

前言：

之前的「OurC grammar 2010-04-04」版是硬掰出來的，曾經小幅修改過數次，但都止於頭痛醫頭、腳痛醫腳。此次的修改是“確立基本精神”。刪掉 2010-04-04 版的五條規則、另增加兩條規則、並修改七條規則。

此 2016-05-05 版所要確立的 OurC grammar 的基本精神如下：

- * expression 是由一個或多個 basic_expression 所組成，其間以','隔開來。
- * basic_expression 是由 unary_expression 所組成，其間以各種 operator 隔開來，而各種 operator 之間有其優先順序。
- * 所謂的“各種 operator”，包括了 conditional operator，即'?' ':'。
- * unary_expression 有以下三種：
 - (a) 有 sign('+', '-', '!')開頭者；
 - (b) 無 sign 開頭者；
 - (c) 有 PP/MM(即'++'與'--')開頭者。
- * ID 與 ID '[' expression ']' 是以上 (a) (b) (c) 三者皆適用。所有其它 case 都只適用於 (a) 與 (b)，包括
 - ※ ID '(' actual_parameter_list ')' 註：這是個 function 的呼叫
 - ※ '(' expression ')'
 - ※ CONSTANT
- * ID 與 ID '[' expression ']' 不只適用於 (a) (b) (c) 三者，也可以有 PP/MM 出現於其後。
- * 但「sign 的出現」與「PP/MM 的出現」必須遵守以下的規矩：
 - ※ 如果有 sign 出現於 ID 或 ID '[' expression ']' 之前、就不可以有 PP/MM 出現於同一 ID 或 ID '[' expression ']' 的前或後。反之亦然。
 - ※ PP/MM 頂多只能出現於 ID 或 ID '[' expression ']' 的前或後一次。
 - ※ sign 可以出現隨便多少次。

註：romce_and_romloe

與

rest_of_maybe_conditional_exp_and_rest_of_maybe_logical_OR_exp

是同一個規則，只是因為後者名字太長，所以用 romce_and_romloe 作為後者的簡稱。

```
/*
 * OurC - the grammar (May 5th, 2016)
 *
 * 缺陷說明： 'a++b' 會被視為 error，因為 '++' 會被視為 'PP'
 *           要 'a+ +b' 才不會被視為 error。
 *
 */
```

// the lexical part

%token Identifier

```

%token Constant    // e.g., 35, 35.67, 'a', "Hi, there", true, false
                  //      .35, 35., 0023
%token INT         // int
%token FLOAT       // float
%token CHAR        // char
%token BOOL        // bool
%token STRING      // string <----- 注意全小寫!
%token VOID        // void
%token IF          // if
%token ELSE        // else
%token WHILE       // while
%token DO          // do
%token RETURN      // return
%token '('
%token ')'
%token '['
%token ']'
%token '{'
%token '}'
%token '+'
%token '-'
%token '*'
%token '/'
%token '%'
%token '^'
%token '>'
%token '<'
%token GE          // >=
%token LE          // <=
%token EQ          // ==
%token NEQ         // !=
%token '&'
%token '|'
%token '='
%token '!'
%token AND         // &&
%token OR          // ||
%token PE          // +=
%token ME          // -=
%token TE          // *=
%token DE          // /=
%token RE          // %=
%token PP          // ++
%token MM          // --
%token RS          // >>
%token LS          // <<
%token ';'
%token ','
%token '?'
%token ':'

/*
 * (僅供參考) precedence (lower ones are given higher precedence) and associativity
 */

%left ' ','
%right '=' PE ME TE DE RE

```

```

%right  '?' '+' ':'
%left   OR
%left   AND
%left   '|'
%left   '^'
%left   '&'
%left   EQ NEQ
%left   '<' '>' GE LE
%left   '+' '-'
%left   '*' '/' '%'
%right  PP MM sign      // sign is '+' or '-' or '!'
%% // the syntactical part (in EBNF)

user_input
    : ( definition | statement ) { definition | statement }

definition
    :          VOID Identifier function_definition_without_ID
    | type_specifier Identifier function_definition_or_declarators

type_specifier
    : INT | CHAR | FLOAT | STRING | BOOL

function_definition_or_declarators
    : function_definition_without_ID
    | rest_of_declarators

rest_of_declarators
    : [ '[' Constant ']' ]
    { ',' Identifier [ '[' Constant ']' ] } ';'

function_definition_without_ID
    : '(' [ VOID | formal_parameter_list ] ')' compound_statement

formal_parameter_list
    : type_specifier [ '&' ] Identifier [ '[' Constant ']' ]
    { ',' type_specifier [ '&' ] Identifier [ '[' Constant ']' ] }

compound_statement
    : '{' { declaration | statement } '}'

declaration
    : type_specifier Identifier rest_of_declarators

statement
    : ';'          // the null statement
    | expression ';' /* expression here should not be empty */
    | RETURN [ expression ] ';'
    | compound_statement
    | IF '(' expression ')' statement [ ELSE statement ]
    | WHILE '(' expression ')' statement
    | DO statement WHILE '(' expression ')' ';'

expression
    : basic_expression { ',' basic_expression }

basic_expression
    : Identifier rest_of_Identifier_started_basic_exp

