# 編譯程式

Programming Assignment 2

# Syntax Analyzer

for miniC language

系級:　資工三
學號:　410121021
姓名:　林育慈

2015. 5. 14

## Problem Description

1. Use yacc or bison to implement a syntax analyzer for the miniC language.

   ➢ The syntax rules in details.

```
Smallc_program      ::= (Type_specifier id '(' (Param_decl_list)? ')' Compound_stmt)+

Type_specifier      ::= int

Param_decl_list     ::= Param_decl (',' Param_decl)*

Param_decl          ::= Type_specifier id

Compound_stmt       ::= '{' (Var_decl* Stmt*)? '}'

Var_decl            ::= Type_specifier Var_decl_list ';'

Var_decl_list       ::= Variable_id (',' Variable_id)*

Variable_id         ::= id ('=' Expr)?

Stmt                ::= Compound_stmt | Cond_stmt | While_stmt | Assign_stmt
                    |   break ';' | continue ';' | return Expr ';'
                    |   printf '(' string (',' Expr)? ')' ';'

Assign_stmt         ::= id '=' Expr ';'

Cond_stmt           ::= if '(' Expr ')' Stmt (else Stmt)?

While_stmt          ::= while '(' Expr ')' Stmt

Expr                ::= id '=' Expr | Condition

Condition           ::= Disjunction | Disjunction '?' Expr ':' Condition

Disjunction         ::= Conjunction | Disjunction '||' Conjunction

Conjunction         ::= Comparison | Conjunction '&&' Comparison

Comparison          ::= Relation | Relation '==' Relation

Relation            ::= Sum | Sum ('<' | '>') Sum

Sum                 ::= Sum '+' Term | Sum '-' Term | Term

Term                ::= Term '*' Factor | Term '/' Factor | Term '%' Factor | Factor

Factor              ::= '!' Factor | '-' Factor | Primary

Primary             ::= num | id | id '(' Expr_list ')' | '(' Expr ')'

Expr_list           ::= Expr (',' Expr )*
```

   ➢ You are requested to separate the C code and the yacc/bison specification into distinct files.

# Program listing

**miniC.h**

```c
extern int yylex();
extern int yyparse();
extern FILE *yyin;
extern FILE *yyout;
extern FILE *yyerr;

extern char c_name[16];
extern int c_val;
```

**main.c**

```c
#include <stdio.h>
#include "miniC.h"

char c_name[16];
int c_val;

int main(int argc, char *argv[]){
      yyin = fopen(argv[1], "r");
      yyparse();
      return 0;
}
```

**Makefile** // 用來呼叫flex並編譯產生執行檔

```makefile
main:  c_yacc.o  c_lex.o main.o
      gcc -o parse.exe c_lex.o c_yacc.o main.o

c_lex.o: c_lex.c
      gcc -c c_lex.c

c_yacc.o: c_yacc.c
      gcc -c c_yacc.c

c_lex.c: c_lex.l c_yacc.h miniC.h
      flex -oc_lex.c c_lex.l

c_yacc.c: c_yacc.y miniC.h
      bison -d -o c_yacc.c c_yacc.y

main.o: main.c
      gcc -c main.c

# To clean the generated files
clean:
      rm *.o c_lex.c c_yacc.c c_yacc.h parse.exe
```

