

● Function Definition

```

39
40 Function_Definition:  Non_Void_Type_Specifier ID '(' ')' Compound_Statement
41                      |  Non_Void_Type_Specifier ID '(' Parameter_List ')' Compound_Statement
42                      |  VOID ID '(' ')' Compound_Statement
43                      |  VOID ID '(' Parameter_List ')' Compound_Statement
44                      ;

```

如圖所見，函式定義是由 Type(即 void, int, double, char, bool)接上一個 ID，由括號圍起來的參數們(0 或多個參數)，最後有一個 Compound Statement。

下圖顯示，參數可以是一般的變數(Ex: int a)，或是 Array(Ex: int a[1])。

```

86 Parameter_List:      Parameter
87                   |  Parameter_List ',' Parameter
88                   ;
89
90 Parameter:          Non_Void_Type_Specifier ID
91                   |  Non_Void_Type_Specifier Array
92                   ;

```

下圖顯示，Array 可以是多維陣列，而括號裡面只能放入非負的整數常數。

```

94 Array:             ID Array_Paranthesis
95                   ;
96
97 Array_Paranthesis:  '[' INT_CONSTANT '['
98                   |  Array_Paranthesis '[' INT_CONSTANT '['
99                   ;

```

● Declaration

我將 Declaration 分為三種，Function, Const 以及 Normal。

```

46 Declaration:       Function_Declaration
47                   |  Const_Declaration
48                   |  Normal_Declaration
49                   ;

```

首先是 **Function Declaration**，這和上一個 **Function Definition** 幾乎一樣，只是他後面接的是分號而不是 **compound statement**。

```

51 Function_Declaration:  Non_Void_Type_Specifier ID '(' ')' ';'
52                       |  Non_Void_Type_Specifier ID '(' Parameter_List ')' ';'
53                       |  VOID ID '(' ')' ';'
54                       |  VOID ID '(' Parameter_List ')' ';'
55                       ;

```

再來是 **Const Declaration**，這類的宣告一定要初始化，不能單獨宣告一個 **Const** 變數而不給值，而初始化給的值限定為常數(**Bool, Int, Double, String, Char** 型態的常數)。另外可以用逗號隔開宣告多個 **Const** 變數，但是 **spec** 中限定不能用在 **Array type** 上。

```

57 Const_Declaration:  CONST Non_Void_Type_Specifier Const_Declarator_List ';'
58                       ;
59
60 Const_Declarator_List:  Const_Declarator
61                       |  Const_Declarator_List ',' Const_Declarator
62                       ;
63
64 Const_Declarator:      ID '=' INT_CONSTANT
65                       |  ID '=' DOUBLE_CONSTANT
66                       |  ID '=' CHAR_CONSTANT
67                       |  ID '=' STRING_CONSTANT
68                       |  ID '=' TRUE
69                       |  ID '=' FALSE
70                       ;

```

最後是 **Normal Declaration**，一樣可以一次宣告多個變數，並且可以用在 **Array type** 上面。也一樣可以在宣告時初始化，一般的變數的初始值可以給他 **Expression**，而 **Array** 的初始值則使用特殊的語法給值。

```

72 Normal_Declaration:  Non_Void_Type_Specifier Normal_Declarator_List ';'
73                       ;
74
75 Normal_Declarator_List:  Normal_Declarator
76                       |  Normal_Declarator_List ',' Normal_Declarator
77                       ;
78
79 Normal_Declarator:      ID
80                       |  Array
81                       |  ID '=' Expression
82                       |  Array '=' Array_Content
83                       ;

```

Array 的初始值只能用以下的形式給予，由大括號圍起來，裡面放入 0 或多個 Expression。

```

101 Array_Content:    '{' '}'
102                  |    '{' Expression_List '}'
103                  ;

```

● Compound Statement

Compound Statement 由兩個大括號組成，大括號中間可以放入 0 或多個 Declaration(上面提到的)和 Statement，我並沒有限制所有 Declaration 一定要在 Statement 前面，Declaration 和 Statement 的順序可以任意安排。

除此之外我根據 C99 Standard，允許 Function Declaration 出現在 Compound Statement 中，並不只限制 Const Declaration 或是 Normal Declaration。

```

124 Compound_Statement:  '{' '}'
125                      |    '{' Block_Item_List '}'
126                      ;
127
128 Block_Item_List:      Block_Item
129                      |    Block_Item_List Block_Item
130                      ;
131
132 Block_Item:           Declaration
133                      |    Statement
134                      ;