```

```

| ( PP | MM ) Identifier rest_of_PPMM_Identifier_started_basic_exp
| sign { sign } signed_unary_exp romce_and_romloe
| ( Constant | '(' expression ')' ) romce_and_romloe

rest_of_Identifier_started_basic_exp
: [ '[' expression ']' ]
  ( assignment_operator basic_expression
    |
    [ PP | MM ] romce_and_romloe
  )
| '(' [ actual_parameter_list ] ')' romce_and_romloe

rest_of_PPMM_Identifier_started_basic_exp
: [ '[' expression ']' ] romce_and_romloe

sign
: '+' | '-' | '!'

actual_parameter_list
: basic_expression { ',' basic_expression }

assignment_operator
: '=' | TE | DE | RE | PE | ME

rest_of_maybe_conditional_exp_and_rest_of_maybe_logical_OR_exp // 即 romce_and_romloe
: rest_of_maybe_logical_OR_exp [ '?' basic_expression ':' basic_expression ]

rest_of_maybe_logical_OR_exp
: rest_of_maybe_logical_AND_exp { OR maybe_logical_AND_exp }

maybe_logical_AND_exp
: maybe_bit_OR_exp { AND maybe_bit_OR_exp }
rest_of_maybe_logical_AND_exp
: rest_of_maybe_bit_OR_exp { AND maybe_bit_OR_exp }

maybe_bit_OR_exp
: maybe_bit_ex_OR_exp { '|' maybe_bit_ex_OR_exp }
rest_of_maybe_bit_OR_exp
: rest_of_maybe_bit_ex_OR_exp { '|' maybe_bit_ex_OR_exp }

maybe_bit_ex_OR_exp
: maybe_bit_AND_exp { '^' maybe_bit_AND_exp }
rest_of_maybe_bit_ex_OR_exp
: rest_of_maybe_bit_AND_exp { '^' maybe_bit_AND_exp }

maybe_bit_AND_exp
: maybe_equality_exp { '&' maybe_equality_exp }
rest_of_maybe_bit_AND_exp
: rest_of_maybe_equality_exp { '&' maybe_equality_exp }

maybe_equality_exp
: maybe_relational_exp
  { ( EQ | NEQ ) maybe_relational_exp }
rest_of_maybe_equality_exp
: rest_of_maybe_relational_exp
  { ( EQ | NEQ ) maybe_relational_exp }

```

```

maybe_relational_exp
    : maybe_shift_exp
      { ( '<' | '>' | LE | GE ) maybe_shift_exp }
rest_of_maybe_relational_exp
    : rest_of_maybe_shift_exp
      { ( '<' | '>' | LE | GE ) maybe_shift_exp }

maybe_shift_exp
    : maybe_additive_exp { ( LS | RS ) maybe_additive_exp }
rest_of_maybe_shift_exp
    : rest_of_maybe_additive_exp { ( LS | RS ) maybe_additive_exp }

maybe_additive_exp
    : maybe_mult_exp { ( '+' | '-' ) maybe_mult_exp }
rest_of_maybe_additive_exp
    : rest_of_maybe_mult_exp { ( '+' | '-' ) maybe_mult_exp }

maybe_mult_exp
    : unary_exp rest_of_maybe_mult_exp
rest_of_maybe_mult_exp
    : { ( '*' | '/' | '%' ) unary_exp } /* could be empty ! */

unary_exp
    : sign { sign } signed_unary_exp
    | unsigned_unary_exp
    | ( PP | MM ) Identifier [ '[' expression ']' ]

signed_unary_exp
    : Identifier [ '(' [ actual_parameter_list ] ')'
      |
      | '[' expression ']'
      ]
    | Constant
    | '(' expression ')'

unsigned_unary_exp
    : Identifier [ '(' [ actual_parameter_list ] ')'
      |
      | '[' expression ']' [ ( PP | MM ) ]
      ]
    | Constant
    | '(' expression ')'

```