編譯程式

Programming Assignment 3

Parse Tree

for miniC language

系級： 資工三
學號： 410121021
姓名： 林育慈

2015-06-11

Problem Description

1. Build the parse tree for input program written in 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 int yyerror(char *);
4    extern FILE *yyin;
5    extern FILE *yyout;
6    extern FILE *yyerr;
7
8    extern char name[16];
9    extern int val;
```

main.c

```
1    #include <stdio.h>
2    #include "minic.h"
3    #include "c_tree.h"
4
5    char name[16];
6    int val;
7    cSTM* program = NULL;
8
9    int main(int argc,char *argv[]) {
10       int t;
11
12       yyin = fopen(argv[1],"r");
13       yyparse();
14       print_stm( program );
15       printf("MiniC successfully builds a
     parse tree for %s!\n\n", argv[1]);
16       free_stm( program );
17   }
```

Makefile // 用來呼叫flex並編譯產生執行檔

```
1    ctree:c_yacc.o c_tree.o c_lex.o main.o
2          gcc -o ctree main.o c_tree.o c_lex.o c_yacc.o
3
4    c_lex.c:    c_lex.l minic.h c_yacc.h
5          flex -oc_lex.c c_lex.l
6
7    c_lex.o:    c_lex.c minic.h
8          gcc -c -o c_lex.o c_lex.c
9
10   c_yacc.c: c_yacc.y minic.h
11         bison -d -oc_yacc.c c_yacc.y
12
13   c_yacc.o: c_yacc.c minic.h
14         gcc -c c_yacc.c
15
16   c_tree.o: c_tree.c c_tree.h
17         gcc -c -o c_tree.o c_tree.c
18
19   main.o:     main.c c_lex.h
20         gcc -c -o main.o main.c
21
22   all:  cparse
23
24   clean:
25         rm *.o c_lex.c c_yacc.c c_yacc.h ctree
26
27   # hidden rules
28   .c:   .l
29         flex -o$< $@
```

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          #include "c_tree.h"
7    %}
8
9    %token BR CONT RET PRF
10   %token LP RP LBP RBP
11   %token INT STR
12   %token IF WHILE
13   %token ELSE ASSIGN SEMI COMMA COL DBQ
14   %token ID NUM
15   %left OR AND
16   %left NOT OPT
17   %left EQU LT GT
18   %left ADD MINUS
19   %left DIVIDE TIMES REMAIN
20   %expect 1
21
22   %union{ cSTM* sm;
23           cEXP* ex;
24           int   nu;
25           char* sr;
26         }
27
28   %type <sm> minic
29   %type <sm> prog
30   %type <sm> fun_dec
31   %type <ex> t_spec
32   %type <ex> par_lst
33   %type <ex> par_nxt
34   %type <ex> par_dec
35   %type <sm> com_stm
36   %type <sm> var_dl
37   %type <sm> var_dec
38   %type <ex> var_lst
39   %type <ex> var_id
40   %type <sm> stm_lst
41   %type <sm> stmt
42   %type <sm> cnd_stm
43   %type <sm> opt_els
44   %type <sm> whi_stm
45   %type <sm> ass_stm
46   %type <ex> opt_exp
47   %type <ex> expr
48   %type <ex> cond
49   %type <ex> disj
50   %type <ex> conj
51   %type <ex> comp
52   %type <ex> rel
53   %type <nu> ltgt
54   %type <ex> sum
55   %type <ex> term
56   %type <ex> factor
57   %type <ex> primary
58   %type <ex> arg_lst
59   %type <ex> arg_nxt
60   %type <nu> NUM
61   %type <sr> ID
62   %type <sr> STR
63   %%
```

```
64
65    miniC    : prog
66                {program = $1; $$ = program; printf("minic parses OK!\n"); }
67             |  {program = NULL; $$ = NULL; printf("******** Parsing failed! \n");}
68             ;
69
70    prog     :  fun_dec prog {$$ = $1; $$->next = $2; }
71             |  fun_dec {$$ = $1;}
72             ;
73
74    fun_dec  : t_spec ID LP par_lst RP com_stm
75                {$$ = create_stm(); $$->stm_id = sFUNC;   strcpy($1->name, $2);
76                 $$->exp1 = $1; $$->exp2 = $4; $$->stm1 = $6; }
77             ;
78
79    t_spec   : INT {$$= create_exp(); $$->exp_id = eTYPE; $$->val = eINT; }
80             ;
81
82    par_lst  : par_dec par_nxt {$$= create_exp(); $$->exp_id = ePLIST;
83                                $$->exp1 = $1;  $$->next = $2; }
84             |  {$$ = NULL;}
85             ;
86
87    par_nxt  : COMMA par_dec par_nxt   {$$= create_exp(); $$->exp_id = ePLIST;
88                                        $$->exp1 = $2; $$->next = $3; }
89             | {$$ = NULL;}
90             ;
91
92    par_dec  : t_spec ID  {$$= create_exp(); $$->exp_id = ePDEC;
93                           $$->exp1 = $1; strcpy($$->name, $2); }
94             ;
95
96    com_stm  : LBP var_dl stm_lst RBP {$$ = create_stm(); $$->stm_id = sCSTM;
97                                       $$->stm1 = $2; $$->stm2 = $3; }
98             ;
99
100   var_dl   : var_dec var_dl {$$ = create_stm(); $$->stm_id= sVDLIST;
101                              $$->stm1 = $1; $$->next = $2; }
102            | {$$ = NULL; }
103            ;
104
105   var_dec  :  t_spec var_lst SEMI {$$ = create_stm(); $$->stm_id= sVDEC;
106                                    $$->exp1 = $1; $$->exp2 = $2; }
107            ;
108
109   var_lst  : var_id COMMA var_lst {$$= create_exp(); $$->exp_id = eVLIST;
110                                    $$->exp1= $1; $$->next=$3; }
111            | var_id {$$ =  $1; }
112            ;
113
114   var_id   : ID ASSIGN expr {$$= create_exp(); $$->exp_id = eVASS;
