编译原理课程实验报告

**实验二 基于BitMiniCC的词法分析器设计**

指导教师：王贵珍老师

班 级：07111507

学 号：1120151880

姓 名：廖汉龙

邮 箱：[liamliaohl@gmail.com](mailto:liamliaohl@gmail.com)

2018 年 4 月 13 日

**一、实验目的**

ACM/IEEE-CS 计算学科 2013 新教程（简称 CS2013 教程）中，根据计算学科的迅速发展和变化，根据学校的定位和培养目标，亦强调计算机学科学生除了掌握本学科领域重要的知识和技能，还要有领域拓宽和终身学习的能力。 CS2013教程比 CC2001 对计算学科涉及的知识领域的凝练在深度和广度上都有较大变化。它将计算学科划分为 18 个知识领域，编译原理与设计课程涉及的知识直接关联到 ACM/IEEE-CS2013 许多知识领域，诸如算法和复杂性、计算科学、架构与组织、 系统的基础、 离散结构、 编程语言、 并行和分布式计算、 软件工程等。因此若本课程教学计划仍然沿用传统的教学模式而不进行改革， 将难以支撑课程改革和学科发展的需求，难以胜任研究型大学的培养目标。编译原理是计算机科学与技术专业的主干课程之一， 在计算机本科教学中占有重要地位。编译原理课程具有较强的理论性和实践性，但在教学过程中容易偏重于理论介绍而忽视实验环节，使学生在学习过程中普遍感到内容抽象，不易理解。

该实验的目的是通过实践环节深入理解与编译实现有关的形式语言理论基本概念， 掌握编译程序构造的一般原理、基本设计方法和主要实现技术，并通过运用自动机理论解决实际问题，从问题定义、分析、建立数学模型和编码的整个实践活动中逐步提高软件设计开发的能力。

**二、实验内容**

该实验以 C 语言作为源语言，构建 C 语言的词法分析器，对于给定的测试程序，输出 XML 格式的属性字符流。词法分析器的构建按照 C 语言的词法规则进行。

本实验实现的内容有：

1.阅读 BITMiniCCompilier 框架代码，熟练掌握和应用框架。

2.自主编写代码（Java/C/C++/C#/Python）实现C语言的词法分析。输入为C语言程序或者经过框架预处理代码处理过的 .pp.c文件（预处理代码会将代码中的注释部分去掉，并且改变代码格式为标准的C语言格式）。

3.对词法分析其进行测试。

以下是C11标准给出的文法规则（子集实现）：

**（1）常数类型**

常数类型包含 - 整型常数，实型常数，8进制常数，16进制常数，字符，字符串等等类型

C11标准定义文法(子集实现)

* **整形常量**

|  |
| --- |
| In         ->        Dec | Oct | Hex  Dec       ->        nd | Dec+d  Oct        ->       0 | Oct+od  Hex        ->       Hex-pre+hd | Hex+hd  hd        ->       0x | 0X  nd         ->       1|2|3|4|5|6|7|8|9  od         ->       0|1|2|3|4|5|6|7  hd         ->       0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f|A|B|C|D|E|F |

* **字符常量**

|  |
| --- |
| c-char-sequence    ->    c-char | c-char-sequence+c-char  c-char            ->    除了 ', \, \n 之外的任何字符 | 转义字符 |

* **字符串常量**

|  |
| --- |
| string-literal:     ->     s-char-sequence  s-char-sequence ->     s-char | s-char-sequence+s-char  s-char             ->     除了 ", \, \n 之外的任何字符 | 转义 字符 |

**（2）标识符 / 关键字类型**

标识符包含各类名字的表示，比如变量名，数组名，函数名，文件名.