**c_yacc.y** // syntax rules, difinition -> To generate c_yacc.h / c_yacc.c

```
%{
      #include <stdio.h>
      #include <stdlib.h>
      #include <string.h>
      #include "miniC.h"
%}
%token INT
%token LP RP LSP RSP
%token SEMI COMMA ASSIGN QUES COLON
%token IF ELSE WHILE
%token PRINTF BREAK CONTINUE RETURN
%token ID NUM STRING
%token OR AND NOT EQ LT GT ADD MINUS DIV TIMES MOD
%token COMMENT
%left OR AND NOT
%left EQ LT GT
%left ADD MINUS
%left DIV TIMES MOD
%left ELSE
%expect 1
%%
```

```

smallc_program   : Type_specifier ID LP Param_decl_list RP Compound_stmt smallc_program {printf("smallc_program => Type_specifier ID LP Prarm_decl_list RP Compound_stmt smallc_program\n******** Parse OK ********\n"); }
                 | Type_specifier ID LP RP Compound_stmt smallc_program {printf("smallc_program => Type_specifier ID LP RP Compound_stmt smallc_program\n******** Parse OK ********\n"); }
                 | Type_specifier ID LP Param_decl_list RP Compound_stmt {printf("smallc_program => Type_specifier ID LP Prarm_decl_list RP Compound_stmt\n"); }
                 | Type_specifier ID LP RP Compound_stmt {printf("smallc_program => Type_specifier ID LP RP Compound_stmt\n"); }
                 ;
Type_specifier   : INT {printf("Type_specifier => INT\n"); }
                 ;
Param_decl_list  : Param_decl_list COMMA Param_decl {printf("Param_decl_list => Param_decl_list COMMA Param_decl\n"); }
                 | Param_decl {printf("Param_decl_list => Param_decl\n"); }
                 ;
Param_decl       : Type_specifier ID {printf("Param_decl => Type_specifier ID\n");
                   }
                 ;
Compound_stmt    : LSP VDs Ss RSP {printf("Compound_stmt => LSP VDs Ss RSP\n"); }
                 | LSP VDs RSP {printf("Compound_stmt => LSP VDs RSP\n"); }
                 | LSP Ss RSP {printf("Compound_stmt => LSP Ss RSP\n"); }
                 | LSP RSP {printf("Compound_stmt => LSP RSP\n"); }
                 ;
VDs              : VDs Var_decl{printf("VDs => VDs Var_decl\n"); }
                 | Var_decl {printf("VDs => Var_decl\n"); }
                 ;
Ss               : Ss Stmt {printf("Ss => Ss Stmt\n"); }
                 | Stmt {printf("Ss => Stmt\n"); }
                 ;
Var_decl         : Type_specifier Var_decl_list SEMI {printf("Var_decl => Type_specifier Var_decl_list SEMI\n"); }
                 ;
Var_decl_list    : Var_decl_list COMMA Variable_id {printf("Var_decl_list => Var_decl_list COMMA Variable_id\n"); }
                 | ariable_id {printf("Var_decl_list => Variable_id\n"); }
                 ;
Variable_id      : ID ASSIGN Expr {printf("Variable_id => ID ASSIGN Expr\n"); }
                 | ID {printf("Variable_id => ID\n"); }
                 ;
Stmt             : Compound_stmt {printf("Stmt => Compound_stmt\n"); }
                 | Cond_stmt {printf("Stmt => Cond_stmt\n"); }
                 | While_stmt {printf("Stmt => While_stmt\n"); }
                 | Assign_stmt {printf("Stmt => Assign_stmt\n"); }
                 | BREAK SEMI {printf("Stmt => BREAK SEMI\n"); }
                 | CONTINUE SEMI {printf("Stmt => CONTINUE SEMI\n"); }
                 | RETURN Expr SEMI {printf("Stmt => RETURN Expr SEMI\n"); }
                 | PRINTF LP STRING COMMA Expr RP SEMI {printf("Stmt => PRINTF LP STRING COMMA Expr RP SEMI\n"); }
                 | PRINTF LP STRING RP SEMI {printf("Stmt => PRINTF LP STRING RP SEMI\n"); }
                 ;
Assign_stmt      : ID ASSIGN Expr SEMI {printf("Assign_stmt => ID ASSIGN Expr SEMI\n"); }
                 ;
Cond_stmt        : IF LP Expr RP Stmt ELSE Stmt {printf("Cond_stmt => IF LP Expr RP Stmt ELSE Stmt\n"); }
                 | IF LP Expr RP Stmt {printf("Cond_stmt => IF LP Expr RP Stmt\n");
                   }
                 ;
```

```
While_stmt        : WHILE LP Expr RP Stmt {printf("While_stmt => WHILE LP Expr RP Stmt\n"); }
                  ;
Expr              : ID ASSIGN Expr {printf("Expr => ID ASSIGN Expr\n"); }
                  | Condition {printf("Expr => Condition\n"); }
                  ;
Condition         : Disjunction QUES Expr COLON Condition {printf("Condition => Disjunction QUEST Expr COLON Condition\n"); }
                  | Disjunction {printf("Condition => Disjunction\n"); }
                  ;
Disjunction       : Disjunction OR Conjunction {printf("Disjunction => Disjunction OR Conjunction\n"); }
                  | Conjunction {printf("Disjunction => Conjunction\n"); }
                  ;
Conjunction       : Conjunction AND Comparison {printf("Conjunction => Conjunction AND Comparison\n"); }
                  | Comparison {printf("Conjunction => Comparison\n"); }
                  ;
Comparison        : Relation EQ Relation {printf("Compatison => Relation EQ Relation\n"); }
                  | Relation {printf("Comparison => Relation\n"); }
                  ;
Relation          : Sum LT Sum {printf("Relation => Sum LT Sum\n"); }
                  | Sum GT Sum {printf("Relation => Sum GT Sum\n"); }
                  | Sum {printf("Relation => Sum\n"); }
                  ;
Sum               : Sum ADD Term {printf("Sum => Sum ADD Term\n"); }
                  | Sum MINUS Term {printf("Sum => Sum MINUS Term\n"); }
                  | Term {printf("Sum => Term\n"); }
                  ;
Term              : Term TIMES Factor {printf("Term => Term TIMES Factor\n"); }
                  | Term DIV Factor {printf("Term => Term DIV Factor\n"); }
                  | Term MOD Factor {printf("Term => Term MOD Factor\n"); }
                  | Factor {printf("Term => Factor\n"); }
                  ;
Factor            : NOT Factor {printf("Factor => NOT Factor\n"); }
                  | MINUS Factor {printf("Factor => MINUS Factor\n"); }
                  | Primary {printf("Factor => Primary\n"); }
                  ;
Primary           : ID LP Expr_list RP {printf("Primary => ID LP Expr_list RP\n"); }
                  | LP Expr RP {printf("Primary => LP Expr RP\n"); }
                  | NUM {printf("Primary => NUM\n"); }
                  | ID {printf("Primary => ID\n"); }
                  ;
Expr_list         : Expr_list COMMA Expr {printf("Expr_list => Expr_list COMMA Expr\n"); }
                  | Expr {printf("Expr_list => Expr\n"); }
                  ;

%%

int yyerror(char *s){
      printf("%s\n", s);
}
```

