PL0编译器开发说明文档

1. 编译器实现原理

本PL0编译器只需生成PCode指令集，采用一遍扫描源文件的方法，通过C++实现。

1.1词法分析

1.1.1词法分析器的实现方式

词法分析器的实现方式主要有两种：手工编码实现和自动机实现。其中自动机的实现方法是通过编写Lex源程序，并通过ParseGenerator来生成自动机实现的词法分析器。

1.1.2通过Lex实现的词法分析器

1.1.2.1 Lex简介

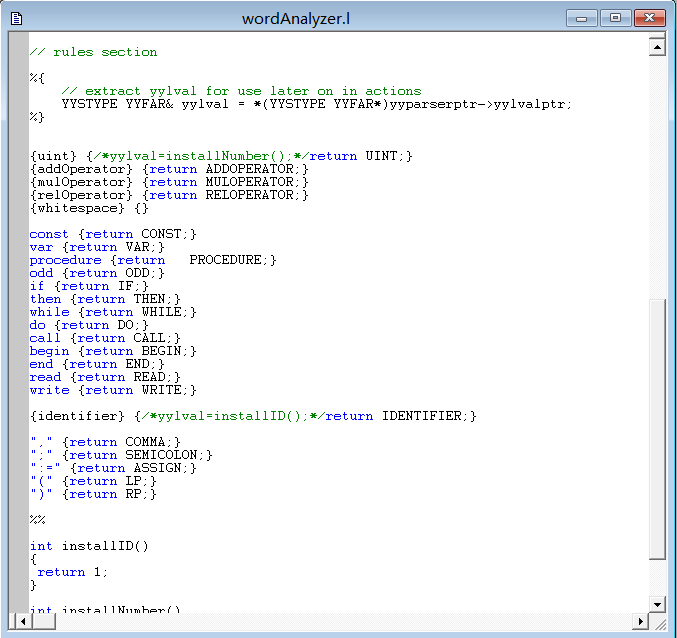
Lex 是一种生成扫描器的工具。扫描器是一种识别文本中的词汇模式的程序。这些词汇模式（或者正则表达式）在一种特殊的句子结构中定义。

一种匹配的正则表达式可能会包含相关的动作。这一动作可能还包括返回一个标记。 当 Lex 接收到文件或文本形式的输入时，它试图将文本与常规表达式进行匹配。它一次读入一个输入字符，直到找到一个匹配的模式。如果能够找到一个匹配的模式，Lex 就执行相关的动作（可能包括返回一个标记）。另一方面，如果没有可以匹配的常规表达式，将会停止进一步的处理，Lex 将显示一个错误消息。

Lex 和 C 是强耦合的。一个 .lex 文件（Lex 文件具有 .lex 的扩展名）通过 lex 公用程序来传递，并生成 C 的输出文件。这些文件被编译为词法分析器的可执行版本。

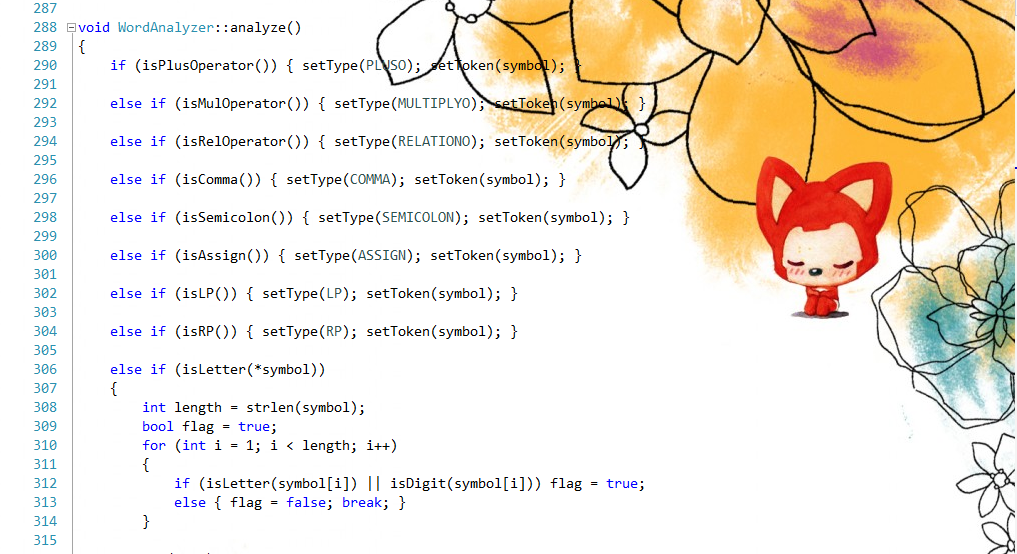
Lex文法这里不详细介绍，主要参考自IBM开发文档：<http://www.ibm.com/developerworks/cn/linux/sdk/lex/>。