因为标识符和关键字的内聚性比较强，所以在词法分析器的具体实现中，我采用了将标识符和关键字混合分类判断的方式:

对每一个识别出来的标识符进行二次加工处理，判断是否是关键字，从而对这两种类型在词法分析的阶段中进行区分

C11标准定义文法(子集实现)

|  |
| --- |
| ID        ->    ID-nd | ID+ID-nd | ID+d  ID-nd   ->    nd  nd        ->    \_|a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|  v|w|x|y|z|A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|  Q|R|S|T|U|V|W|X|Y|Z  d         ->    0|1|2|3|4|5|6|7|8|9 |

**（3）运算符**

表示程序中的算数运算，逻辑运算，字符，串操作等运算的确定字符(串)

|  |
| --- |
| [], (), ->, .  !, -, ++, --, &, \*, +, -, ~  /, %, <<, >>, <, >, <=, >=, ==, !=, ^, |, &&, ||  ?, :, ;, ...  =, \*=, /=, %=, +=, -=, <<=, >>=, &=, ^=, |=  ,, #, ## |

（4）分隔符

逗号，分号，括号，单引号，双引号等等

|  |
| --- |
| ,, ;, {, } |

**三、实验原理与实现步骤**

**1.阅读框架代码**

框架代码主要的实现部分在.\src\bit\minisys\minicc中的三段代码，BITMiniCC.java， BITMiniCCompiler.java, BITMiniCCCfg.java。其中，BITMiniCC为程序运行的主程序，BITMIniCCompiler.java为编译器的各个阶段的运行主控程序，根据不同的选择配置，运行不同的程序。

配置程序为 ./run/configximl

查看BITMiniCCompiler.java中的部分代码：

|  |
| --- |
|  |

上述代码中有三个条件判断语句：

如果当前阶段的 skip == false，表示当前步骤不跳过，判断 当前阶段的 type, 如果 type == java，判断 path, 如果path == “”,使用默认路径，否则需要使用用户自定义的路径。

如果当前的type == python，则运行函数runPy（）函数，如果type为其他类型，就运行run函数。

下面以python为例，查看runPy（）函数：

|  |
| --- |
|  |

runPy（）函数调用了Jython中的方法，建立了PythonInterpreter对象实例pyi, PythonInterpreter为Jython的python解释器对象。在run()函数中是通过新建了进程，运行可执行文件实现的，如果不使用内置的解释器，则可以使用python第三方库pyinstaller对python代码打包成exe文件。

运行框架代码方法：

在./run的config.xml文件中，查看框架的配置：

|  |
| --- |
|  |

这些配置对应的功能在BITMiniCCompiler.java中实现，在使用框架时只要修改即可。

例如，使用可执行文件：将skip 改为 false，将type 改为 binary, path是可执行文件的绝对路径。如果是python， 则在path 这一项，使用【 python + 绝对路径】，python是系统路径中的python路径。

另外需要说明的是，框架中的路径的input文件在./run/input中，所以应该在./run中建立input文件，将需要测试的代码存在这个文件夹下，而不是存放在根目录文件夹input下。

**2.实验实现步骤**

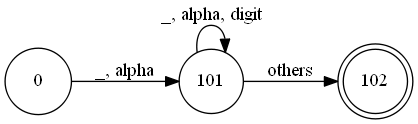
本实验使用了框架中的预处理代码，预处理实现了去除注释以及将代码格式转化为标准的C语言代码格式，比如a=6;会被处理为a = 6, 去除了头文件代码。

**（1）.DFA设计**

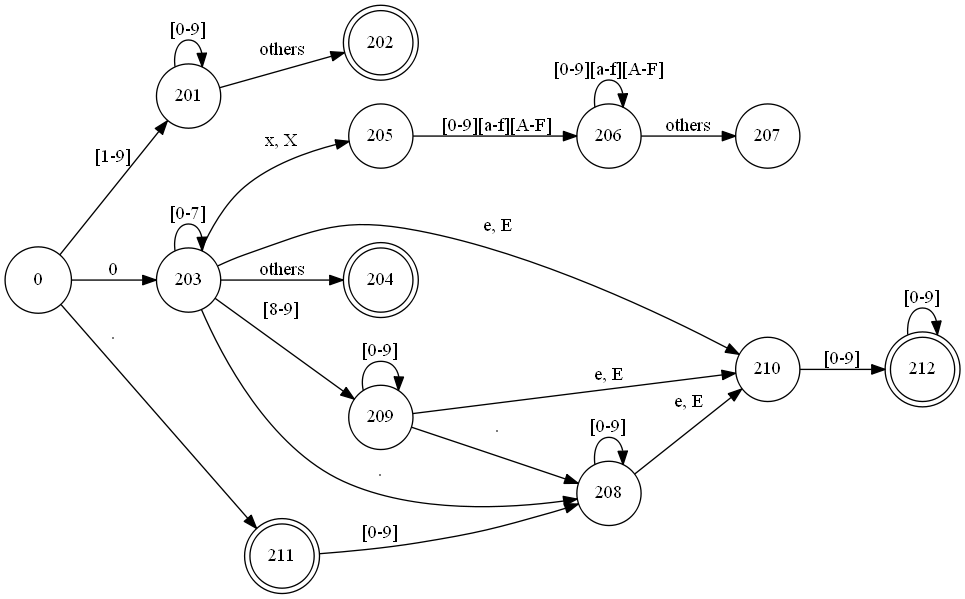
根据C语言的C11标准，将识别的接受态分为常整数（十进制，八进制，十六进制），浮点数（十进制），字符（串），关键字，运算符，分隔符6类。

