1 搭建开发环境

本章介绍MKS lex&yacc 在windows下开发的环境搭建

lex&yacc有多个实现的版本，比如。。。我们后文中的lex和yacc，如果不加说明，均指的是MKS lex&yacc。

2 编译器知识速览

2.1 first语言

为了介绍语言和编译相关的概念，我们先引入一门简单的语言：first语言。

frist语言是一门脚本语言，这个语言的语句表现出如下特征：

（1）每一行为一句

（2）变量的定义和赋值

（左边为一条语句，右边为语句执行的结果）

a --> create variable a=0 （向系统定义一个变量a，赋予初始值0）

a = 5 --> create variable a=5 （向系统定义一个变量a，并赋值为5，覆盖原有的定义）

（3）计算表达式

1+2\*3 --> 7 （计算表达式，并返回结果）

1 --> 1 （计算表达式，并返回结果）

a --> 5 （计算表达式，并返回结果）

1+a\*2 --> 11 （计算表达式，并返回结果）

1+b\*2 --> error: no variable “b” （报告错误，表达式中引用了未定义的变量）

（4）if-then 条件执行

if a > 4 then b=5 --> create variable b=5 （if-then语句）

（5）打印print，可同时打印一个或多个表达式的值

print ( a ) --> 5 （打印表达式的值）

print ( a+1, a\*2, 3+2 ) --> 6， 10， 5 （打印表达式们的值）

2.2 形式化语言

2.2.1 何谓“语言”

语言是特定群体用于特定交流的工具。人与人之间进行交流的工具是自然语言，人与计算机之间进行交流的工具是编程语言。

语言的使用，有两步：语言的产生、语言的识别。

人与人之间的交流是双向的，一个人既要说，也要听，即要一边产生语言，还要一边识别语言。而人与计算机的交流则不一样。编程时，人产生计算机编程语言，即代码，计算机识别代码并作出回应（对于脚本语言，这里的“回应”是指执行代码；对于编译型语言，这里的“回应”是指生成目标代码）。这个过程中，人在说，计算机在听，角色固定。编译器的作用仅仅是识别语言。

为了研究语言，前人对语言进行了建模，即“形式化语言”。首先解决的问题就是：如何定义一门语言。

语言学家乔姆斯基从产生语言的角度定义了语言。他指出，定义一门语言，即定义它的产生规则，这些规则有很多称呼：文法规则（简称“文法”）、语法、规范、规则、grammar。我们下文中使用“文法”这个称呼。

通过一门语言的文法规则，能产生很多语句，这些语句均属于这门语言。由文法规则能产生的所有语句的集合，便称之为这门语言。判断一个语句是否属于一门语言，只要判断该语句是否能由该语言的文法规则所产生。