```

● Statement

依據 Spec，我歸類出五種 Statement。

```

136 Statement:      Simple_Statement
137                |      Switch_Statement
138                |      Selection_Statement
139                |      Iteration_Statement
140                |      Jump_Statement
141                ;

```

Simple Statement 如下，相當簡單明瞭，就是將一個 var 賦值。

```

143 Simple_Statement:  Var '=' Expression ';'
144                ;

```

```

105 Array_Expression: '[' Expression ']'
106                |      Array_Expression '[' Expression ']'
107                ;
108
109 Var:      ID
110        |      ID Array_Expression
111        ;

```

Switch Statement 照著 Spec 的規定，switch 後面的括號內只能放入 ID，而 case 後面接的只能是 Int 或是 Char 的常數。Case 和 default 後面是 0 或多個 statement，整個 switch 之中至少要有一個 case，而若有 default 的話他必須在最後面。

```

146 Switch_Statement:    SWITCH '(' ID ')' '{' Switch_Content '}'
147                      ;
148
149 Switch_Content:      Case_List
150                      | Case_List Default_Content
151                      ;
152
153 Case_List:           Case_Content
154                      | Case_List Case_Content
155                      ;
156
157 Case_Content:        CASE INT_CONSTANT ':' Statement_List
158                      | CASE CHAR_CONSTANT ':' Statement_List
159                      | CASE INT_CONSTANT ':'
160                      | CASE CHAR_CONSTANT ':'
161                      ;
162
163 Default_Content:     DEFAULT ':' Statement_List
164                      | DEFAULT ':'
165                      ;

```

Selection Statement 中就是 if-else 判斷式，如下圖

```

171 Selection_Statement:  IF '(' Expression ')' Compound_Statement
172                      | IF '(' Expression ')' Compound_Statement ELSE Compound_Statement
173                      ;

```

Iteration Statement 包含 while 以及 for 迴圈，其中 for 迴圈的括號裡面可以放 0 到 3 個 expression，由分號隔開，這裡我用 Expression Statement 來解決，Expression Statement 是一個分號或是一個 Expression 加上分號。

```

174
175 Iteration_Statement:  WHILE '(' Expression ')' Compound_Statement
176                      | DO Compound_Statement WHILE '(' Expression ')' ';'
177                      | FOR '(' Expression_Statement Expression_Statement ')' Compound_Statement
178                      | FOR '(' Expression_Statement Expression_Statement Expression ')' Compound_Statement
179                      ;

```

Jump Statement 則是 continue, break, return 的集合。

```

185  Jump_Statement:      CONTINUE ';'
186                      |      BREAK ';'
187                      |      RETURN Expression ';'
188                      |      RETURN ';'
189                      ;

```

● Expression

依據 Spec 提供的 Precedence 順序定義了以下的 rule，另外我根據 C99 Standard 額外多實作了 Conditional Expression。

```

196  Expression:  Conditional_Expression
197              ;

```

首先是 Precedence 最低的 Conditional Expression。

```

199  Conditional_Expression:  Logical_Or_Expression
200                          |  Logical_Or_Expression '?' Expression ':' Conditional_Expression
201                          ;

```

第二個是 Spec 中 Precedence 最低的 Logical Or Expression。

```

203  Logical_Or_Expression:  Logical_And_Expression
204                          |  Logical_Or_Expression OR_OP Logical_And_Expression
205                          ;

```

下一個是 Logical And Expression

```

207  Logical_And_Expression:  Not_Expression
208                          |  Logical_And_Expression AND_OP Not_Expression
209                          ;

```

下一個是 Not Expression，這邊可以看得出來是右結合。

```

211  Not_Expression:  Relational_Expression
212                  |  NOT_OP Relational_Expression
213                  ;

```

下一個是比較關係的 Relational Expression

```

215 Relational_Expression:    Additive_Expression
216                          | Relational_Expression LT_OP Additive_Expression
217                          | Relational_Expression LE_OP Additive_Expression
218                          | Relational_Expression GT_OP Additive_Expression
219                          | Relational_Expression GE_OP Additive_Expression
220                          | Relational_Expression EQUAL_OP Additive_Expression
221                          | Relational_Expression NEQUAL_OP Additive_Expression
222                          ;

```

下一個是加減乘除的 Expression，乘除的優先順序高於加減

```

224 Additive_Expression:    Multiplicative_Expression
225                          | Additive_Expression PLUS_OP Multiplicative_Expression
226                          | Additive_Expression MINUS_OP Multiplicative_Expression
227                          ;
228
229 Multiplicative_Expression:  Unary_Expression
230                          | Multiplicative_Expression MUL_OP Unary_Expression
231                          | Multiplicative_Expression DIV_OP Unary_Expression
232                          | Multiplicative_Expression MOD_OP Unary_Expression
233                          ;

