Yacc is a tool that will generate a parser given an LR(0) grammar.

Structure of a Yacc Grammar

… definition section …
%%
… rules section …
%%
… user subroutine section …

Symbol Conventions

Typically, non-terminal symbols are given in lowercase and terminal symbols are assigned all capital letters. For example, the rule:

program → *program-head declarations program-body* **.**

would be expressed for a yacc grammar as

```
program:
    program_head declarations program_body PERIOD
```

Note that hyphens have been change to underscores to satisfy the C++ rules for identifiers and the period at the extreme right on the right hand side (RHS) of the rule has been changed to a capitalized spelling.

Definition Section

The definition section can contain

- *literal block*
  Declarations necessary for grammar actions and user subroutines are placed in the *literal block*. The *literal block* includes all **.h** files. A *literal block* is enclosed between **%{** and **%}** on separate lines as shown below.
  %{
  … C++ macro preprocessor definitions, declarations, and code …
  %}

- **%union** declarations
  The **%union** declaration associates terminal and non-terminal symbols with C-types. Identifiers defined in **%token** declarations and **%type** declarations are given specific types to be exploited in actions in the rules section. For example

```
%union {
    string* token;
    SList* slist;
}
%%
%token <token> ID
%type <slist> identifier_list
%%
identifier_list:
    ID
     {tfs << endl << "identifier_list -> ID(" << (*$1) << ")";
      $$=new SList;
      $$->Insert(*$1);
     }
identifier_list:
    identifier_list COMMA ID
     {tfs << endl << "identifier_list -> " << (*$1) << " , ID(" << (*$3) << ")";
      $$->Insert(*$3);
     }
```

  Notes:
  1. The symbolic type *token* is assigned the type *string**** in the **%union** declaration.
  2. The symbolic type *slist* is assigned the type *SList**** in the **%union** declaration.
  3. The **%token** declaration defines the terminal symbol *ID* to have the type assigned to the symbol type *token*. The terminal symbol *ID* now has type *string**.
  4. The **%type** declaration defines the non-terminal symbol *identifier_list* to have the type assigned to the symbol type *slist*. The non-terminal symbol now has type *SList**.
  5. The rule, *identifier-list* → **id**, expressed in the yacc grammar as
     ```
     identifier_list:
         ID
     ```
     has symbolic references to actual symbols represented by the terminal and non-terminal symbols in the rule. $1 refers to the first symbol on the RHS of the rule. $2, refers to the second symbol on the RHS of the rule. If there were five symbols on the RHS, $5 would refer to the fifth symbol on the RHS. $$ refers to the non-terminal on the LHS.

     In this case, because of prior **%union** and **%token** declarations, $1 has type *string** and contains a pointer to the actual string recognized by the scanner and parser.

     $$ has type SList* because of prior **%union** and **%type** declarations.

     We first created a new *SList*, a string list, and, then we inserted the first identifier, a string, in the *SList*.
- **%token** declarations
  %token declarations are used to define terminal symbols. Terminal symbols defined by **%token** declarations are made available to a scanner implemented using ***lex***. File y.tab.h is created when ***yacc*** is invoked. File **y.tab.h** assigns positive integer values to terminal symbols

defined using **%token** declarations. The values assigned to the terminal symbols are their token codes not the actual values represented by the token. A token is an integer code and a spelling. The spelling is the string of characters recognized by the scanner for that token.

To make the strings recognized by the scanner available to the parser for the example above, you must add the following statement to the scanner.

yylval.*token*=**new** *string*(*yytext*);

Variable *yytext* has type **char*** and points to the most recent string of characters recognized by the scanner.

- **%type** declarations
  %type declarations perform much the same function as **%token** declarations with the difference that **%type** declarations are designed for non-terminal symbols whereas **%token** declarations are reserved for terminal symbols. **%type** declarations work in concert with **%union** declarations. **%union** declarations associate a C type with a symbolic type name. The symbolic type name is associated with a non-terminal symbol by a **%type** declaration.

- **%start** declarations
  The **%start** declaration is used to alter the start symbol, a non-terminal.
- **%left** declarations
  The **%left** declaration makes an operator left associative.
  The order in which operators are specified defines their precedence. For example,
  %left '+' '-'
  %left '*' '/'
  make operators '+' , '-' , '*', and '/' left associative. Operators '*' and '/' have higher precedence than '+' and '-'.

- **%right** declarations
- The **%right** declaration makes an operator right associative.
- **%nonassoc** declarations
  The **%nonassoc** defines an operator to have to association.

Rules Section

The rules section contains

- grammar rules
  A rule of the grammar has a Left Hand Side (LHS) and a Right Hand Side (RHS). For example, consider the following expression grammar below with actions enclosed between **{** and **}**.
- actions containing C++ code

```
%union {
    double real;
    string* strlit
}
%token PLUS MINUS STAR SLASH LPAREN RPAREN
%token <real> REALIT
%token <strlit> ID
%type  <real> expression term factor
%%
```

statement:
    ID ASSIGN expression **{***cout* **<<** *endl* **<< (*$1) " := " << $3;}**
*expression***:**
    term **{$$=$1;}**
*expression***:**
    expression **PLUS** *term* **{$$=$1+$3;}**
expression:
    expression **MINUS** *term* **{$$=$1-$3;}**
*term***:**
    factor **{$$=$1;}**
*term***:**
    term **STAR** *factor* **{$$=$1*$3;}**
*term***:**
    term **SLASH** *factor* **{$$=$1/$3;}**
*factor***:**
    REALIT **{$$=$1;}**
*factor***:**
    LPAREN *expression* **RPAREN** **{$$=$2;}**
*factor***:**
    MINUS *factor* **{$$=-$1;}**