接下来的几节内容，我们要解决一个问题：定义first语言，或者说，写出它的文法规则。

文法规则，由词法规则和语法规则构成。

2.2.2 词法规则

相关术语

词法规范： lexical grammar / lexer grammar / lexture

词法分析： lexical analysis

词法分析器： lexer

词法符号： token

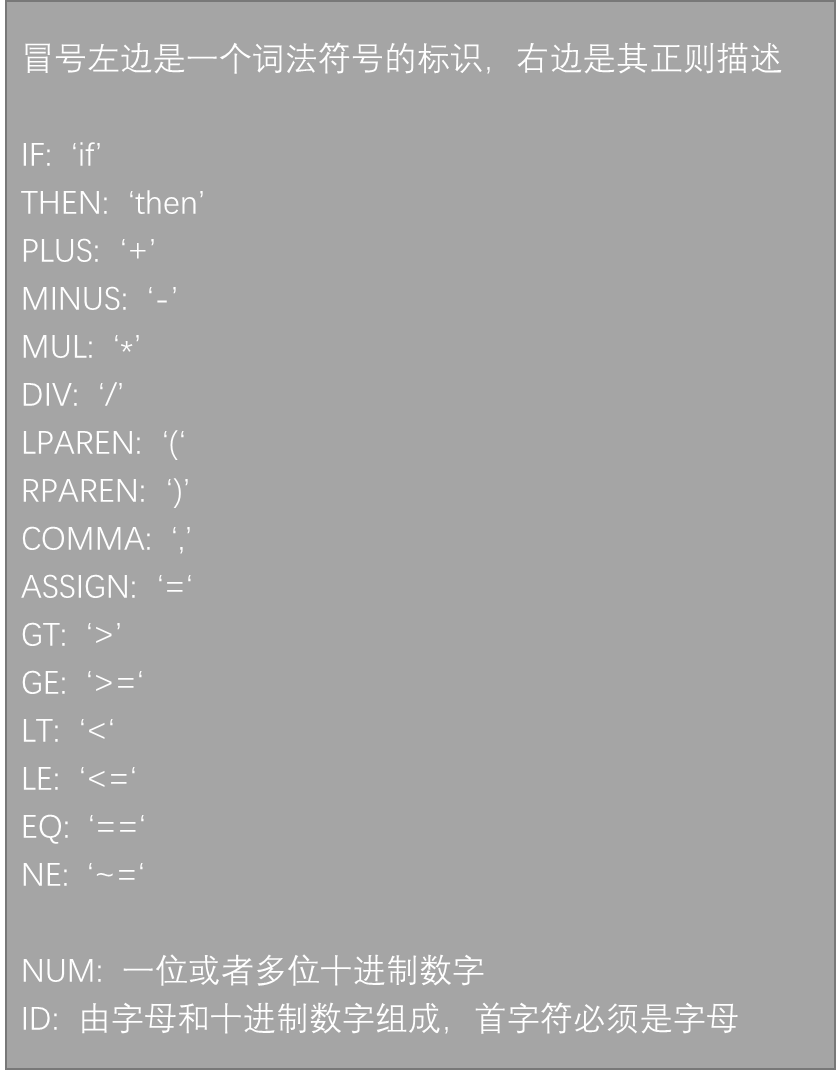
字符组成单词，单词组成句子。

描述哪些字符如何组成一个单词的规则，便是词法规则。

一个语言所具有的的词法符号的种类是有限的，每种词法符号我们称为一个token，比如C语言中，‘int’是一个整型定义的token，‘a1’是一个变量的token。

我们先写first语言的词法规则。

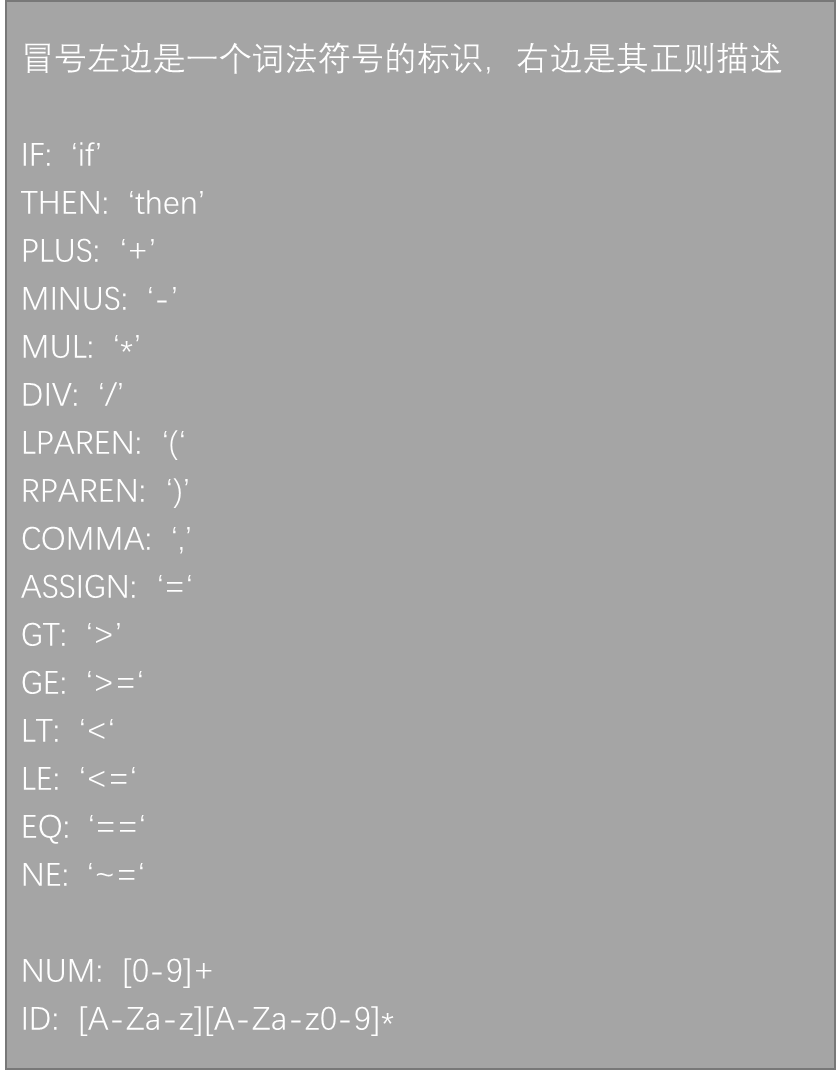
first语言中的token有以下几种：变量、数值、‘if’、‘then’、算数运算符（‘+’、‘-’、‘\*’、‘/’）、比较运算符（‘>’、‘>=’等）、赋值运算符（‘=’）、左右括号（‘（’，‘）’）。



