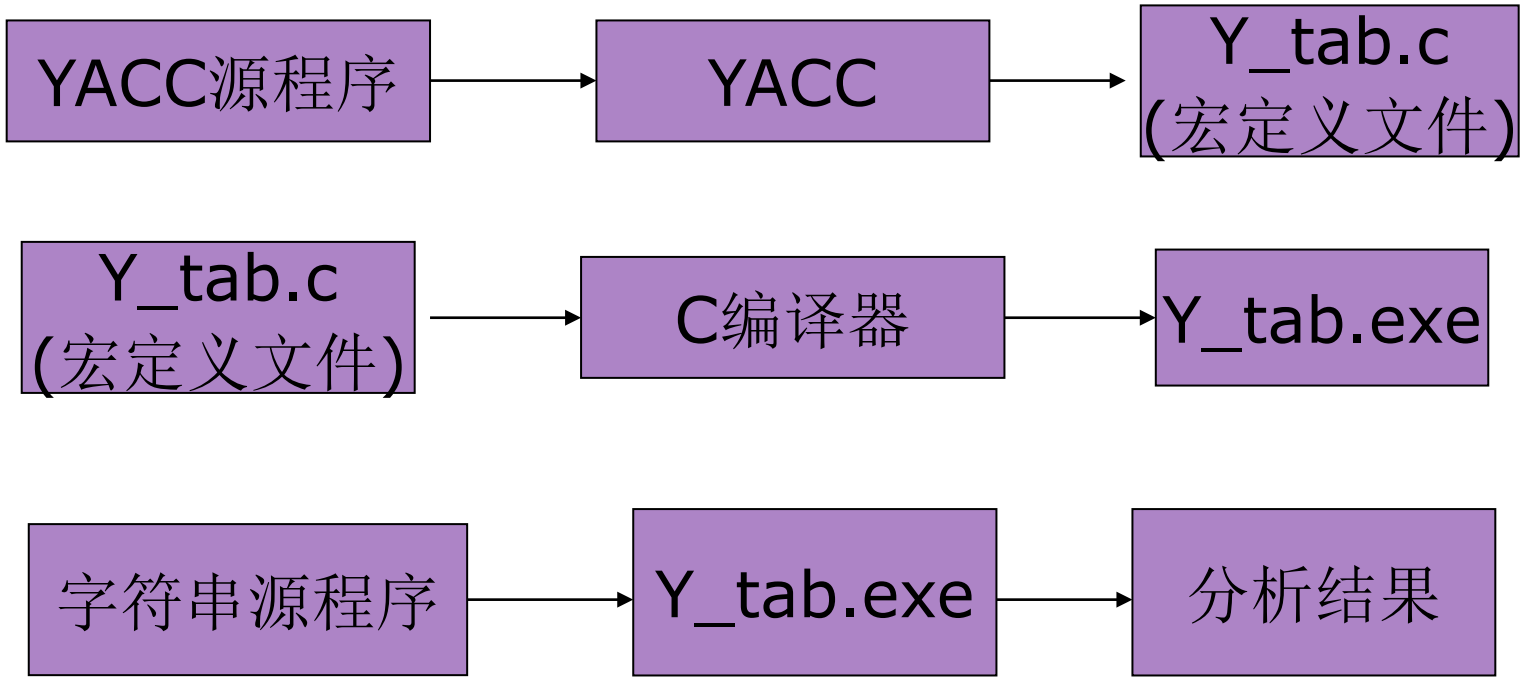
语法分析程序的自动生成工具

YACC

YACC的使用流程


YACC源程序
YACC
Y_tab.c
(宏定义文件)
Y_tab.c
(宏定义文件)
C编译器
Y_tab.exe
字符串源程序
Y_tab.exe
分析结果

YACC源程序结构

YACC源程序由三个部分组成，各部分以“%%”为分隔符。说明部分和程序部分可选，规则部分是必需的。

说明部分

%%

规则部分

%%

程序部分

YACC源程序结构—说明部分

•YACC源程序说明部分定义语法规则中要用的终结符号，语义动作中使用的数据类型、变量、语义值的联合类型以及语法规则中运算符的优先级等。说明部分可以是空的。

•说明部分通常包含两部分内容：

C语言代码部分

Yacc说明部分

YACC源程序结构—说明部分

%{
头文件表
宏定义
数据类型定义
全局变量定义
%}
文法开始符号定义
语义值类型定义
终结符定义
非终结符定义
优先级和结合性定义

1-头文件表

- yacc直接把这部分定义抄到所生成的C语言程序y.tab.c去的，所以要按C语言的语法规定来写。头文件表是一系列C语言的#include语句，要从每行的第一列开始写，例如：

```
%{
#include <stdio.h>
#include <math.h>
#include <ctype.h>
#include “header.h”
%}
…
%}
```

2-宏定义

- 这部分用C语言的 #define语句定义程序中要用的宏
- 例如

```
%{
…
#define max(x,y)((x＞y)?x:y)
…
%}
```

3–数据类型定义

- 这部分定义语义动作中或程序段部分中要用到的数据类型
- 例如

```
%{
…
typedef struct interval{
double lo,hi;
}INTERVAL;
…
%}
```