115                              strcpy($$->name, $1); $$->exp1=$3; }
116            | ID {$$= create_exp(); $$->exp_id = eID; strcpy($$->name, $1); }
117            ;
118
119   stm_lst  :  stmt  stm_lst {$$ = create_stm(); $$->stm_id =sSLIST;
120                              $$->stm1=$1; $$->next=$2; }
121            | {$$ = NULL; }
122            ;
123
124   stmt     : com_stm {$$ = $1; }
125            | cnd_stm {$$ = $1; }
126            | whi_stm {$$ = $1; }
127            | ass_stm {$$ = $1; }
```

```
128           | BR SEMI {$$=create_stm(); $$->stm_id =sBR; }
129           | CONT SEMI {$$=create_stm(); $$->stm_id = sCONT; }
130           | RET expr SEMI {$$=create_stm(); $$->stm_id = sRET; $$->exp1=$2; }
131           | PRF LP STR opt_exp RP SEMI { $$=create_stm(); $$->stm_id = sASTM;
132                                          strcpy($$->name, $3); $$->exp1=$4; }
133           ;
134
135   cnd_stm : IF LP expr RP stmt opt_els {$$ = create_stm(); $$->stm_id = sISTM;
136                                        $$->exp1= $3; $$->stm1=$5; $$->stm2=$6; }
137           ;
138
139   opt_els : ELSE stmt {$$ = create_stm(); $$->stm_id = sOELS; $$->stm1 = $2; }
140           | {$$ = NULL;}
141           ;
142
143   whi_stm : WHILE LP expr RP stmt {$$ = create_stm(); $$->stm_id = sWSTM;
144                                    $$->exp1=$3; $$->stm1=$5; }
145           ;
146
147   ass_stm : ID ASSIGN expr SEMI {$$ = create_stm(); $$->stm_id = sASTM;
148                                  strcpy($$->name, $1); $$->exp1 = $3; }
149           ;
150
151   opt_exp : COMMA expr {$$= create_exp(); $$->exp_id =  eOEXP ; $$->exp1 = $2; }
152           | {$$ = NULL;}
153           ;
154
155   expr    : ID ASSIGN expr {$$= create_exp(); $$->exp_id = eEXP;
156                             strcpy($$->name, $1); $$->next = $3; }
157           | cond {$$=$1; }
158           ;
159
160   cond    : disj {$$=$1; }
161           | disj OPT expr COL cond {$$= create_exp(); $$->exp_id = eEOPT;
162                                     $$->exp1=$1; $$->exp2=$3; $$->next=$5; }
163           ;
164
165   disj    : conj OR disj {$$= create_exp(); $$->exp_id = eOR; $$->exp1 =$1;
166                           $$->exp2 = $3; }
167           | conj {$$=$1; }
168           ;
169
170   conj    : comp AND conj {$$= create_exp(); $$->exp_id = eAND ; $$->exp1 =$1;
171                            $$->exp2 =$3; }
172           | comp {$$=$1; }
173           ;
174
175   comp    : rel EQU rel {$$= create_exp(); $$->exp_id = eEQU; $$->exp1 = $1;
176                          $$->exp2 = $3; }
177           | rel {$$=$1; }
178           ;
179
180   rel     : sum ltgt sum {$$= create_exp(); $$->exp_id =eREL; $$->exp1 = $1;
181                           $$->exp2 = $3; }
182           | sum {$$=$1; }
183           ;
184   ltgt    : LT {$$ = eLT; }
185           | GT {$$ = eGT; }
186           ;
187
188   sum     : sum ADD term { $$= create_exp(); $$->exp_id =eADD;
189                            $$->exp1 = $1; $$->exp2 = $3; }
190           | sum MINUS term { $$= create_exp(); $$->exp_id =eMINUS;
191                              $$->exp1 = $1; $$->exp2 = $3; }
```

```
192           | term {$$=$1; }
193           ;
194    term   : term TIMES factor {$$= create_exp(); $$->exp_id =eTIMES;
195                                $$->exp1 = $1; $$->exp2 = $3; }
196           | term DIVIDE factor {$$= create_exp(); $$->exp_id =eDIVIDE;
197                                 $$->exp1 = $1; $$->exp2 = $3; }
198           | term REMAIN factor {$$= create_exp(); $$->exp_id =eREMAIN;
199                                 $$->exp1 = $1; $$->exp2 = $3; }
200           | factor {$$=$1; }
201           ;
202
203    factor : NOT factor {$$= create_exp(); $$->exp_id =eNOT; $$->exp1 = $2; }
204           | MINUS factor {$$= create_exp(); $$->exp_id =eUMINUS; $$->exp1 = $2; }
205           | primary {$$=$1; }
206           ;
207
208    primary : NUM {$$= create_exp(); $$->exp_id = eNUM; $$->val = $1; }
209           | ID {$$= create_exp(); $$->exp_id = eID; strcpy($$->name, $1); }
210           | ID LP arg_lst RP {$$= create_exp(); $$->exp_id = eFCALL;
211                               strcpy($$->name, $1); $$->exp1 = $3; }
212           | LP expr RP {$$= create_exp(); $$->exp_id =eLP; $$->exp1 = $2; }
213           ;
214
215    arg_lst : expr arg_nxt {$$= $1; $$->next=$2; }
216           | {$$=NULL; }
217           ;
218
219    arg_nxt : COMMA expr arg_nxt {$$= create_exp(); $$->exp_id = ePLIST;
220                                  $$->exp1 = $2; $$->next = $3; }
221           | {$$=NULL; }
222           ;
223
224    %%
225
226    int yyerror(char *s)
227    {
228      printf("%s\n",s);
229      return 0;
230    }
231
232
233
234
235
236
237
```