上面我们用自然语言描述了first语言的词法规则。你可以试着打开某个编程语言的官方网站（Java、Python、C），官方在对用户介绍该语言的词法规则时并不会用自然语言，因为它比较冗长，而且很难获得风格的统一（每个人表达时有自己的风格，尽管是描述同一个意思，但表达出来的句子可以很不一样）。而是使用正则或者类正则的表达式。

正则表达式，regular expression，常用于字符串的模式匹配。许多语言或者工具都有提供正则匹配相关的资源，比如Java的java.util.regex包。如果从未接触过正则表达式，建议先在网上简单地学习一下，再继续阅读后面的内容。

下面我们用正则表达式来描述first语言。



2.2.3 语法规则

相关术语

语法规范： syntax / syntactic grammar / parser grammar

语法分析： syntactic analysis

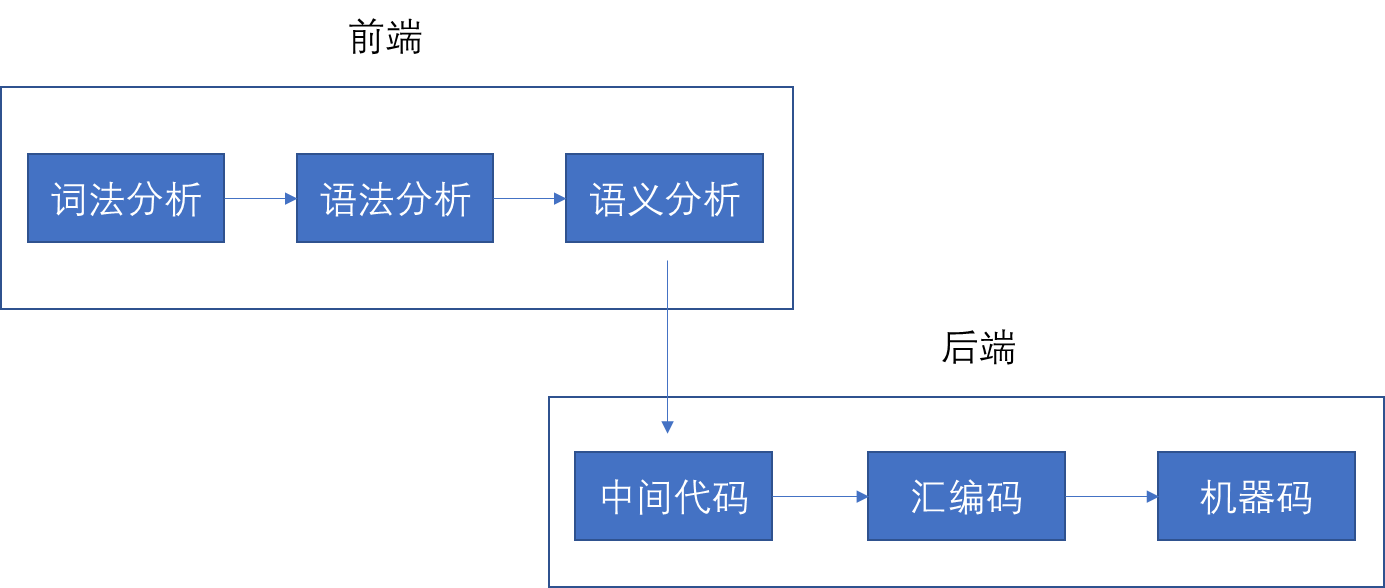
语法分析器： parser

2.2.4 语义规则

不同编程语言之间，词法和语法其实相差不大，重点是语义，语义的设计才是衡量一门语言是否易于使用的关键。C++的语义特性最多，在编译器行业人士中，公认的最复杂的语言，不仅仅是难用，还难实现其编译器。

自然语言描述。

2.3 编译器



脚本引擎。AOT、JIT，虚拟机。

2.4 编译器前端

2.4.1 词法分析

2.4.2 语法分析

2.4.3 语义分析

2.5 编译器后端

后端，优化是重点。与前端算法不同，NP完备。

不需要知道，因为高创用虚拟机和IR。

很多借助虚拟机的静态语言或者脚本语言，最初都是未考虑JIT的，但成功以后，为了获得效率上的大幅改进，引入了编译器后端技术，实现了JIT，如Java、Python、Lua。化为的方舟编译器。Java spring。

后端开发，也有像Antlr和lex&yacc这样能帮助你大幅提升效率的工具。后端工具：GGC、LLVM。

3 lex&yacc的简单介绍

4 利用lex&yacc实现first语言

5 利用lex&yacc实现MB-Basic的translator模块