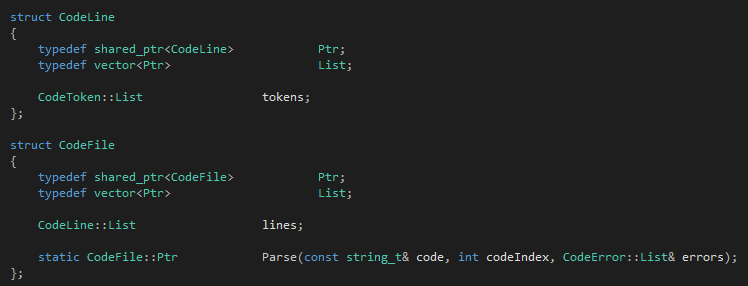
首先是token：



CodeTokenType是一个枚举类型，标记一个token的类型。这个类型比较细化，每一个关键字有自己的类型，每一个符号也有自己的类型，剩下的按种类来分。我们可以看到token需要记录的最关键的东西只有三个：类型、内容和代码位置。在token记录代码位置是十分重要的，正确地记录代码位置可以让你能够报告带位置的错误、从语法树的节点还原成代码位置、甚至在调试的时候可以把指令也换成位置。

这里需要提到的是，string\_t是一个typedef，具体的声明可以在这里看到：<https://github.com/vczh/tinymoe/blob/master/Development/Source/TinymoeSTL.h。Tinymoe>是完全由标准的C++11和STL写成的，但是为了适应不同情况的需要，Tinymoe分为code page awared版本和unicode版本。如果编译Tinymoe代码的时候声明了全局的宏UNICODE\_TINYMOE的话，那Tinymoe所有的字符处理将使用wchar\_t，否则使用char。char的类型和Tinymoe编译器在运行的时候操作系统当前的code page是绑定的。所以这里会有类似string\_t啊、ifstream\_t啊、char\_t等类型，会在不同的编译选项的影响下指向不同的STL类型或者原生的C++类型。github上的VC++2013工程使用的是wchar\_t的版本，所以string\_t就是std::wstring。

Tinymoe的词法分析器除了token的类型以外，肯定还需要定义整个文件结构在词法分析后的结果：



这个数据结构体现了“Tinymoe的token列表是二维的”的这个观点。一个文件会被词法分析器处理成一个shared\_ptr<CodeFIle>对象，CodeFile::lines记录了所有非空的行，CodeLine::tokens记录了该行的所有token。