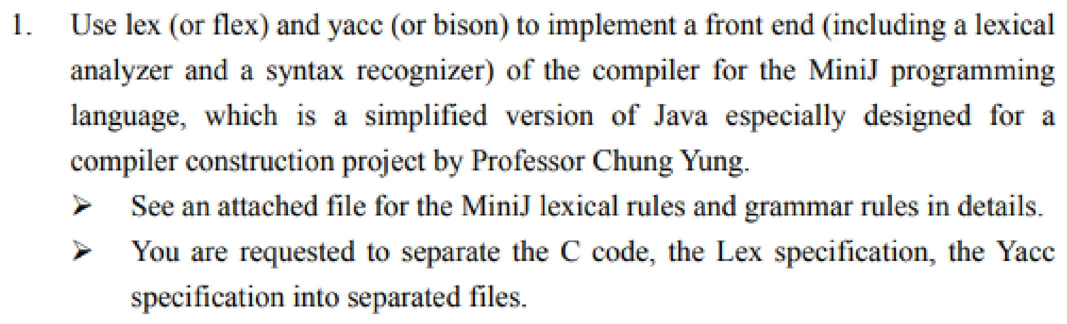
程式語言與編譯器

組員: 410821204 杜昉紜
410821203 朱婉云

題目



1. Use lex (or flex) and yacc (or bison) to implement a front end (including a lexical analyzer and a syntax recognizer) of the compiler for the MiniJ programming language, which is a simplified version of Java especially designed for a compiler construction project by Professor Chung Yung.
   - ➢ See an attached file for the MiniJ lexical rules and grammar rules in details.
   - ➢ You are requested to separate the C code, the Lex specification, the Yacc specification into separated files.

問題概述:
使用 lex（或 flex）和 yacc（或 bison）實現 MiniJ 編程語言編譯器的前端（包括詞法分析器和語法識別器），並要求將 C 代碼、Lex 規範、Yacc 規範分離成單獨的文件

.l 檔 主要修改

我們寫了老師要求的部分，然後發現.l 檔重點地方是 Print 和 comment 部分:

1. System.Out.println 因為 . 有被宣告過，所以要寫成 System"."Out"."println
2. Comment 因為上面有 NONNL [^\n]表示讀入直到換行為止，所以直接寫成 "//"[^\n]*

◎Programlisting (灰色我們寫的，黃色要注意的)

我們寫的 .l 全部:

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "minij.h"
#include "minij_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;}
{ID}                            {return ID;}
{LIT}                           {return LIT;}
"//"[^\n]*                      {return COMMENT;}
[ \t\n]         {/* skip BLANK */}
.               {/* skip redundant characters */}
%%

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

.y 檔 主要修改

主要在 cdcl 中增加了 token – comment，為了把”//”後面的文字忽略不記
新增了 boolean，主要回傳 true / false

◎Programlisting (灰色我們寫的，黃色要注意的)

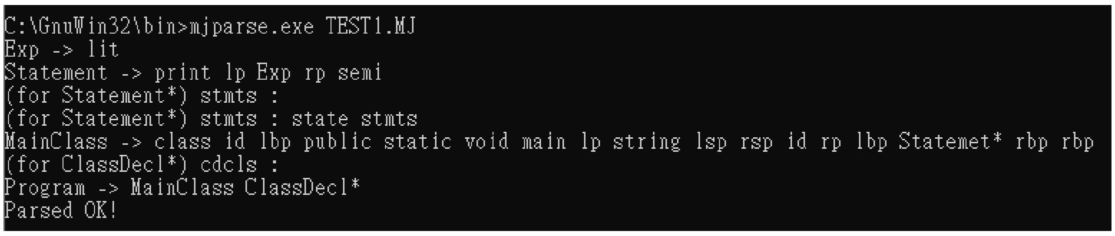
我們寫的 .y 全部:

```
%{
	#include <stdio.h>
	#include <stdlib.h>
	#include <string.h>
	#include "minij.h"
	#include "minij_parse.h"
%}

%token CLASS PUB STATIC
%left	AND OR
%left	LT LE EQ
%left	ADD MINUS
%left	TIMES
%token LBP RBP LSP RSP LP RP
%token INT
%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
%token COMMENT
%%
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"); }
        ;

cdcls:      cdcl cdcls
            { printf("(for ClassDecl*) cdcls : cdcl cdcls\n"); }
        |
            { printf("(for ClassDecl*) cdcls : \n"); }
        ;

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

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"); }
        ;

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"); }
        ;

boolean       :    TRUE
                { printf("boolean -> true\n"); }
        |
                FALSE
                { printf("boolean -> false\n"); }
        ;

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"); }
        ;

stmts       :      state stmts
                { printf("(for Statement*) stmts : state stmts\n"); }
        |
            { printf("(for Statement*) stmts :\n"); }
        ;

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

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
            { printf("Exp -> id lsp Exp rsp\n"); }
        |
            exp LP explist 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
            { 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"); }
	;

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

exrt  :	COMMA exp exrt
		{ printf("ExpRest -> comma exp\n"); }
	|
		{ printf("ExpRest -> \n"); }
	;
%%

int yyerror(char *s)
{
	printf("%s\n",s);
		return 1;
}
```

