

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

```
1  extern int yylex();
2  extern int yyparse();
3  extern FILE *yyin;
4  extern FILE *yyout;
5  extern FILE *yyerr;
6
7  extern char c_name[16];
8  extern int c_val;
```

main.c

```
1  #include <stdio.h>
2  #include "miniC.h"
3
4  char c_name[16];
5  int c_val;
6
7  int main(int argc, char *argv[]){
8      yyin = fopen(argv[1], "r");
9      yyparse();
10     return 0;
11 }
```

Makefile // 用來呼叫flex並編譯產生執行檔

```
1  main:  c_yacc.o  c_lex.o main.o
2          gcc -o parse.exe c_lex.o c_yacc.o main.o
3
4  c_lex.o: c_lex.c
5          gcc -c c_lex.c
6
7  c_yacc.o: c_yacc.c
8          gcc -c c_yacc.c
9
10 c_lex.c: c_lex.l c_yacc.h miniC.h
11         flex -oc_lex.c c_lex.l
12
13 c_yacc.c: c_yacc.y miniC.h
14         bison -d -o c_yacc.c c_yacc.y
15
16 main.o: main.c
17         gcc -c main.c
18
19 # To clean the generated files
20 clean:
21         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

```
1  %{
2      #include <stdio.h>
3      #include <stdlib.h>
4      #include <string.h>
5      #include "miniC.h"
6  %}
7  %token INT
8  %token LP RP LSP RSP
9  %token SEMI COMMA ASSIGN QUES COLON
10 %token IF ELSE WHILE
11 %token PRINTF BREAK CONTINUE RETURN
12 %token ID NUM STRING
13 %token OR AND NOT EQ LT GT ADD MINUS DIV TIMES MOD
14 %token COMMENT
15 %left OR AND NOT
16 %left EQ LT GT
17 %left ADD MINUS
18 %left DIV TIMES MOD
19 %left ELSE
20 %expect 1
21 %%
```

```

22
23     smallc_program      : Type_specifier ID LP Param_decl_list RP Compound_stmt
                           smallc_program {printf("smallc_program => Type_specifier ID LP
                           Parm_decl_list RP Compound_stmt smallc_program\n***** Parse
                           OK *****\n"); }
24     | 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"); }
25     | Type_specifier ID LP Param_decl_list RP Compound_stmt
                           {printf("smallc_program => Type_specifier ID LP Parm_decl_list
                           RP Compound_stmt\n"); }
26     | Type_specifier ID LP RP Compound_stmt {printf("smallc_program =>
                           Type_specifier ID LP RP Compound_stmt\n"); }
27     ;
28     Type_specifier      : INT {printf("Type_specifier => INT\n"); }
29     ;
30     Param_decl_list     : Param_decl_list COMMA Param_decl {printf("Param_decl_list =>
                           Param_decl_list COMMA Param_decl\n"); }
31     | Param_decl {printf("Param_decl_list => Param_decl\n"); }
32     ;
33     Param_decl          : Type_specifier ID {printf("Param_decl => Type_specifier ID\n");
34     }
35     ;
36     Compound_stmt       : LSP VDs Ss RSP {printf("Compound_stmt => LSP VDs Ss RSP\n"); }
37     | LSP VDs RSP {printf("Compound_stmt => LSP VDs RSP\n"); }
38     | LSP Ss RSP {printf("Compound_stmt => LSP Ss RSP\n"); }
39     | LSP RSP {printf("Compound_stmt => LSP RSP\n"); }
40     ;
41     VDs                 : VDs Var_decl {printf("VDs => VDs Var_decl\n"); }
42     | Var_decl {printf("VDs => Var_decl\n"); }
43     ;
44     Ss                  : Ss Stmt {printf("Ss => Ss Stmt\n"); }
45     | Stmt {printf("Ss => Stmt\n"); }
46     ;
47     Var_decl            : Type_specifier Var_decl_list SEMI {printf("Var_decl =>
                           Type_specifier Var_decl_list SEMI\n"); }
48     ;
49     Var_decl_list       : Var_decl_list COMMA Variable_id {printf("Var_decl_list =>
                           Var_decl_list COMMA Variable_id\n"); }
50     | Variable_id {printf("Var_decl_list => Variable_id\n"); }
51     ;
52     Variable_id         : ID ASSIGN Expr {printf("Variable_id => ID ASSIGN Expr\n"); }
53     | ID {printf("Variable_id => ID\n"); }
54     ;
55     Stmt                : Compound_stmt {printf("Stmt => Compound_stmt\n"); }
56     | Cond_stmt {printf("Stmt => Cond_stmt\n"); }
57     | While_stmt {printf("Stmt => While_stmt\n"); }
58     | Assign_stmt {printf("Stmt => Assign_stmt\n"); }
59     | BREAK SEMI {printf("Stmt => BREAK SEMI\n"); }
60     | CONTINUE SEMI {printf("Stmt => CONTINUE SEMI\n"); }
61     | RETURN Expr SEMI {printf("Stmt => RETURN Expr SEMI\n"); }
62     | PRINTF LP STRING COMMA Expr RP SEMI {printf("Stmt => PRINTF LP
                           STRING COMMA Expr RP SEMI\n"); }
63     | PRINTF LP STRING RP SEMI {printf("Stmt => PRINTF LP STRING RP
                           SEMI\n"); }
64     ;
65     Assign_stmt         : ID ASSIGN Expr SEMI {printf("Assign_stmt => ID ASSIGN Expr
                           SEMI\n"); }
66     ;
67     Cond_stmt           : IF LP Expr RP Stmt ELSE Stmt {printf("Cond_stmt => IF LP Expr RP
                           Stmt ELSE Stmt\n"); }
68     | IF LP Expr RP Stmt {printf("Cond_stmt => IF LP Expr RP Stmt\n");
69     }
    ;

```

