

程式語言與編譯器報告

作業一

(Programming Assignment1 - MiniJ)

系級：資工二

組員學號&姓名：

410921202 林芷萱

410921309 陳采瑜

一、 The problem description

此次作業要求我們使用 flex 及 bison 完成 MiniJ 的前端編譯器，我們需要完成的部分有 .l 檔(lex)及 .y 檔(yacc)。先使用 flex 生成 scanner(掃描器)，再使用 bison 讀取 scanner 分析出來的 token(記號)建立並返回一顆 “Parse Tree” (也稱作 syntax tree)，詳細的關係圖如下圖一。



● 圖一、scanner 及 parser 關係圖

在這次作業中，當完成 .l 檔及 .y 檔後，使用 CMD(命令提示字元)叫出 flex 打上參數 -o 將 .l 檔轉成 .c 檔，並叫出 bison 打上參數 -d，這個參數的意思是將 .y 檔編譯的結果分拆成 .c 檔及 .h 檔，此時應有三個 .c 檔(另一個為作業檔案夾裡附上的主程式)及一個 .h 檔，再叫出 gcc 將三個 .c 檔都轉成 .o 檔後，將三者 link 在一起，最終會產生一個叫做 “mjparse” 的 .exe 檔。

*補充 1：flex 的前身為 lex；bison 的前身為 yacc。

*補充 2：flex 的作用為 lexical analysis(詞法分析)；

bison 的作用為 syntactic analysis(語法分析)。

二、 Highlight of the way you write the program

mini_j_lex.l

這個檔案主要參考的資料有：

1. lex 維基百科(<https://zh.wikipedia.org/wiki/Lex>)
2. The Fundamentals of lex Rules 中的 lex Operators 部分(<https://docs.oracle.com/cd/E19504-01/802-5880/lex-6/index.html>)
3. Lex Practice(<https://www.epaperpress.com/lexandyacc/pr1.html>)

此檔案的結構分為三大塊，以%%來區隔：

定義區塊 // 用來定義巨集以及匯入 C 寫成的標頭檔

%%

規則區塊 // 將樣式(patterns)與 C 的陳述串連在一起

%%

C 程式碼區塊 // C 的陳述與函式

以下是關於 lex 結構的詳細說明：

➤ **定義區塊**可細拆分成兩個區塊：

(1). %{ 資料結構&函式定義&匯入標頭檔}%

(2). 寫在(1)下方的 token 名稱及其匹配方式，舉例：

NONNL [^\n] // NONNL：「不是換行」，^為 NOT 的意思

➤ **規則區塊**是最重要的區塊，當 lexer 看到輸入裡面有合乎給定的樣式時，則會操作相對應的 C 程式碼。

在這次作業中我們要在這裡寫 10 個 rules，其中

"/" {NONNL}* { /* DO NOTHING */ }

因為在此要引用定義區塊的 NONNL，故在此須將其用 {} 框起來。

在寫 code 的時候透過參考資料來知道 lex 的編譯規則，以完成作業需要完成的部分，下圖為根據參考資料整理出來.l 檔中主要會用到的部分：

pattern	含義
A-Z, a-z, 0-9	構成了部分模式的字元和數字
-	用來指定範圍(ASCII code需要連續)
[]	字符集合
*	匹配0個或多個。ex:ab*為a, ab, abb, ...
+	匹配1個或多個。ex:ab+為ab, abb, abb, ...
^	否定(NOT)
" 符號"	字符裡面的字面含義

● 圖二、lex 常規表達式

mini_j_parse.y

這個檔案主要參考的資料有：

1. Bison Grammar Files
(http://web.mit.edu/gnu/doc/html/bison_6.html#SEC34)
2. lex yacc 學習(<https://iter01.com/178869.html>)
3. 以 lex/yacc 實作算式計算機中將 EBNF 表示式改寫成 BNF 再改寫成給 yacc 看的定義檔的部分
(<http://good-ed.blogspot.com/2010/04/lex yacc.html>)

yacc 的結構跟 lex 非常類似，在規則區塊的部分 lex 放置的 rules 就是每個正規表示式要對應的動作，一般是返回一個 token；而 yacc 放置的 rules 就是滿足一個語法描述時要執行的動作。