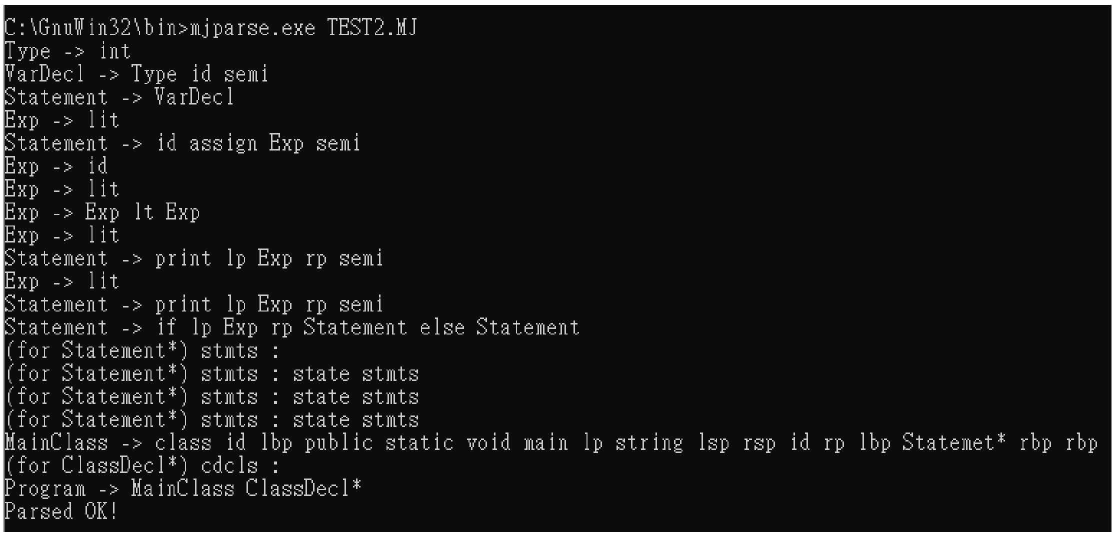
Test run results.