c_lex.h

```
1   #define L_MIN 0
2   #define NUM 1
3   #define ID 2
4   #define STR 3
5   #define BR 4
6   #define CONT 5
7   #define ELSE 6
8   #define IF 7
9   #define INT 8
10  #define RET 9
11  #define WHILE 10
12  #define PRF 11
13  #define ADD 12
14  #define MINUS 13
15  #define TIMES 14
16  #define DIVIDE 15
17  #define REMAIN 16
18  #define NOT 17
19  #define OPT 18
20  #define COL 19
21  #define ASSIGN 20
22  #define COMMA 21
23  #define LT 22
24  #define GT 23
25  #define LP 24
26  #define RP 25
27  #define LBP 26
28  #define RBP 27
29  #define OR 28
30  #define AND 29
31  #define EQU 30
32  #define DBQ 31
33  #define SEMI 32
34  #define COMMENT 33
35  #define L_MAX 34
36
37  extern int yylex();
38  extern FILE *yyin;
39
40  extern void print_lex( int );
41
42  extern char name[16];
43  extern int val;
```

c_tree.h

```
1   extern int  yylex();
2   extern int  yyparse();
3   extern FILE *yyin;
4   extern FILE *yyout;
5   extern FILE *yyerr;
6
7   // define tags for parse tree
8
9   #define sMIN 0
10  #define sFUNC 1
11  #define sSLIST 2
12  #define sCSTM 3
13  #define sVDLIST 4
14  #define sVDEC 5
15  #define sISTM 6
16  #define sWSTM 7
17  #define sASTM 8
18  #define sBR 9
19  #define sCONT 10
20  #define sRET 11
21  #define sPRF 12
22  #define sMAX 13
23  #define sOELS 14
24
25  #define eMIN 0
26  #define eTYPE 1
27  #define eINT 2
28  #define ePLIST 3
29  #define ePDEC 4
30  #define eVLIST 5
31  #define eVASS 6
32  #define eID 7
33  #define eEOPT 8
34  #define eOR 9
35  #define eAND 10
36  #define eEQU 11
37  #define eREL 12
38  #define eLT 13
39  #define eGT 14
40  #define eADD 15
41  #define eMINUS 16
42  #define eTIMES 17
43  #define eDIVIDE 18
44  #define eREMAIN 19
45  #define eNOT 20
46  #define eUMINUS 21
47  #define eNUM 22
48  #define eFCALL 23
49  #define eALIST 24
50  #define eMAX 25
51  #define eOEXP 26
52  #define eEXP 27
53  #define eLP 28
54
55  // define data structures for parse tree
56  typedef struct c_exp {
57     int  exp_id;
58     char name[16];
59     int  val;
60     struct c_exp *exp1;
61     struct c_exp *exp2;
62     struct c_exp *next;
63  } cEXP;
64
65  typedef struct c_stm {
66     int  stm_id;
67     char name[16];
68     struct c_exp *exp1;
69     struct c_exp *exp2;
70     struct c_stm *stm1;
71     struct c_stm *stm2;
72     struct c_stm *next;
73  } cSTM;
74
75  // declare utility functions
76  extern cEXP* create_exp();
77  extern cSTM* create_stm();
78  extern void free_exp( cEXP* );
79  extern void free_stm( cSTM* );
80  extern void print_exp( cEXP* );
81  extern void print_stm( cSTM* );
82
83  // global variables
84  extern cSTM* program;
85  extern char name[16];
86  extern int  val;
```