```

70  While_stmt      : WHILE LP Expr RP Stmt {printf("While_stmt => WHILE LP Expr RP
71                  :                               Stmt\n"); }
72  Expr            : ID ASSIGN Expr {printf("Expr => ID ASSIGN Expr\n"); }
73                  | Condition {printf("Expr => Condition\n"); }
74                  ;
75  Condition       : Disjunction QUES Expr COLON Condition {printf("Condition =>
76                  :                               Disjunction QUEST Expr COLON Condition\n"); }
77                  | Disjunction {printf("Condition => Disjunction\n"); }
78                  ;
79  Disjunction     : Disjunction OR Conjunction {printf("Disjunction => Disjunction
80                  :                               OR Conjunction\n"); }
81                  | Conjunction {printf("Disjunction => Conjunction\n"); }
82                  ;
83  Conjunction     : Conjunction AND Comparison {printf("Conjunction => Conjunction
84                  :                               AND Comparison\n"); }
85                  | Comparison {printf("Conjunction => Comparison\n"); }
86                  ;
87  Comparison      : Relation EQ Relation {printf("Comparision => Relation EQ
88                  :                               Relation\n"); }
89                  | Relation {printf("Comparison => Relation\n"); }
90                  ;
91  Relation        : Sum LT Sum {printf("Relation => Sum LT Sum\n"); }
92                  | Sum GT Sum {printf("Relation => Sum GT Sum\n"); }
93                  | Sum {printf("Relation => Sum\n"); }
94                  ;
95  Sum             : Sum ADD Term {printf("Sum => Sum ADD Term\n"); }
96                  | Sum MINUS Term {printf("Sum => Sum MINUS Term\n"); }
97                  | Term {printf("Sum => Term\n"); }
98                  ;
99  Term            : Term TIMES Factor {printf("Term => Term TIMES Factor\n"); }
100                 | Term DIV Factor {printf("Term => Term DIV Factor\n"); }
101                 | Term MOD Factor {printf("Term => Term MOD Factor\n"); }
102                 | Factor {printf("Term => Factor\n"); }
103                 ;
104  Factor          : NOT Factor {printf("Factor => NOT Factor\n"); }
105                 | MINUS Factor {printf("Factor => MINUS Factor\n"); }
106                 | Primary {printf("Factor => Primary\n"); }
107                 ;
108  Primary         : ID LP Expr_list RP {printf("Primary => ID LP Expr_list RP\n"); }
109                 | LP Expr RP {printf("Primary => LP Expr RP\n"); }
110                 | NUM {printf("Primary => NUM\n"); }
111                 | ID {printf("Primary => ID\n"); }
112                 ;
113  Expr_list       : Expr_list COMMA Expr {printf("Expr_list => Expr_list COMMA
114                 :                               Expr\n"); }
115                 | Expr {printf("Expr_list => Expr\n"); }
116                 ;
117
118  %%
119
120  int yyerror(char *s){
121      printf("%s\n", s);
122  }

```

c_lex.1 // lexical定義、規則，讓flex用來產生c_lex.c