test1 執行結果



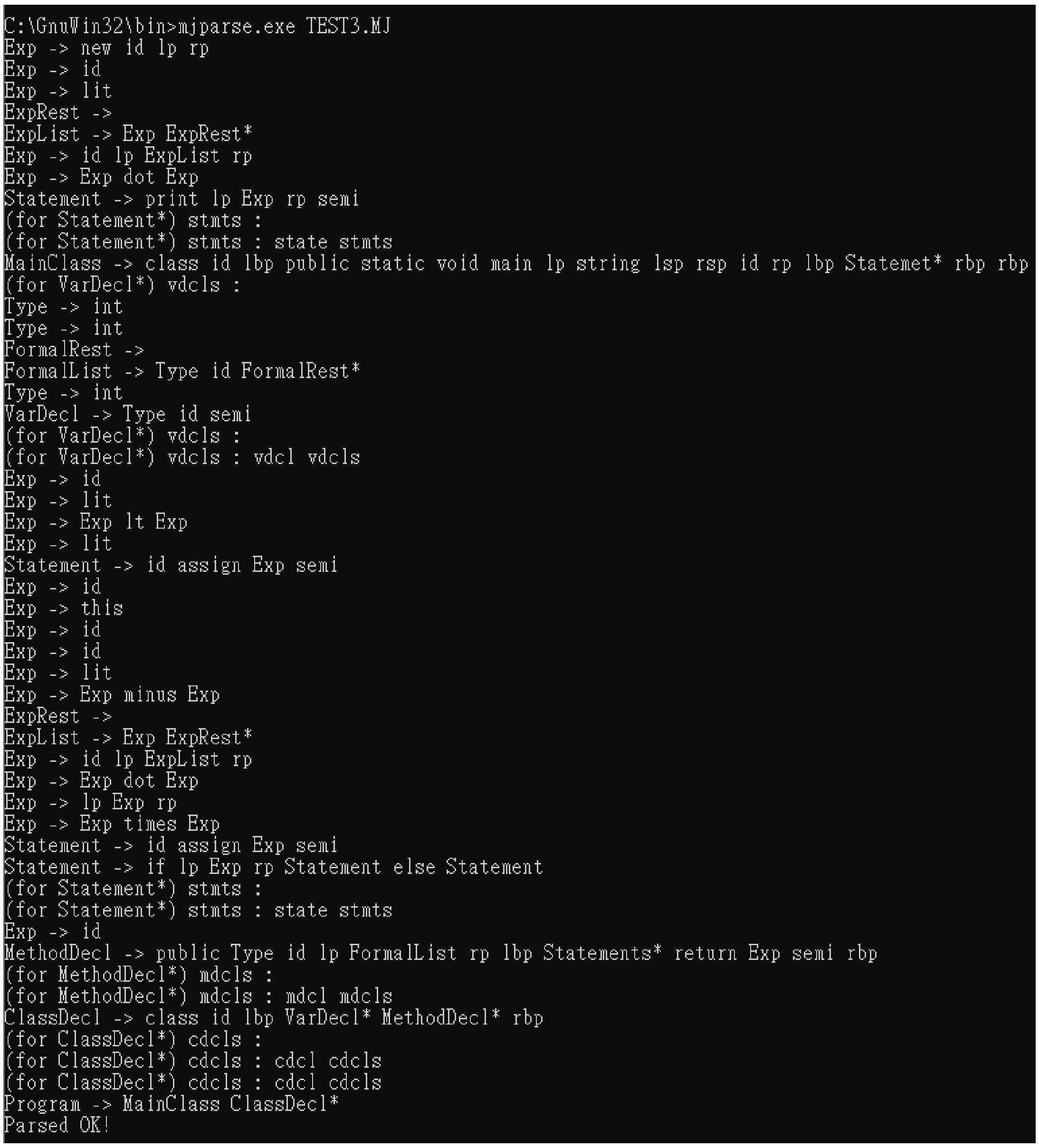
```
C:\GnuWin32\bin>mjparse.exe TEST1.MJ
Exp -> lit
Statement -> print lp Exp rp semi
(for Statement*) stmts :
(for Statement*) stmts : state stmts
MainClass -> class id lbp public static void main lp string lsp rsp id rp lbp Statemet* rbp rbp
(for ClassDecl*) cdcls :
Program -> MainClass ClassDecl*
Parsed OK!
```

test2 執行結果



```
C:\GnuWin32\bin>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 : state stmts
(for Statement*) stmts : state stmts
(for Statement*) stmts : state 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!
```

test3 執行結果



```
C:\GnuWin32\bin>mjparse.exe TEST3.MJ
Exp -> new id lp rp
Exp -> id
Exp -> lit
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 : state 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 -> id
Exp -> lit
Exp -> Exp minus Exp
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 : state 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
(for ClassDecl*) cdcls : cdcl cdcls
Program -> MainClass ClassDecl*
Parsed OK!
```

The problem description & Discussion.

1. Bison 的 m4 檔 不能放在有空白的資料夾

   在執行 bison 的時候，一直有出現"找不到 m4"的檔，後來上網找了很久，才發現 bison m4 的 parent folder 檔名不能有空白，所以我們在 C:\ 新增了檔名中沒空白的 folder，並複製了原本下載的全部內容，還加了新的環境變數並刪除舊的，這樣就解決了。

   結論: 把 bison 移到目前工作目錄中檔名沒空白的 folder

2. minij_parse.y: 衝突：34 項偏移/縮減

   發現跑指令 bison -d -o minij_parse.c minij_parse.y 的時候，會出現 shift / reduce 錯誤提醒，但是可以生成 .c 檔，問老師說: 文法有些細節，寫法不同，conflict 個數有點差距是可以被允許的。

   結論: 只要能生成.c 檔基本上就沒問題，conflict 可以被允許。

3. syntax error

   在跑 test 檔時都有出現 syntax error 的問題，是我們自己新增 test 把每一行 .y 檔 的規則都試過，發現出現錯誤的地方都有 SEMI 這個共通點，最後發現是 .l 檔 裡面宣告 SEMI 的時候，把";"寫成中文打字的"；"，型態不同而出現錯誤，改完就成功了。

   結論: 宣告的符號記的注意有沒有寫到中打的版本