* + - 1. PL0文法的词法分析器Lex源程序



1.1.3手工实现的词法分析器

通过为每一种词法单元编写一个函数来实现词法单元的识别。所封装的数据中type表示当前识别的词法单元的类型，symbol表示当前读入的字符串，symbols为vector<string>类型的字符串缓存，token表示若扫描的字符串合法则保存的词法单元，cur表示词法分析器当前读入的字符串的索引值。主要代码实现如图：





1.2语法分析

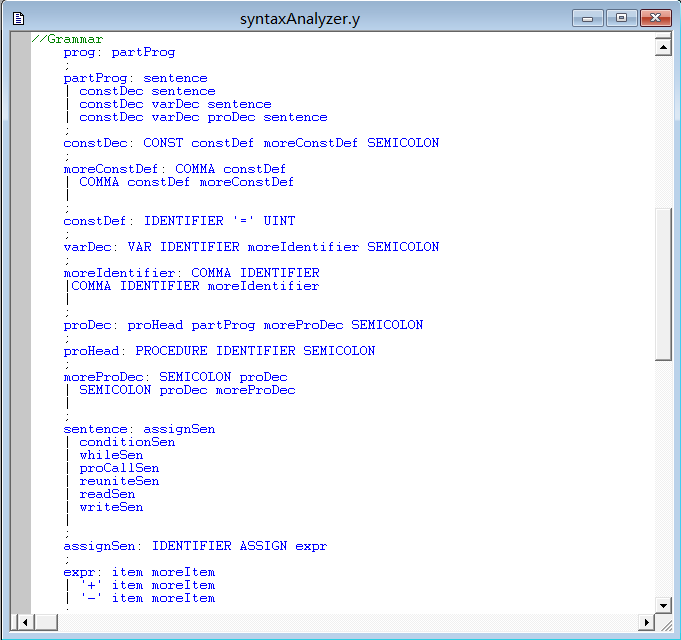
1.2.1语法分析器的实现方式

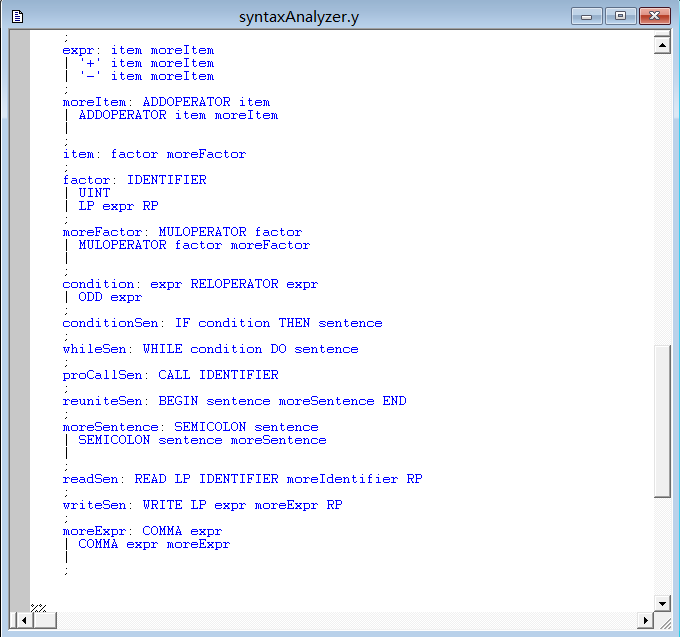
1.2.2通过Yacc实现的语法分析器

1.2.2.1 Yacc简介

Yacc 代表 Yet Another Compiler Compiler。 Yacc 的 GNU 版叫做 Bison。它是一种工具，将任何一种编程语言的所有语法翻译成针对此种语言的 Yacc 语 法解析器。它用巴科斯范式(BNF, Backus Naur Form)来书写。按照惯例，Yacc 文件有 .y 后缀。