c_tree.c

```
1     #include <stdio.h>
2     #include <stdlib.h>
3     #include "c_lex.h"
4     #include "c_tree.h"
5
6     cEXP* create_exp ( ) {
7       cEXP* tmp;
8       tmp = (struct c_exp *)malloc(sizeof(struct c_exp));
9       if( tmp ) {
10        tmp->exp_id = eMIN;
11        tmp->name[0] = '\0';
12        tmp->val = 0;
13        tmp->exp1 = NULL;
14        tmp->exp2 = NULL;
15        tmp->next = NULL;
16      }
171     return tmp;
8     }
19
20    cSTM* create_stm ( ) {
21      cSTM* tmp;
22      tmp = (struct c_stm *)malloc(sizeof(struct c_stm));
23      if( tmp ) {
24        tmp->stm_id = sMIN;
25        tmp->exp1 = NULL;
26        tmp->exp2 = NULL;
27        tmp->stm1 = NULL;
28        tmp->stm2 = NULL;
29        tmp->next = NULL;
30      }
31      return tmp;
32    }
33
34    void free_exp ( cEXP* p ) {
35      if( p ) {
36        if( p->exp1 ) free_exp( p->exp1 );
37        if( p->exp2 ) free_exp( p->exp2 );
38        if( p->next ) free_exp( p->next );
39        free( p );
40      }
41    }
42
43    void free_stm ( cSTM* p ) {
44      if( p ) {
45        if( p->exp1 ) free_exp( p->exp1 );
46        if( p->exp2 ) free_exp( p->exp2 );
474       if( p->stm1 ) free_stm( p->stm1 );
8         if( p->stm2 ) free_stm( p->stm2 );
49        if( p->next ) free_stm( p->next );
50        free( p );
51      }
52    }
53
54    void print_exp ( cEXP* p ) {
55      if( p ) {
56        switch( p->exp_id ) {
57        case eTYPE: printf("cEXP(eTYPE):%d\n", p->val);
58                  printf("-- end of cEXP(eTYPE)\n");
59                   break;
60        case ePLIST: printf("cEXP(ePLIST):\n");
61                  print_exp( p->exp1 );
62                  print_exp( p->next );
63                  printf("-- end of cEXP(ePLIST)\n");
```

```
64                break;
65        case ePDEC: printf("cEXP(ePDEC): %s\n", p->name);
66                print_exp( p->exp1 );
67                printf("-- end of cEXP(ePDEC)\n");
68                break;
69        case eVLIST: printf("cEXP{eVLIST):\n");
70                  print_exp( p->exp1 );
71                 print_exp( p->next );
72                 printf("-- end of cEXP(eVLIST)\n");
73                   break;
74        case eVASS: printf("cEXP{eVASS): %s = \n", p->name);
75                  print_exp( p->exp1 );
76                printf("-- end of cEXP(eVASS)\n");
77                  break;
78        case eID: printf("cEXP{eID): %s\n", p->name);
79                printf("-- end of cEXP(eID)\n");
80                break;
81        case eEOPT: printf("cEXP{eEOPT):\n");
82                  print_exp( p->exp1 );
83                  print_exp( p->exp2 );
84                  print_exp( p->next );
85                printf("-- end of cEXP(eEOPT)\n");
86                  break;
87        case eOR: printf("cEXP{eOR):\n");
88                print_exp( p->exp1 );
89                print_exp( p->exp2 );
90                printf("-- end of cEXP(eOR)\n");
91                break;
92        case eAND: printf("cEXP{eAND):\n");
93                 print_exp( p->exp1 );
94                 print_exp( p->exp2 );
95                 printf("-- end of cEXP(eAND)\n");
96                 break;
97        case eEQU: printf("cEXP{eEQU):\n");
98                 print_exp( p->exp1 );
99                 print_exp( p->exp2 );
100                printf("-- end of cEXP(eEQU)\n");
101                break;
102       case eREL: printf("cEXP{eREL): %d\n", p->val);
103                print_exp( p->exp1 );
104                print_exp( p->exp2 );
105                printf("-- end of cEXP(eREL)\n");
106                break;
107       case eADD: printf("cEXP{eADD):\n");
108                print_exp( p->exp1 );
109                print_exp( p->exp2 );
110                printf("-- end of cEXP(eADD)\n");
111                break;
112       case eMINUS: printf("cEXP{eMINUS):\n");
113                  print_exp( p->exp1 );
114                  print_exp( p->exp2 );
115                  printf("-- end of cEXP(eMINUS)\n");
116                  break;
117       case eTIMES: printf("cEXP{eTIMES):\n");
118                  print_exp( p->exp1 );
119                  print_exp( p->exp2 );
120                  printf("-- end of cEXP(eTIMES)\n");
121                  break;
122       case eDIVIDE: printf("cEXP{eDIVIDE):\n");
123                  print_exp( p->exp1 );
124                  print_exp( p->exp2 );
125                  printf("-- end of cEXP(eDIVIDE)\n");
126                  break;
127       case eREMAIN: printf("cEXP{eREMAIN):\n");
```

```
128                 print_exp( p->exp1 );