```
1  %{
2  #include "c_lex.h"
3  %}
4
5  ID [A-Za-z_][A-Za-z0-9_]*
6  NUM [0-9]+
7  STRING \"(\\.|[^\"]|[\n])*\"
8  COMMENT \"//\"[^\n]*
9
10 %%
11 break      {return BREAK; }
12 continue   {return CONTINUE; }
13 else       {return ELSE; }
14 if         {return IF; }
15 int        {return INT; }
16 return     {return RETURN; }
17 while      {return WHILE; }
18 printf     {return PRINTF; }
19 {STRING}   {sscanf(yytext, \"%s\", c_name); return STRING; }
20 \"+\"       {return ADD; }
21 \"-\"       {return MINUS; }
22 \"*\"       {return TIMES; }
23 \"/\"       {return DIV; }
24 \"%\"       {return MOD; }
25 \"!\"       {return NOT; }
26 \"?\"       {return QUES; }
27 \":\"       {return COLON; }
28 \"=\"       {return ASSIGN; }
29 \",\"       {return COMMA; }
30 \"<\"       {return LT; }
31 \">\"       {return GT; }
32 \"(\"       {return LP; }
33 \")\"       {return RP; }
34 \"{\"       {return LSP; }
35 \"}\"       {return RSP; }
36 \"||\"      {return OR; }
37 \"&&\"      {return AND; }
38 \"==\"      {return EQ; }
39 \"\\\"\"     {return QUOTE; }
40 \";\"       {return SEMI; }
41 {ID}       {sscanf(yytext, \"%s\", c_name); return ID; }
42 {NUM}      {sscanf(yytext, \"%d\", &c_val); return NUM; }
43 {COMMENT}  {return COMMENT; }
44 [ \t\n]    {}
45 .          {}
46 %%
47
48 int yywrap(){
49     return 1;
50 }
```

Test run results

test.c // 測試程式

```
1  int ComputeFac(int num){
2      int num_aux;
3      if (num < 1)
4          num_aux = 1;
5      else
6          num_aux = num * ComputeFac(num - 1);
7      return num_aux;
8  }
9
10 int main(){
11     printf("%d\n", ComputeFac(10));
12 }
```

result.txt // 根據測試程式分析後的結果

| | | | |
|----|------------------------------------|----|---|
| 1 | Type_specifier => INT | 41 | Factor => Primary |
| 2 | Type_specifier => INT | 42 | Term => Factor |
| 3 | Param_decl => Type_specifier ID | 43 | Sum => Term |
| 4 | Param_decl_list => Param_decl | 44 | Primary => NUM |
| 5 | Type_specifier => INT | 45 | Factor => Primary |
| 6 | Variable_id => ID | 46 | Term => Factor |
| 7 | Var_decl_list => Variable_id | 47 | Sum => Sum MINUS Term |
| 8 | Var_decl => Type_specifier | 48 | Relation => Sum |
| 9 | Var_decl_list SEMI | 49 | Comparison => Relation |
| 10 | VDs => Var_decl | 50 | Conjunction => Comparison |
| 11 | Primary => ID | 51 | Disjunction => Conjunction |
| 12 | Factor => Primary | 52 | Condition => Disjunction |
| 13 | Term => Factor | 53 | Expr => Condition |
| 14 | Sum => Term | 54 | Expr_list => Expr |
| 15 | Primary => NUM | 55 | Primary => ID LP Expr_list RP |
| 16 | Factor => Primary | 56 | Factor => Primary |
| 17 | Term => Factor | 57 | Term => Term TIMES Factor |
| 18 | Sum => Term | 58 | Sum => Term |
| 19 | Relation => Sum LT Sum | 59 | Relation => Sum |
| 20 | Comparison => Relation | 60 | Comparison => Relation |
| 21 | Conjunction => Comparison | 61 | Conjunction => Comparison |
| 22 | Disjunction => Conjunction | 62 | Disjunction => Conjunction |
| 23 | Condition => Disjunction | 63 | Condition => Disjunction |
| 24 | Expr => Condition | 64 | Expr => Condition |
| 25 | Primary => NUM | 65 | Assign_stmt => ID ASSIGN Expr SEMI |
| 26 | Factor => Primary | 66 | Stmt => Assign_stmt |
| 27 | Term => Factor | 67 | Cond_stmt => IF LP Expr RP Stmt ELSE Stmt |
| 28 | Sum => Term | 68 | Stmt => Cond_stmt |
| 29 | Relation => Sum | 69 | Ss => Stmt |
| 30 | Comparison => Relation | 70 | Primary => ID |
| 31 | Conjunction => Comparison | 71 | Factor => Primary |
| 32 | Disjunction => Conjunction | 72 | Term => Factor |
| 33 | Condition => Disjunction | 73 | Sum => Term |
| 34 | Expr => Condition | 74 | Relation => Sum |
| 35 | Assign_stmt => ID ASSIGN Expr SEMI | 75 | Comparison => Relation |
| 36 | Stmt => Assign_stmt | 76 | Conjunction => Comparison |
| 37 | Primary => ID | 77 | Disjunction => Conjunction |
| 38 | Factor => Primary | 78 | Condition => Disjunction |
| 39 | Term => Factor | 79 | Expr => Condition |
| 40 | Primary => ID | 80 | 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！