```

最後是 Unary 和 Postfix 的 Expression，Unary 是左結合，而且優先順序最高。

```

235 Unary_Expression:    Postfix_Expression
236                      | MINUS_OP Postfix_Expression
237                      ;
238
239 Postfix_Expression:    Primary_Expression
240                      | Primary_Expression PLUSPLUS_OP
241                      | Primary_Expression MINUSMINUS_OP
242                      ;

```

根據 Spec 定義，Operand 的合法組成有 Literal Constant(Char, Bool, Int, Double, String 的常數)，以及 Var(上面提到的)，還有呼叫 function。將 Expression 包上括號也可以當作是 Operand 的一種。


```
244 Primary_Expression: Var
245                       | INT_CONSTANT
246                       | DOUBLE_CONSTANT
247                       | CHAR_CONSTANT
248                       | STRING_CONSTANT
249                       | TRUE
250                       | FALSE
251                       | '(' Expression ')'
252                       | ID '(' ')'
253                       | ID '(' Expression_List ')'
254                       ;
```

3. Change to Scanner.

第一步是在 Definition Section 裡 include “y.tab.h”，這樣才能取得 parser.y 中 token 的定義。

再來是在 Keyword 的部分以及標點符號和運算符號的部分，依據讀到的東西，回傳對應的 token。

```
122 {KEY} {
123     /* Check pragma */
124     if(token0n == 1)    printf("#key:%s\n",yytext);
125
126     strcpy(&(buf[bufIndex]), yytext);
127     bufIndex += yyleng;
128
129     /* Return to parser */
130     if(strcmp(yytext, "void") == 0) return VOID;
131     if(strcmp(yytext, "int") == 0) return INT;
132     if(strcmp(yytext, "double") == 0) return DOUBLE;
133     if(strcmp(yytext, "bool") == 0) return BOOL;
134     if(strcmp(yytext, "char") == 0) return CHAR;
135     if(strcmp(yytext, "null") == 0) return NUL;
136     if(strcmp(yytext, "for") == 0) return FOR;
137     if(strcmp(yytext, "while") == 0) return WHILE;
138     if(strcmp(yytext, "do") == 0) return DO;
139     if(strcmp(yytext, "if") == 0) return IF;
140     if(strcmp(yytext, "else") == 0) return ELSE;
141     if(strcmp(yytext, "switch") == 0) return SWITCH;
142     if(strcmp(yytext, "return") == 0) return RETURN;
143     if(strcmp(yytext, "break") == 0) return BREAK;
144     if(strcmp(yytext, "continue") == 0) return CONTINUE;
145     if(strcmp(yytext, "const") == 0) return CONST;
146     if(strcmp(yytext, "struct") == 0) return STRUCT;
147     if(strcmp(yytext, "true") == 0) return TRUE;
148     if(strcmp(yytext, "false") == 0) return FALSE;
149     if(strcmp(yytext, "case") == 0) return CASE;
150     if(strcmp(yytext, "default") == 0) return DEFAULT;
```

```

199 {PUNC}  {
200         /* Check pragma */
201         if(tokenOn == 1)    printf("#punc:%s\n",yytext);
202
203         strcpy(&(buf[bufIndex]), yytext);
204         bufIndex += yyleng;
205
206         /* Return to Parser */
207         if(strcmp(yytext, ":") == 0) return ':';
208         if(strcmp(yytext, ";") == 0) return ';';
209         if(strcmp(yytext, ",") == 0) return ',';
210         if(strcmp(yytext, ".") == 0) return '.';
211         if(strcmp(yytext, "[") == 0) return '[';
212         if(strcmp(yytext, "]") == 0) return ']';
213         if(strcmp(yytext, "(") == 0) return '(';
214         if(strcmp(yytext, ")") == 0) return ')';
215         if(strcmp(yytext, "{") == 0) return '{';
216         if(strcmp(yytext, "}") == 0) return '}';
217         if(strcmp(yytext, "?") == 0) return '?';
218     }