129                 print_exp( p->exp2 );
130                 printf("-- end of cEXP(eREMAIN)\n");
131                 break;
132       case eNOT: printf("cEXP{eNOT):\n");
133                print_exp( p->exp1 );
134                printf("-- end of cEXP(eADD)\n");
135                break;
136       case eUMINUS: printf("cEXP{eUMINUS):\n");
137                 print_exp( p->exp1 );
138                 printf("-- end of cEXP(eUMINUS)\n");
139                 break;
140       case eNUM: printf("cEXP{eNUM): %d\n", p->val);
141                printf("-- end of cEXP(eNUM)\n");
142                break;
143       case eFCALL: printf("cEXP{eFCALL): %s\n", p->name);
144                  print_exp( p->exp1 );
145                  printf("-- end of cEXP(eFCALL)\n");
146                  break;
147       case eALIST: printf("cEXP{eALIST): %s\n", p->name);
148                  print_exp( p->exp1 );
149                  print_exp( p->next );
150                  printf("-- end of cEXP(eALIST)\n");
151                  break;
152         case eOEXP: printf("cEXP{eOEXP):\n");
153                   print_exp( p->exp1 );
154                   printf("-- end of cEXP(eOEXP)\n");
155                   break;
156         case eEXP: printf("cEXP{eEXP): %s\n", p->name);
157                  print_exp( p->next );
158                  printf("-- end of cEXP(eFCALL)\n");
159                  break;
160         case eLP: printf("cEXP{eLP):\n");
161                   print_exp( p->exp1 );
162                   printf("-- end of cEXP(eLP)\n");
163                   break;
164       default: fprintf(stderr, "******* An error in expressions!\n");
165              break;
166       }
167     }
168   }
169
170   void print_stm ( cSTM* p ) {
171     cEXP *te;
172     cSTM *ts;
173     if( p ) {
174       switch( p->stm_id ) {
175       case sFUNC: printf("cSTM(sFUNC): %s\n", p->exp1->name);
176               print_exp( p->exp1 );
177               print_exp( p->exp2 );
178                 print_stm( p->stm1 );
179                 printf("** end of cSTM(sFUNC)\n");
180                 break;
181       case sSLIST: printf("cSTM(sSLIST):\n");
182                  print_stm( p->stm1 );
183                  print_stm( p->next );
184                  printf("** end of cSTM(sSLIST)\n");
185                  break;
186       case sCSTM: printf("cSTM(sCSTM):\n");
187                 print_stm( p->stm1 );
188                 print_stm( p->stm2 );
189                 printf("** end of cSTM(sCSTM)\n");
190                 break;
191       case sVDLIST: printf("cSTM(sVDLIST):\n");
```

```
192                 print_stm( p->stm1 );
193                 print_stm( p->next );
194                 printf("** end of cSTM(sVDLIST)\n");
195                 break;
196      case sVDEC: printf("cSTM(sVDEC):\n");
197                 print_exp( p->exp1 );
198                 print_exp( p->exp2 );
199                 printf("** end of cSTM(sVDEC)\n");
200                 break;
201      case sISTM: printf("cSTM(sISTM):\n");
202                 print_exp( p->exp1 );
203                 print_stm( p->stm1 );
204                 print_stm( p->stm2 );
205                 printf("** end of cSTM(sISTM)\n");
206                 break;
207      case sWSTM: printf("cSTM(sWSTM):\n");
208                 print_exp( p->exp1 );
209                 print_stm( p->stm1 );
210                 printf("** end of cSTM(sWSTM)\n");
211                 break;
212      case sASTM: printf("cSTM(sASTM): %s\n", p->name);
213                 print_exp( p->exp1 );
214                 printf("** end of cSTM(sASTM)\n");
215                 break;
216      case sBR: printf("cSTM(sBR):\n");
217                printf("** end of cSTM(sBR)\n");
218                break;
219      case sCONT: printf("cSTM(sCONT):\n");
220                 printf("** end of cSTM(sCONT)\n");
221                 break;
222      case sRET: printf("cSTM(sRET):\n");
223                 print_exp( p->exp1 );
224                 printf("** end of cSTM(sRET)\n");
225                 break;
226      case sPRF: printf("cSTM(sPRF):\n");
227                 print_exp( p->exp1 );
228                 print_exp( p->exp2 );
229                 printf("** end of cSTM(sPRF)\n");
230                 break;
231        case sOELS: printf("cSTM(sOELS):\n");
232                 print_stm( p->stm1 );
233                 printf("** end of cSTM(sOELS)\n");
234                   break;
235      default: fprintf(stderr, "******* An error in statements!\n");
236             break;
237      }
238    }
239  }
```