作業的要求為將 The MiniJ Grammar(EBNF 表示式寫成)改寫成給 yacc 看的定義檔。在此我的作法為參考資料 3 中的做法，以下為我的解題步驟：

```
// EBNF表示式
Statement → lbp Statement* rbp

// BNF表示式
statement := lbp statements rbp
statements := statement statements
```

● 圖三、stmt 改寫部分

```
// EBNF表示式
ExpList → Exp ExpRest*
ExpRest → comma Exp

// BNF表示式
explist := explist exprest exprest
exprest := comma explist exprest
```

● 圖四、exp 及 expr 改寫部分

再將其改為符合 Bison 文法規則的格式，其文法規則一般形式為：

```
result: rule1-components...
      | rule2-components...
      ...
      ;
```

三、 The program listing

mini_j_lex.l 程式碼

```
%{
#include "mini_j.h"
#include "mini_j_parse.h"
%}

ID [A-Za-z][A-Za-z0-9_]*
LIT [0-9][0-9]*
NONNL [^\n]

%%

class {return CLASS;}
public {return PUB;}
static {return STATIC;}
String {return STR;}
void {return VOID;}
main {return MAIN;}
int {return INT;}
if {return IF;}
else {return ELSE;}
while {return WHILE;}
new {return NEW;}
return {return RETURN;}
this {return THIS;}
true {return TRUE;}
false {return FALSE;}
"&&" {return AND;}
"<" {return LT;}
"<=" {return LE;}
"+" {return ADD;}
"-" {return MINUS;}
"*" {return TIMES;}
"(" {return LP;}
")" {return RP;}
"{" {return LBP;}
"}" {return RBP;}
"," {return COMMA;}
"." {return DOT;}

System.Out.println {return PRINT;}
"||" {return OR;}
"==" {return EQ;}
"[" {return LSP;}
"]" {return RSP;}
";" {return SEMI;}
"=" {return ASSIGN;}
"//" {NONNL}* { /* DO NOTHING */ }
{ID} {scanf("%s",name); return (ID);}
{LIT} {return (LIT);}

[ \t\n] {/* skip BLANK */}
. {/* skip redundant characters */}

%%

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

mini_j_parse.y 程式碼

```
// C declarations
%{
    #include <stdio.h>
    #include <stdlib.h>
    #include "mini_j.h"
    #include "mini_j_parse.h"
}%

// Bison declarations

// token:Bison will convert it into a #define directive in the parser

%token CLASS PUB STATIC
%left AND OR NOT
%left LT LE EQ
%left ADD MINUS
%left TIMES
%token LBP RBP LSP RSP LP RP
%token INT BOOLEAN
%token IF ELSE
%token WHILE PRINT
%token ASSIGN
%token VOID MAIN STR
%token RETURN
%token SEMI COMMA
%token THIS NEW DOT
%token ID LIT TRUE FALSE

%expect 24

// Grammar rules
%%
prog      :      mainc cdcls
           { printf("Program -> MainClass ClassDecl*\n");
             printf("Parsed OK!\n"); }
           |
           { printf("***** Parsing failed!\n"); }
           ;

mainc     :      CLASS ID LBP PUB STATIC VOID MAIN LP STR LSP RSP ID RP LBP stmts
RBP RBP
           { printf("MainClass -> class id lbp public static void main lp
string lsp rsp id rp lbp Statemet* rbp rbp\n"); }
           ;
//class類別宣告
cdcls    :      cdcl cdcls
           { printf("(for ClassDecl*) cdcls : cdcl cdcls\n"); }
           |
           { printf("(for ClassDecl*) cdcls : \n"); }
           ;

cdcl      :      CLASS ID LBP vdcls mdcls RBP // LBP { RBP }
rbp\n"); }
           { printf("ClassDecl -> class id lbp VarDecl* MethodDecl*
rbp\n"); }
           ;

//參數宣告
vdcls    :      vdcl vdcls
```

```

        { printf("(for VarDecl*) vdcls : vdcl vdcls\n"); }
    |
        { printf("(for VarDecl*) vdcls : \n"); }
    ;

vdcl    :    type ID SEMI
        { printf("VarDecl -> Type id semi\n"); }
    ;