```

```

{OP}  {
    /* Check pragma */
    if(tokenOn == 1)    printf("#op:%s\n",yytext);

    strcpy(&(buf[bufIndex]), yytext);
    bufIndex += yyleng;

    /* Return to Parser */
    if(strcmp(yytext, "+") == 0) return PLUS_OP;
    if(strcmp(yytext, "-") == 0) return MINUS_OP;
    if(strcmp(yytext, "*") == 0) return MUL_OP;
    if(strcmp(yytext, "/") == 0) return DIV_OP;
    if(strcmp(yytext, "%") == 0) return MOD_OP;
    if(strcmp(yytext, "++") == 0) return PLUSPLUS_OP;
    if(strcmp(yytext, "--") == 0) return MINUSMINUS_OP;
    if(strcmp(yytext, "<") == 0) return LT_OP;
    if(strcmp(yytext, "<=") == 0) return LE_OP;
    if(strcmp(yytext, ">") == 0) return GT_OP;
    if(strcmp(yytext, ">=") == 0) return GE_OP;
    if(strcmp(yytext, "==" == 0) return EQUAL_OP;
    if(strcmp(yytext, "!=" == 0) return NEQUAL_OP;
    if(strcmp(yytext, "=") == 0) return '=';
    if(strcmp(yytext, "&&") == 0) return AND_OP;
    if(strcmp(yytext, "||") == 0) return OR_OP;
    if(strcmp(yytext, "!") == 0) return NOT_OP;
    if(strcmp(yytext, "&") == 0) return '&';
}

```

還有讀到的 Constant(Int, Double, Char, String, Bool)還有 ID，也要回傳，要注意的是，原本有區分科學記號和一般小數，但是這邊回傳都是 Double_Constant。

```
177 {DOUBLE} {
178     /* Check pragma */
179     if(token0n == 1)    printf("#double:%s\n",yytext);
180
181     strcpy(&(buf[bufIndex]), yytext);
182     bufIndex += yyleng;
183
184     /* Return to Parser */
185     return DOUBLE_CONSTANT;
186 }
187
188 {SCI} {
189     /* Check pragma */
190     if(token0n == 1)    printf("#sci:%s\n",yytext);
191
192     strcpy(&(buf[bufIndex]), yytext);
193     bufIndex += yyleng;
194
195     /* Return to Parser */
196     return DOUBLE_CONSTANT;
197 }
```

```
295 ~ {ID} {
296     /* Check pragma */
297     if(token0n == 1)    printf("#id:%s\n",yytext);
298
299     strcpy(&(buf[bufIndex]), yytext);
300     bufIndex += yyleng;
301
302     /* Return to Parser */
303     return ID;
304 }
```

```

263 {STR} {
264     /* Extract String */
265     char str[1000];
266     if(yytext[0] == 'L')
267     {
268         strcpy(str,yytext+2); /* The first " is gone */
269         str[yyleng-3] = 0; /* The last " is gone */
270     }
271     else
272     {
273         strcpy(str,yytext+1);
274         str[yyleng-2] = 0; /* The last " is gone */
275     }
276
277
278     /* Error Detect? */
279
280     /* Check pragma */
281     if(tokenOn == 1)    printf("#string:%s\n",str);
282
283     strcpy(&(buf[bufIndex]), yytext);
284     bufIndex += yyleng;
285
286     /* Return to Parser */
287     return STRING_CONSTANT;
288 }

```

最後下面要把原本的 main function 註解掉。

```

330  /* C code Section */
331  /*
332  int main(int argc, char* argv[])
333  {
334      ++argv; --argc;
335      yyin = ( argc > 0 ) ? fopen(argv[0], "r") : stdin;
336
337      yylex();
338
339      return 0;
340  }
341  */

```

4. Advance Part

- **Compound Statement** 裡不用 **Declaration** 優先於 **Statement** , 可以先有 **Statement** 才有其他 **Declaration** 。
- **Local Scope(Compound Statement)**中可放入 **Function Declaration** , 不限定於 **const** 變數或一般變數的宣告。
- **Conditional Expression** 。
- **Normal Declaration** 可以 **invoke function** 來給予初始值。

我提供的 testcase 檔名為 `advance.c` , 內容如下

```
1  #pragma token off
2
3  int main()
4  {
5      //Statement first
6      a = 1;
7      //Declaration second
8      int a = 1;
9
10     //Function Declaration in Compound Statement
11     int a(int b);
12
13     // Conditional Expression
14     a = (a > 1) ? 1 : 0;
15
16     // Initialization using invoking function
17     int c = a();
18 }
19
```

而執行結果如下

```
root@1028ecdd1d07:/CTF/Compiler# ./parser < advance.c
1:#pragma token off
2:
3:int main()
4:{
5:    //Statement first
6:    a = 1;
7:    //Declaration second
8:    int a = 1;
9:
10:    //Function Declaration in Compound Statement
11:    int a(int b);
12:
13:    // Conditional Expression
14:    a = (a > 1) ? 1 : 0;
15:
16:    // Initialization using invoking function
17:    int c = a();
18:}
No syntax error!
```