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

```
1    %{
2    #include "minic.h"
3    #include "c_tree.h"
4    #include "c_yacc.h"
5    %}
6
7    NUM [0-9][0-9]*
8    ID  [_A-Za-z][A-Za-z0-9_]*
9    NONNL [^\n]
10   NONNLQ [^"\n]
11
12   %%
13
14   break       {return BR;}
15   continue    {return CONT;}
16   else        {return ELSE;}
17   if              {return IF;}
18   int             {return INT;}
19   return      {return RET;}
20   while       {return WHILE;}
21   printf      {return PRF;}
22   "+"         {return ADD;}
23   "-"         {return MINUS;}
24   "*"         {return TIMES;}
25   "/"         {return DIVIDE;}
26   "%"         {return REMAIN;}
27   "!"         {return NOT;}
28   "?"         {return OPT;}
29   ":"         {return COL;}
30   "="         {return ASSIGN;}
31   ","         {return COMMA;}
32   "<"         {return LT;}
33   ">"         {return GT;}
34   "("         {return LP;}
35   ")"         {return RP;}
36   "{"         {return LBP;}
37   "}"         {return RBP;}
38   "||"        {return OR;}
39   "&&"        {return AND;}
40   "=="        {return EQU;}
41   \"          {return DBQ;}
42   ";"         {return SEMI;}
43   {NUM}       {sscanf(yytext,"%d", &val); yylval.nu = val; return NUM;}
44   {ID}        {sscanf(yytext,"%s", name); yylval.sr = strdup( name ); return ID; }
45   \"{NONNLQ}*\" {sscanf(yytext,"%s", name); yylval.sr = strdup( name ); return STR;}
46   "//"{NONNL}*{/* return COMMENT; */}
47   [ \t\n]     {/* skip white space */}
48   .           {printf("****** A bug is found in MiniC!\n");}
49
50   %%
51
52   int yywrap() {return 1;}
```