1.2.2.2 PL0文法的词法分析器Lex源程序

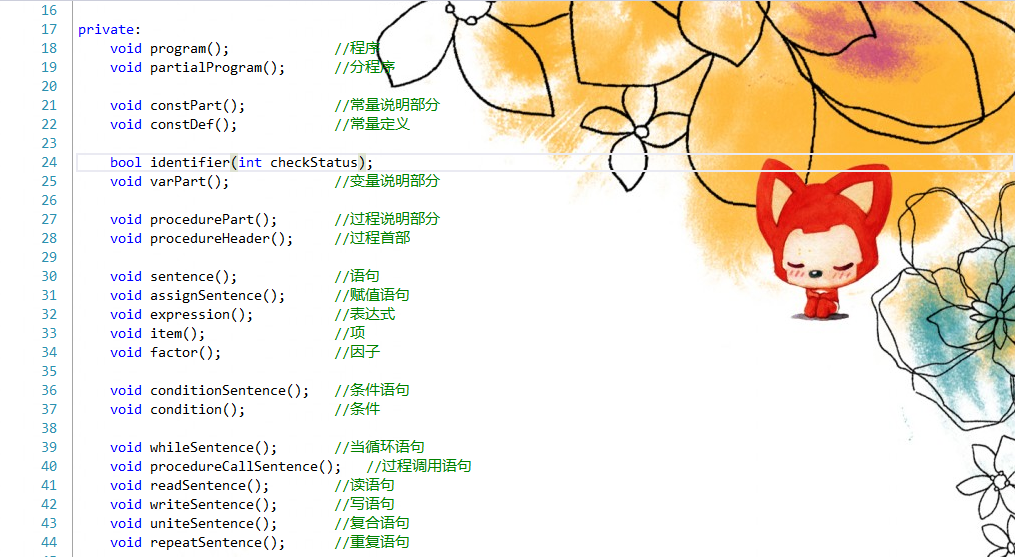




1.2.3自顶向下的语法分析器实现

对于自顶向下实现的语法分析器，要求文法不能是左递归的，否则会导致无限递归。而所给的PL0文法已满足这一要求。

自顶向下的语法分析器的实现思想是：为每一个产生式的左边项编写一个递归子程序。在语法分析器中，调用了3.1.3节实现的词法分析器来扫描源程序。由于语法分析器的主要实现代码较多，这里只给出主要函数的实现定义。



1.3语义分析

PL0编译系统的语义分析部分主要有一下几个部分组成：

1. 声明部分的语义分析。这一部分包括了变量声明部分、常量声明部分以及过程声明部分。
2. 表达式的语义分析。
3. 语句部分的语义分析。语句部分的语义分析又包括赋值语句、条件语句、循环语句、过程调用语句、读语句和写语句。