**c_lex.l**  // lexical定義、規則，讓flex用來產生c_lex.c

```
%{
#include "c_lex.h"
%}

ID [A-Za-z_][A-Za-z0-9_]*
NUM [0-9]+
STRING \"(\\.|[^"]|[^\n])*\"
COMMENT "//"[^"\n"]*

%%
break         {return BREAK; }
continue      {return CONTINUE; }
else          {return ELSE; }
if            {return IF; }
int           {return INT; }
return        {return RETURN; }
while         {return WHILE; }
printf        {return PRINTF; }
{STRING}      {sscanf(yytext, "%s", c_name); return STRING; }
"+"           {return ADD; }
"-"           {return MINUS; }
"*"           {return TIMES; }
"/"           {return DIV; }
"%"           {return MOD; }
"!"           {return NOT; }
"?"           {return QUES; }
":"           {return COLON; }
"="           {return ASSIGN; }
","           {return COMMA; }
"<"           {return LT; }
">"           {return GT; }
"("           {return LP; }
")"           {return RP; }
"{"           {return LSP; }
"}"           {return RSP; }
"||"          {return OR; }
"&&"          {return AND; }
"=="          {return EQ; }
"\""          {return QUOTE; }
";"           {return SEMI; }
{ID}          {sscanf(yytext, "%s", c_name); return ID; }
{NUM}         {sscanf(yytext, "%d", &c_val); return NUM; }
{COMMENT}     {return COMMENT; }
[ \t\n]       {}
.             {}
%%

int yywrap(){
return 1;
}
```