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   minic parses OK!
2   cSTM(sFUNC): ComputeFac
3   cEXP(eTYPE):2
4   -- end of cEXP(eTYPE)
5   cEXP(ePLIST):
6   cEXP(ePDEC): num
7   cEXP(eTYPE):2
8   -- end of cEXP(eTYPE)
9   -- end of cEXP(ePDEC)
10  -- end of cEXP(ePLIST)
11  cSTM(sCSTM):
12  cSTM(sVDLIST):
13  cSTM(sVDEC):
14  cEXP(eTYPE):2
15  -- end of cEXP(eTYPE)
16  cEXP{eID): num_aux
17  -- end of cEXP(eID)
18  ** end of cSTM(sVDEC)
19  ** end of cSTM(sVDLIST)
20  cSTM(sSLIST):
21  cSTM(sISTM):
22  cEXP{eREL): 0
23  cEXP{eID): num
24  -- end of cEXP(eID)
25  cEXP{eNUM): 1
26  -- end of cEXP(eNUM)
27  -- end of cEXP(eREL)
28  cSTM(sASTM): num_aux
29  cEXP{eNUM): 1
30  -- end of cEXP(eNUM)
31  ** end of cSTM(sASTM)
32  cSTM(sOELS):
33  cSTM(sASTM): num_aux
34  cEXP{eTIMES):
35  cEXP{eID): num
36  -- end of cEXP(eID)
37  cEXP{eFCALL): ComputeFac
38  cEXP{eMINUS):
39  cEXP{eID): num
40  -- end of cEXP(eID)
41  cEXP{eNUM): 1
42  -- end of cEXP(eNUM)
43  -- end of cEXP(eMINUS)
44  -- end of cEXP(eFCALL)
45  -- end of cEXP(eTIMES)
46  ** end of cSTM(sASTM)
47  ** end of cSTM(sOELS)
48  ** end of cSTM(sISTM)
49  cSTM(sSLIST):
50  cSTM(sRET):
51  cEXP{eID): num_aux
52  -- end of cEXP(eID)
53  ** end of cSTM(sRET)
54  ** end of cSTM(sSLIST)
55  ** end of cSTM(sSLIST)
56  ** end of cSTM(sCSTM)
57  ** end of cSTM(sFUNC)
58  MiniC successfully builds a parse tree for
59  test.c!
```

Discussion

真心覺得三次作業下來，這次雖稱不上難，但卻能說是最煩最雜的，而更需要耐心與細心，尤其樹狀結構的撰寫牽扯到了指標，一不小心就會獲得滿滿的core dump啊！幸好最後總算成功了，不然整個超想砸電腦啊！