1.4符号表管理

由于采用的是一遍扫描生成中间代码的方式，故只要在语义分析这一部分进行符号表管理。本编译器为简单起见，采用栈式符号表实现。

1.4.1符号表条目组成

符号表条目的数据结构主要包括五个部分：

1. name，标识符的名字。
2. kind，标识符的种类。
3. value，标识符的值。若标识符表示常量，则此处为空。
4. level，标识符所在程序块的层次，主程序为第一层。
5. address，标识符的地址。实际表示的是标识符所在程序块内的偏移，初始偏移位置为3。

1.4.2符号表操作

在本编译系统的实现中，符号表由SymbolTableManager管理。其中对符号表的主要操作有：

1. 插入（insert）。每次识别一个合法的变量声明，则向栈顶插入一项符号表条目。
2. 查找（find）。在语义分析时，每次在语句部分遇见一个标识符，则在符号表中查找。若在符号表中找不到该标识符，则说明该标识符为声明。
3. 获得符号表条目（getTableEntry）。
4. 删除/出栈(pop)。当一个程序块扫描完成时，应当在符号表中将该程序块中的所有标识符对应的符号表条目删除。

1.5中间代码生成

PCode指令：

LIT 0 a

OPR 0 a

LOD 1 a

STO 1 a

CAL 1 a

INT 0 a

JMP 0 a

JPC 0 a