//函式
mdcls   :    mdcl mdcls
        { printf("(for MethodDecl*) mdcls : mdcl mdcls\n"); }
    |
        { printf("(for MethodDecl*) mdcls : \n"); }
    ;

mdcl    :    PUB type ID LP formals RP LBP vdcls stmts RETURN exp SEMI RBP
        { printf("MethodDecl -> public Type id lp FormalList rp lbp
Statements* return Exp semi rbp\n"); }
    ;

formals :    type ID frest
        { printf("FormalList -> Type id FormalRest*\n"); }
    |
        { printf("FormalList -> \n"); }
    ;

frest   :    COMMA type ID frest
        { printf("FormalRest -> comma Type id FormalRest\n"); }
    |
        { printf("FormalRest -> \n"); }
    ;

//-----
//型別
type    :    INT LSP RSP
        { printf("Type -> int lsp rsp\n"); }
    | BOOLEAN
        { printf("Type -> boolean\n"); }
    | INT
        { printf("Type -> int\n"); }
    | ID
        { printf("Type -> id\n"); }
    ;

//陳述
stmts   :    stmt stmts
        { printf("(for Statement*) stmts : stmt stmts\n"); }
    |
        { printf("(for Statement*) stmts : \n"); }
    ;

stmt    :    LBP stmts RBP
        { printf("Statement -> lbp Statement* rbp\n"); }
    | IF LP exp RP stmt ELSE stmt
        { printf("Statement -> if lp Exp rp Statement else
Statement\n"); }
    | WHILE LP exp RP stmt
        { printf("Statement -> while lp Exp rp Statement\n"); }
    | PRINT LP exp RP SEMI

```

```

        { printf("Statement -> print lp Exp rp semi\n"); }
| ID ASSIGN exp SEMI
    { printf("Statement -> id assign Exp semi\n"); }
| ID LSP exp RSP ASSIGN exp SEMI
    { printf("Statement -> id lsp Exp rsp assign Exp semi\n"); }
| vdcl
    { printf("Statement -> VarDecl\n"); }
;
//基本單元
exp :
    exp ADD exp
        { printf("Exp -> Exp add Exp\n"); }
| exp MINUS exp
    { printf("Exp -> Exp minus Exp\n"); }
| exp TIMES exp
    { printf("Exp -> Exp times Exp\n"); }
| exp AND exp
    { printf("Exp -> Exp and Exp\n"); }
| exp OR exp
    { printf("Exp -> Exp or Exp\n"); }
| exp LT exp // <
    { printf("Exp -> Exp lt Exp\n"); }
| exp LE exp // >
    { printf("Exp -> Exp le Exp\n"); }
| exp EQ exp
    { printf("Exp -> Exp eq Exp\n"); }
| ID LSP exp RSP // LSP [ RSP ]
    { printf("Exp -> id lsp Exp rsp\n"); }
| ID LP explist RP // LP ( RP )
    { printf("Exp -> id lp ExpList rp\n"); }
| LP exp RP
    { printf("Exp -> lp Exp rp\n"); }
| exp DOT exp
    { printf("Exp -> Exp dot Exp\n"); }
| LIT // 老師的是num(tl不一樣)
    { printf("Exp -> lit\n"); }
| TRUE
    { printf("Exp -> true\n"); }
| FALSE
    { printf("Exp -> false\n"); }
| ID
    { printf("Exp -> id\n"); }
| THIS
    { printf("Exp -> this\n"); }
| NEW INT LSP exp RSP
    { printf("Exp -> new int lsp Exp rsp\n"); }
| NEW ID LP RP
    { printf("Exp -> new id lp rp\n"); }
| NOT exp
    { printf("Exp -> not Exp\n"); }
;

explist :
    exp expr
        { printf("ExpList -> Exp ExpRest*\n"); }
|
    { printf("ExpList -> \n"); }
;

expr :
    COMMA exp expr // COMMA ,

```



```

        { printf("ExpRest -> comma Exp ExpRest\n"); }
    |
    { printf("ExpRest -> \n"); }
;