## Test run results

**test.c** // 測試程式

```c
int ComputeFac(int num){
      int num_aux;
      if (num < 1)
            num_aux = 1;
      else
            num_aux = num * ComputeFac(num - 1);
      return num_aux;
}

int main(){
      printf("%d\n", ComputeFac(10));
}
```

**result.txt** // 根據測試程式分析後的結果

```
Type_specifier => INT
Type_specifier => INT
Param_decl => Type_specifier ID
Param_decl_list => Param_decl
Type_specifier => INT
Variable_id => ID
Var_decl_list => Variable_id
Var_decl => Type_specifier
Var_decl_list SEMI
VDs => Var_decl
Primary => ID
Factor => Primary
Term => Factor
Sum => Term
Primary => NUM
Factor => Primary
Term => Factor
Sum => Term
Relation => Sum LT Sum
Comparison => Relation
Conjunction => Comparison
Disjunction => Conjunction
Condition => Disjunction
Expr => Condition
Primary => NUM
Factor => Primary
Term => Factor
Sum => Term
Relation => Sum
Comparison => Relation
Conjunction => Comparison
Disjunction => Conjunction
Condition => Disjunction
Expr => Condition
Assign_stmt => ID ASSIGN Expr SEMI
Stmt => Assign_stmt
Primary => ID
Factor => Primary
Term => Factor
Primary => ID
Factor => Primary
Term => Factor
Sum => Term
Primary => NUM
Factor => Primary
Term => Factor
Sum => Sum MINUS Term
Relation => Sum
Comparison => Relation
Conjunction => Comparison
Disjunction => Conjunction
Condition => Disjunction
Expr => Condition
Expr_list => Expr
Primary => ID LP Expr_list RP
Factor => Primary
Term => Term TIMES Factor
Sum => Term
Relation => Sum
Comparison => Relation
Conjunction => Comparison
Disjunction => Conjunction
Condition => Disjunction
Expr => Condition
Assign_stmt => ID ASSIGN Expr SEMI
Stmt => Assign_stmt
Cond_stmt => IF LP Expr RP Stmt ELSE Stmt
Stmt => Cond_stmt
Ss => Stmt
Primary => ID
Factor => Primary
Term => Factor
Sum => Term
Relation => Sum
Comparison => Relation
Conjunction => Comparison
Disjunction => Conjunction
Condition => Disjunction
Expr => Condition
Stmt => RETURN Expr SEMI
```

```
81   Ss => Ss Stmt
82   Compound_stmt => LSP VDs Ss RSP
83   Type_specifier => INT
84   Primary => NUM
85   Factor => Primary
86   Term => Factor
87   Sum => Term
88   Relation => Sum
89   Comparison => Relation
90   Conjunction => Comparison
91   Disjunction => Conjunction
92   Condition => Disjunction
93   Expr => Condition
94   Expr_list => Expr
95   Primary => ID LP Expr_list RP
96   Factor => Primary
97   Term => Factor
98   Sum => Term
99   Relation => Sum
100  Comparison => Relation
101  Conjunction => Comparison
102  Disjunction => Conjunction
103  Condition => Disjunction
104  Expr => Condition
105  Stmt => PRINTF LP STRING COMMA Expr RP SEMI
106  Ss => Stmt
107  Compound_stmt => LSP Ss RSP
     smallc_program => Type_specifier ID LP RP
108  Compound_stmt
     smallc_program => Type_specifier ID LP
     Prarm_decl_list RP Compound_stmt
     smallc_program
109  ******** Parse OK ********
```

## Discussion

HW1做的是miniC的詞彙分析，而這次是接續著詞彙分析，去進行文法的分析。

其實Bison的描述不難，但是過程中真的要細心，否則不小心寫了會混淆、矛盾的語法，可就一直無法進行下去了。因此在做的過程當中，真心覺得一個語言的產生其實真的很不容易，若這次的作業事要自行設計文法的話應該是真的沒有辦法吧！總之，這次的作業真心覺得是考驗細心啊！