4-全局变量定义

- 外部变量（external variable）和yacc源程序中要用到的全局变量都在这部分定义
- 例如

```
%{
...
extern int nfg;
douhle dreg[ 26];
INTERVAL Vreg[26];
...
%}
```

5-语法开始符定义

- 上下文无关文法的开始符号是一个特殊的非终结符，所有的推导都从这个非终结符开始
- 在yacc中，语法开始符定义语句是

% start 非终结符……

- 如果没有上面的说明，yacc自动将语法规则部分中第一条语法规则左部的非终结符作为语法开始符

6-语义值类型定义

- 语法分析程序yyparse用的是LR分析方法，它在作语法分析时除了有一个状态栈外，还有一个语义值栈
- 语义值栈存放它所分析到的非终结符和终结符的语义值，这些语义值有的是从词法分析程序传回的，有的是在语义动作中赋与的
- 如果没有对语义值的类型做定义，那么 yacc认为它是整型（int）的，即所有语法符号如果赋与了语义值，则必须是整型的，否则会出类型错

6-语义值类型定义

- 但是用户经常会希望语义值的类型比较复杂，如双精度浮点数，字符串或树结点的指针
- 这时就可以用语义值类型定义进行说明。因为不同的语法符号的语义值类型可能不同，所以语义值类型说明就是将语义值的类型定义为一个联合（Union），这个联合包括所有可能用到的类型（各自对应一个成员名）
- 为了使用户不必在存取语义值时每次都指出成员名，在语义值类型定义部分还要求用户说明每一个语法符号（终结符和非终结符）的语义值是哪一个联合成员类型

6-语义值类型定义

- 例:

```
 % union{
int ival
double dval
INTERVAL vval;
}
```

- 引用时候的方式

```
%token <ival> DREG VREG
%token <dval> CONST
%type  <dval>dexp
%type  <vval>vexp
```

以%token开始的行定义的是终结符的类型

以%type开始的行定义是非终结符的类型

7-终结符定义

- 在yacc源程序语法规则部分出现的所有终结符（正文字符“+”，“-”等除外）等必须用%token定义，定义形式：
- 单一数据类型：

%token 终结符1 终结符2

多数据类型：

%token <类型> 终结符1 终结符2 …

8-终结符定义

优先级和结合性定义

%left 左结合

%right 右结合

%nonassoc 无结合性

%prec <终结符> 强制定义优先级

YACC源程序—语法规则部分

•语法规则部分是整个YACC源程序的主体，它是由一组产生式及相应的语义动作组成。

•规则部分包括修改的BNF格式的文法规则，以及将在识别出识别出相关的文法规则时被执行的C代码中的动作（即根据LALR（1）分析算法，在归约中使用）。

•文法规则中使用的元符号惯例如下：

通常，竖线|被用作替换（也可以分别写出替换项），而用来分隔文法规则的左右两边的箭头符号->在YACC中用冒号表示，最后，必须用分号来结束每个文法规则。

YACC源程序—语法规则部分

对文法中的产生式

$A \to \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_m$

在YACC程序中可表示成

A: α_1 {语义动作1}

| α_2 {语义动作2}

……

| α_m {语义动作m}

;

YACC源程序—语法规则部分

- YACC中的动作是由在每个文法规则中将其写作真正的C代码（在大括号中）来实现的。
- 在书写动作时，可以使用YACC伪变量。当识别一个文法规则时，规则中的每个符号都拥有一个值，除非它被参数改变了。
- 这些值由YACC保存在一个与分析栈保持平行的值栈（value stack）中，每个在栈中的符号值都可以使用以$开始的伪变量来引用。
- $$代表刚才被识别出来的非终结符的值，也就是文法规则左边的符号。伪变量$1、$2、$3等代表了文法规则右边的每个连续的符号。

YACC源程序—语法规则部分

•例：文法规则和动作：

```
exp :    exp '+' term { $$ = $1 + $3; }
```

含义是：当识别规则exp->exp+term时，左边exp值为右边的exp的值与右边的term的值之和，其中$$代表规则左部符号exp的值，$1代表规则右部第一个符号exp的值、$3表示规则右部第三个符号term的值。

YACC源程序—程序部分组成

- YACC源程序的程序部分包括：
  - 主程序 main()
  - 错误信息执行程序 yyerror(s)
  - 词法分析程序yylex()，可以与LEX进行整合
  - 用户在语义动作中用到的子程序
- YACC约定：
  - 传递词法分析程序token属性值的全程变量名：yylval
  - 生成的语法分析程序名为：yyparse();

Yacc的内置名称	含义／用处
y.tab.c	Yacc输出文件名称
y.tab.h	Yacc生成的头文件，包含了记号定义
yyparse	Yacc分析例程
yylval	栈中当前记号的值
yyerror	由Yacc使用的用户定义的错误信息打印机
error	Yacc错误伪记号
yyerrok	在错误之后重置分析程序的过程
yychar	包括导致错误的先行记号
YYSTYPE	定义分析栈的值类型的预处理器符号
yydebug	变量，当由用户设置为1时则导致生成有关分析动作的运行信息