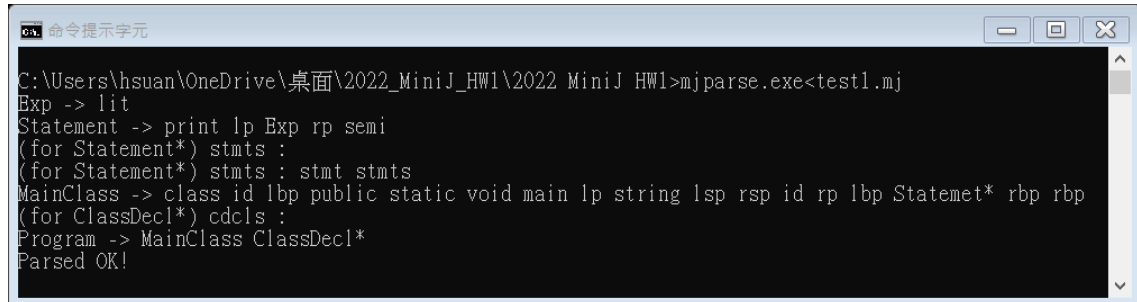
// Practice on writing the grammar rules for
// 1. type
// 2. statement
// 3. exp
// 4. explist 5. exprest
// (see the description in Programming Assignment #1)

%%

// Additional C code
int yyerror(char *s)
{
    printf("%s\n",s);
    return 1;
}

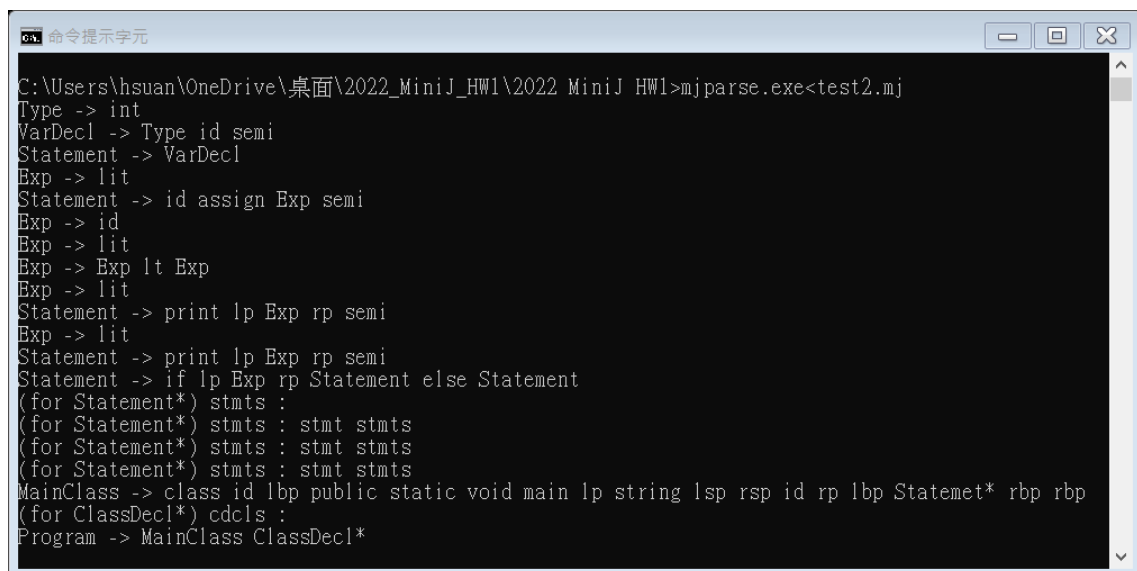
```

四、 Test run results



```
C:\Users\hsuan\OneDrive\桌面\2022_MiniJ_HW1\2022 MiniJ HW1>mjparse.exe<test1.mj
Exp -> lit
Statement -> print lp Exp rp semi
(for Statement*) stmts :
(for Statement*) stmts : stmt stmts
MainClass -> class id lbp public static void main lp string lsp rsp id rp lbp Statemet* rbp rbp
(for ClassDecl*) cdcls :
Program -> MainClass ClassDecl*
Parsed OK!
```

● 圖七、test1.mj 執行結果