考虑到程序的运行效率和代码实现的难易程度，采用程序中心法，python语言。

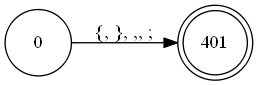
* 字符（串），关键字的DFA状态图



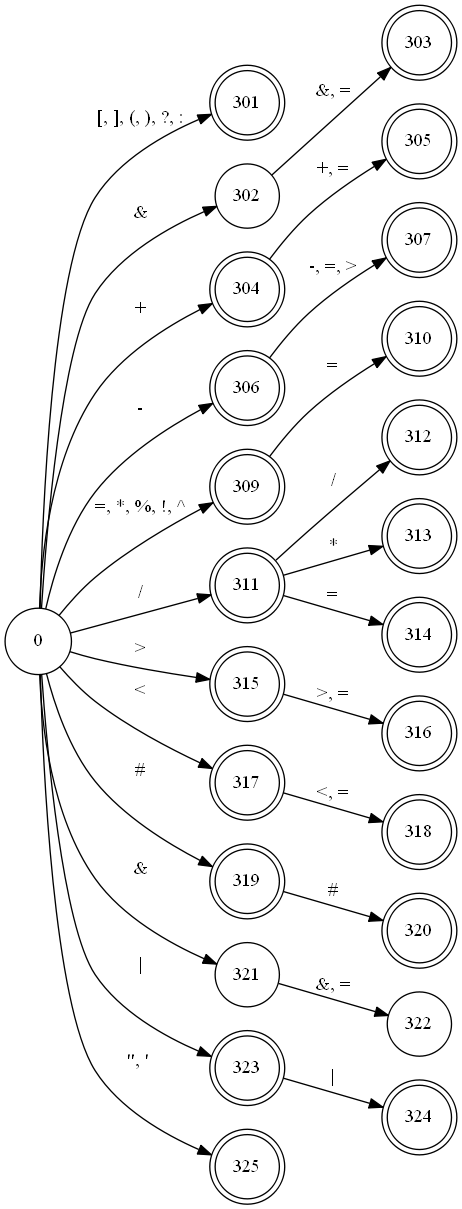
* 常整数和浮点数的DFA状态图：



* 分隔符的DFA状态图：

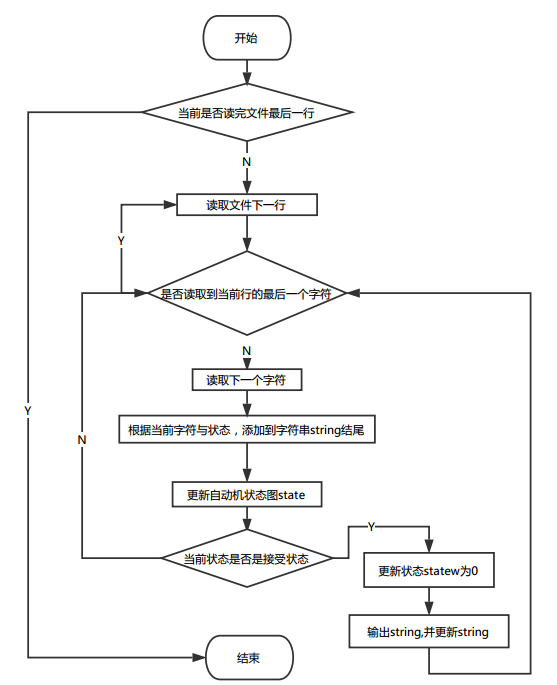


* 识别运算符的DFA状态图：



**3.程序思路与实现**

**1)程序实现的流程图如下图所示**：

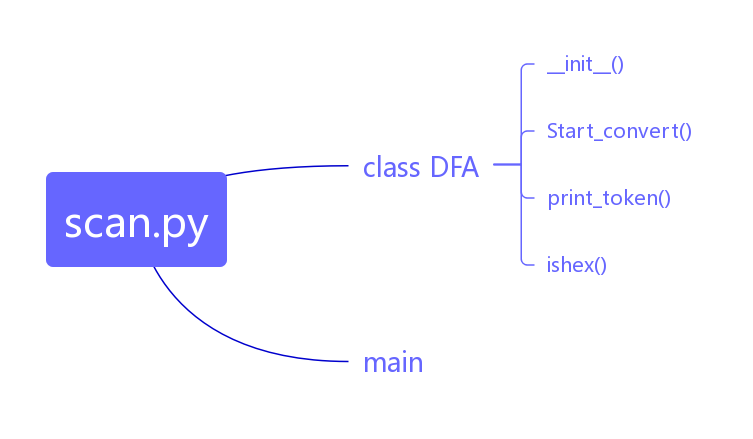


在流程图中，需要注意的是有些状态的识别过程中，需要回退一个字符，比如识别字符串时，当识别到当前是非字母，数字、下划线字符时，先回退一个字符，才更新状态为0.

**2)代码实现实例：**

由于python的语言特性，没有其他语言中常用的swith…case语句，如果需要实现类似功能，需要使用字典的方法，但是这会引起效率降低。所以程序采用了if…else的判断方式。

代码的总体结构如下：



在main()中写入需要读入和输出的文件，调用DFA类对象；在DFA中，初始化函数\_\_init\_\_（）初始化和定义了输入输出文件，一些全局的变量，如错误信息；Start\_convert()是主要的函数，进行字符的读取与状态的变换；print\_token()是输出函数，ishex()是判断当前字符是否是十六进制数字组成字符的判断函数。

程序的示例如下：

|  |
| --- |
|  |

如上图代码所示，是类DFA中start\_convert()函数的部分代码，定义了两个循环，分别判断行数与字符数。定义了状态的变量state，当前已经读取的字符串string。如果当前不是接受状态，则添加当前字符，更新状态，否则就打印字符串，更新字符串，更新状态为0，进行下一次字符的读取，如下图所示：