RED l a

WRT 0 0

1. 可执行文件说明

2.1编译运行环境

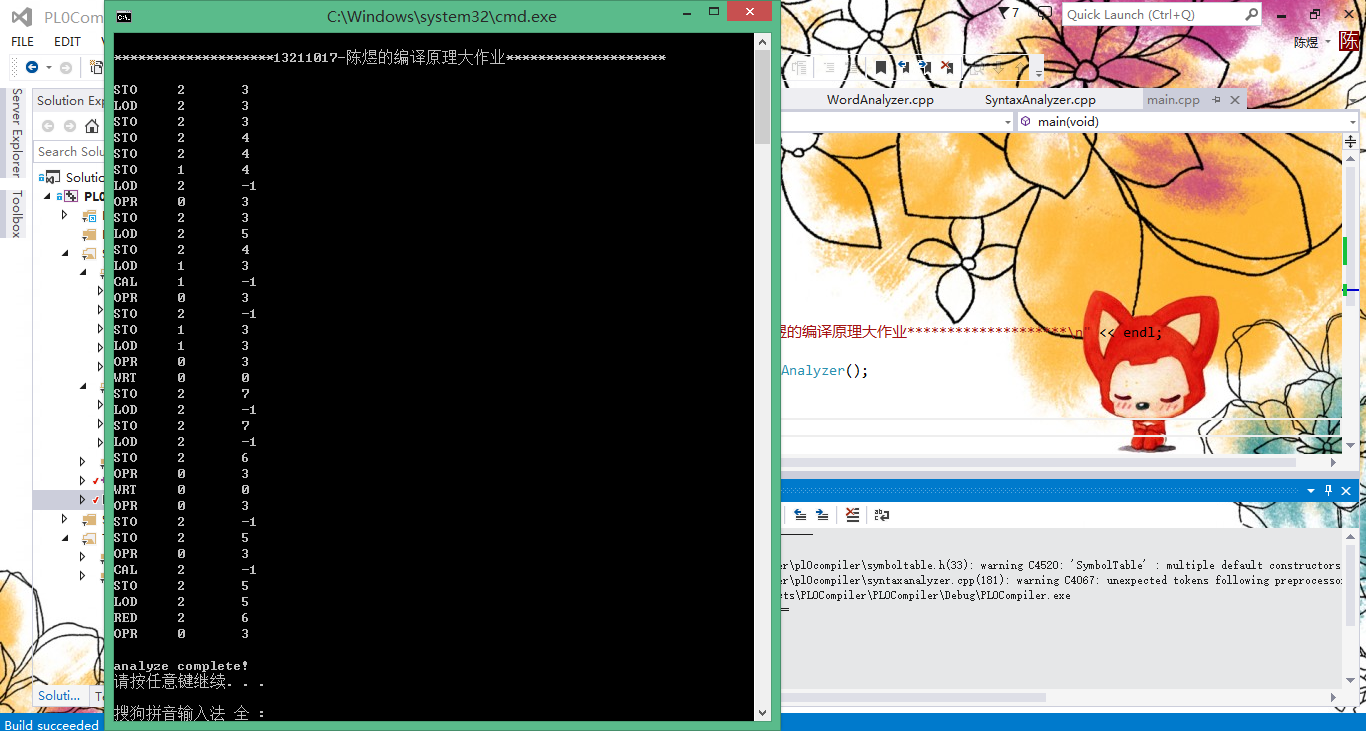
Windows环境下均可执行。

2.2可执行文件执行方法

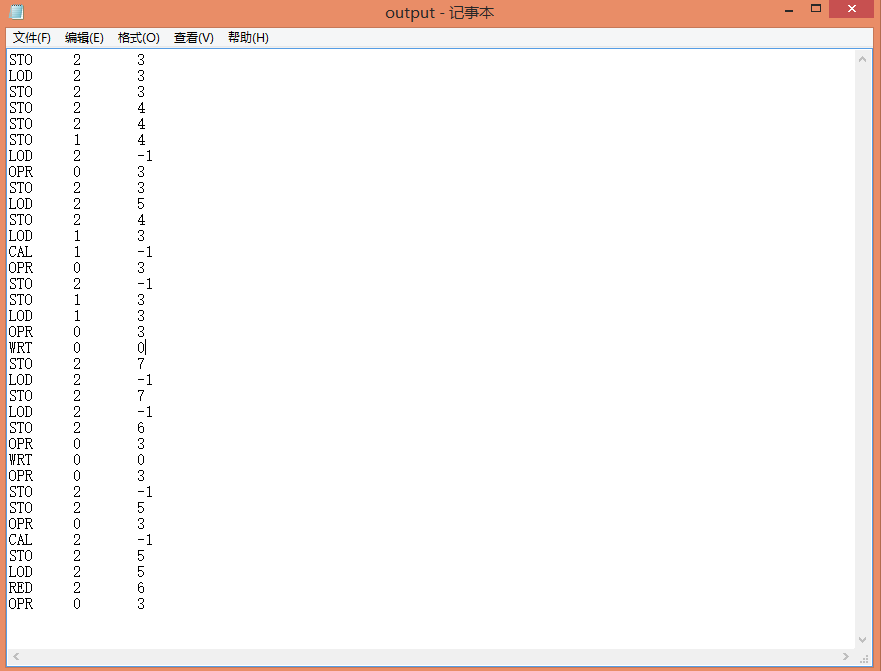
在可执行文件的debug目录下双击PL0Compiler.exe，然后在相同目录下的output.txt文件中即可查看生成的PCode中间代码。

2.3执行效果截图

1.调试版：



2.提交版：



1. 总结

本编译系统功能尚不完全，而且生成的部分PCode代码是错误的。不过词法分析、语法分析、语义分析和符号表的管理是完善的，整个编译系统的架构基本完成。以下为原因总结：

其实对这个编译系统挺感兴趣的。刚开始想尝试用自动机的方法来实现，但是由于错误地估计了用自动机实现的难度，以致于在花费了两个礼拜的实践之后正式宣告失败（而且后来发现邵老师要求的是用自顶向下的方法来实现）。然后，后来的时间就很紧张了。因此只花了一个多星期断断续续地从头开始写，在整个架构基本完工的时候，为了期末考试，本着“舍卒保车”的策略放弃了最后的完善过程。

尽管整个编译系统并不完善，但是自己在做这个编译系统的过程中，通过自己动手实践，弄清楚了编译器前端实现的原理和大多数的细节，对编译原理的理解更进一步。