```
C:\Users\hsuan\OneDrive\桌面\2022_MiniJ_HW1\2022 MiniJ HW1>mjparse.exe<test2.mj
Type -> int
VarDecl -> Type id semi
Statement -> VarDecl
Exp -> lit
Statement -> id assign Exp semi
Exp -> id
Exp -> lit
Exp -> Exp lt Exp
Exp -> lit
Statement -> print lp Exp rp semi
Exp -> lit
Statement -> print lp Exp rp semi
Statement -> if lp Exp rp Statement else Statement
(for Statement*) stmts :
(for Statement*) stmts : stmt stmts
(for Statement*) stmts : stmt stmts
(for Statement*) stmts : stmt stmts
MainClass -> class id lbp public static void main lp string lsp rsp id rp lbp Statemet* rbp rbp
(for ClassDecl*) cdcls :
Program -> MainClass ClassDecl*
```

● 圖八、test2.mj 執行結果

```
命令提示字元
C:\Users\hsuan\OneDrive\桌面\2022_MiniJ_HW1\2022 MiniJ HW1>mjparse.exe<test3.mj
Exp -> new id lp rp
Exp -> lit
ExpRest ->
ExpRest ->
ExpList -> Exp ExpRest*
Exp -> id lp ExpList rp
Exp -> Exp dot Exp
Statement -> print lp Exp rp semi
(for Statement*) stmts :
(for Statement*) stmts : stmt stmts
MainClass -> class id lbp public static void main lp string lsp rsp id rp lbp Statemet* rbp rbp
(for VarDecl*) vdcls :
Type -> int
Type -> int
FormalRest ->
Formallist -> Type id FormalRest*
Type -> int
VarDecl -> Type id semi
(for VarDecl*) vdcls :
(for VarDecl*) vdcls : vdcl vdcls
Exp -> id
Exp -> lit
Exp -> Exp lt Exp
Exp -> lit
Statement -> id assign Exp semi
Exp -> id
Exp -> this
Exp -> id
```