|  |
| --- |
|  |

**四、实验测试结果**

**1.实验结果**

将配置文件config.xml中的scan阶段的type 设置为 python, 将路径设置为 python3 ./scan,py。

运行的测试代码：

|  |
| --- |
|  |

运行框架代码如下图所示：

|  |
| --- |
|  |

查看生成的test.token.xml文件（部分）

|  |
| --- |
|  |

从以上的截图发现，识别字符串，字符，十进制等数字是没有问题的。

下面对一些浮点数以及八进制，十六进制数字进行测试：

|  |
| --- |
|  |

得到的token.xml文件为：

|  |
| --- |
|  |

如图所示，对于不同进制的数字，都可以识别，并且区分出浮点数与整常数。

**2实验总结与思考**

目前程序仍然存在一些问题：

比如程序中还没有添加错误的处理机制，如果当前字符是不能识别的，比如出现违法的常数，此时会转到一个错误状态，并且又重新回到初始0状态，但是这些是透明的，没有报错的机制。

程序中肯定还存在大量的bug还未发现和解决，有待进一步改进。

实现的词法分析C语言词法的子集，并没有实现全部的识别功能，如“

* 未识别十六进制浮点数
* 未识别转义字符串
* 未识别带后坠的常数的识别，如12L等
* 未识别一些极少使用到的运算符，如>>=, <<=等

本次实验，是使用python实现的，但是众所周知，python语言的效率是非常低的，只是为了实现课程的实验而已，在实际过程中是无实用性。

通过这个实验下来，我已经基本了解了如何使用DFA进行词法分析，对理论有了非常直观的认识，掌握也更加扎实了，为后续的实验打下了坚实的基础。同时非常感谢BITMiniCC框架，可以让我们在紧张的时间内，有很好的实践的效果。