```
Exp -> lit
Exp -> Exp minus Exp
ExpRest ->
ExpRest ->
ExpList -> Exp ExpRest*
Exp -> id lp ExpList rp
Exp -> Exp dot Exp
Exp -> lp Exp rp
Exp -> Exp times Exp
Statement -> id assign Exp semi
Statement -> if lp Exp rp Statement else Statement
(for Statement*) stmts :
(for Statement*) stmts : stmt stmts
Exp -> id
MethodDecl -> public Type id lp Formallist rp lbp Statements* return Exp semi rbp
(for MethodDecl*) mdcls :
(for MethodDecl*) mdcls : mdcl mdcls
ClassDecl -> class id lbp VarDecl* MethodDecl* rbp
(for ClassDecl*) cdcls :
(for ClassDecl*) cdcls : cdcl cdcls
Program -> MainClass ClassDecl*
Parsed OK!
```

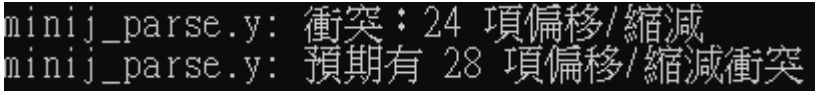
● 圖九、test3.mj 執行結果

五、 Discussion

這部分我們的參考資料有：

1. Bison Declaration Summary([Bison - Bison Grammar Files \(mit.edu\)](https://mit.edu/~bison/))
2. postgresql 核心語法解析器詳解的除錯衝突部分
(<https://codertw.com/%E7%A8%B%E5%BC%3F%E8%AA%9E%E8%A8%80/617577/>)

在將.y 檔 debug 完之後，發現在編譯時出現了一個問題：



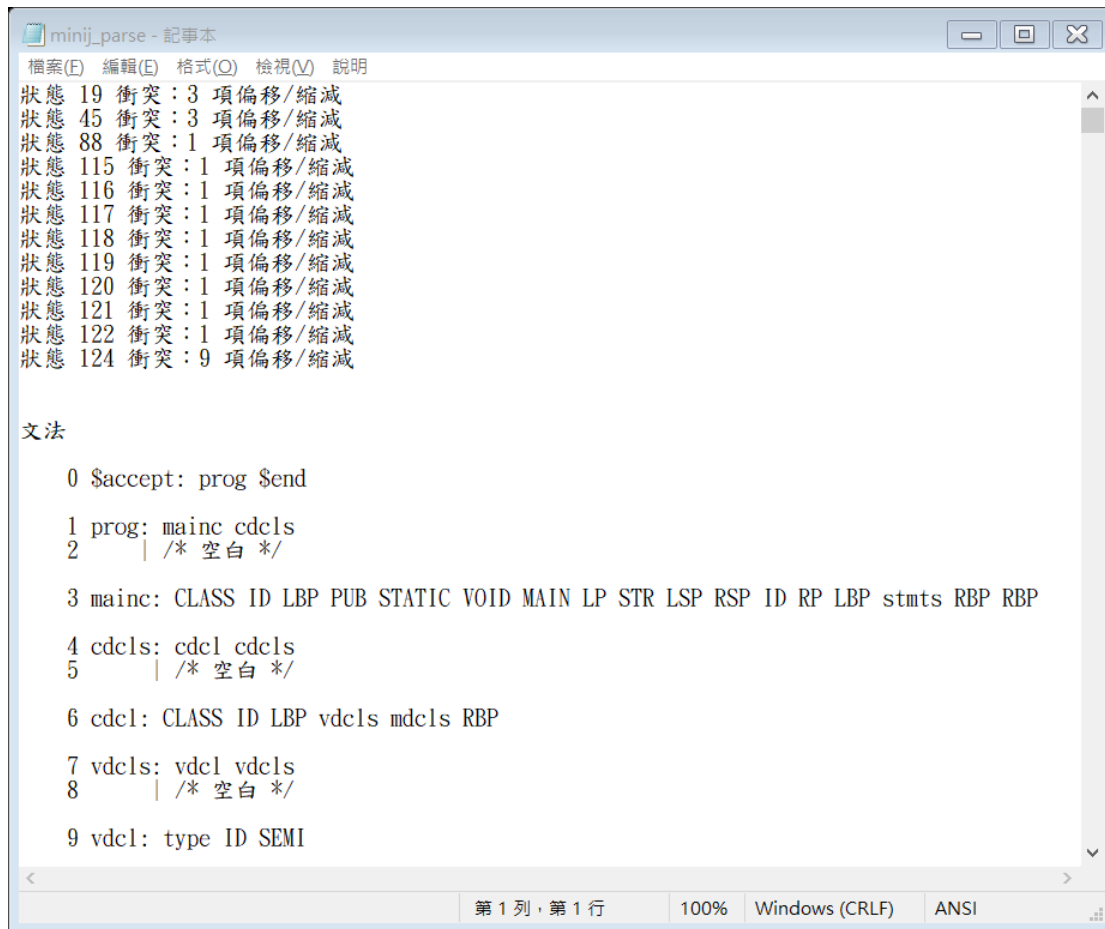
```
mini_parse.y: 衝突: 24 項偏移/縮減  
mini_parse.y: 預期有 28 項偏移/縮減衝突
```

● 圖十、bison 編譯問題

在詢問老師要怎麼解決這個問題後得知可將%expect 28 改為%expect 24，就可以讓 bison 完成編譯。

關於%expect 的用途，在參考資料 1 中的說明為「Declare the expected number of shift-reduce conflicts」，意即此指令是告訴 Bison 解析器應有 N 個 shift/reduce 衝突，如果不匹配，Bison 將報告編譯時錯誤(如圖十)。

在參考資料 2 中說明如果想要除錯移進/規約衝突時，可在 CMD 叫出 bison 時打上-v 可生成一個 OUTPUT 檔，他會將語法解析過程中的某個具體節點的推導路徑給打印出來(如圖十一)，此外，在遇到其他問題例如語法錯誤需要 debug 時，透過 output 檔就能夠很容易地定位到自己規則哪一部分出現異常(如圖十二)。

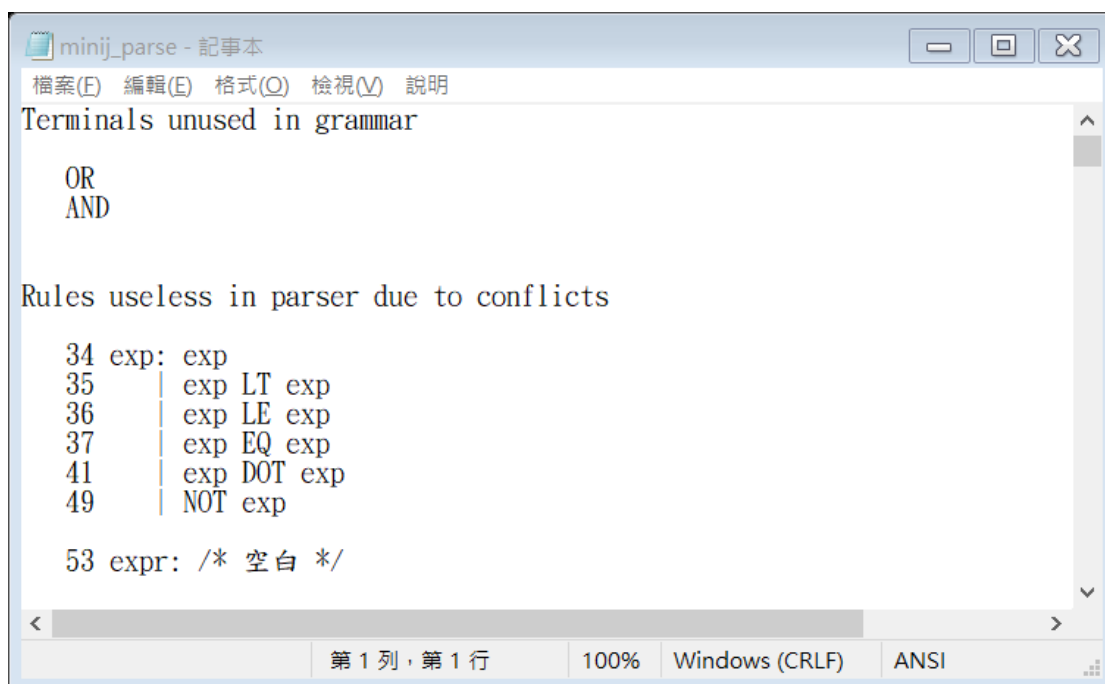


```
minij_parse - 記事本
檔案(E) 編輯(E) 格式(O) 檢視(V) 說明
狀態 19 衝突: 3 項偏移/縮減
狀態 45 衝突: 3 項偏移/縮減
狀態 88 衝突: 1 項偏移/縮減
狀態 115 衝突: 1 項偏移/縮減
狀態 116 衝突: 1 項偏移/縮減
狀態 117 衝突: 1 項偏移/縮減
狀態 118 衝突: 1 項偏移/縮減
狀態 119 衝突: 1 項偏移/縮減
狀態 120 衝突: 1 項偏移/縮減
狀態 121 衝突: 1 項偏移/縮減
狀態 122 衝突: 1 項偏移/縮減
狀態 124 衝突: 9 項偏移/縮減

文法

0 $accept: prog $end
1 prog: mainc cdcls
2   | /* 空白 */
3 mainc: CLASS ID LBP PUB STATIC VOID MAIN LP STR LSP RSP ID RP LBP stmts RBP RBP
4 cdcls: cdcl cdcls
5   | /* 空白 */
6 cdcl: CLASS ID LBP vdcls mdcls RBP
7 vdcls: vdcl vdcls
8   | /* 空白 */
9 vdcl: type ID SEMI
```

● 圖十一、minij_parse.out 截圖



```
minij_parse - 記事本
檔案(E) 編輯(E) 格式(O) 檢視(V) 說明
Terminals unused in grammar

OR
AND

Rules useless in parser due to conflicts

34 exp: exp
35   | exp LT exp
36   | exp LE exp
37   | exp EQ exp
41   | exp DOT exp
49   | NOT exp

53 expr: /* 空白 */
```

● 圖十二、output 檔報告之錯誤部分