# 程式語言與編譯器
# PROGRAMMING LANGUAGES AND COMPILERS
# PROGRAMMING ASSIGNMENT 1

張芯翎410921304

陳京生410921313

# THE PROBLEM DESCRIPTION

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.

# THE WAY TO WRITE THE PROGRAM

- **For minij_lex.l**

  **Return token.**

  **<COMMENT>\n ->** 遇到換行，回到初始狀態

- **For minij_parse.y**

  **“extern int yylineno” ->** 顯示行數用

  參考老師原本附的，其他上網查

minij_lex.l

flex –o minij_lex.c minij_lex.l



minij_lex.c

minij_parse.y

bison -d -o minij_parse.c minij_parse.y



minij_parse.c

minij_parse.h

minij_lex.c
minij_parse.h
minij.h

gcc -c -o minij_lex.o minij_lex.c

minij_lex.o

minij_parse.c
minij_parse.h
minij.h

gcc -c -o minij_parse.o minij_parse.c

minij_parse.o


minij.c
minij.h
minij_parse.h
gcc -c -o minij.o minij.c
minij.o


minij.o
minij_lex.o
minij_parse.o
gcc -o mjparse minij.o minij_parse.o minij_lex.o
mjparse.exe

# THE PROGRAM LISTING

## The code of minij_lex.l

```
%{
#define YYSTYPE char *
#include <string.h>
#include <stdlib.h>
#include "minij.h"
#include "minij_parse.h"
%}

ID  [A-Za-z][A-Za-z0-9_]*
LIT [0-9][0-9]*
NONNL [^\n]
%x  COMMENT
%option yylineno

%%

class       {return CLASS;}
public      {return PUB;}
static
String{return STATIC;}
void        {return VOID;}
main        {return MAIN;}
int         {return INT;}
if          {return IF;}
else        {return ELSE;}
while       {return WHILE;}
new         {return NEW;}
return
this  {return RETURN;}
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;}
"//"        {BEGIN COMMENT;}
<COMMENT>\n {BEGIN INITIAL;}
<COMMENT>.  { }
{ID}        {yylval = strdup
(yytext); return ID;}
{LIT}       {yylval = strdup
(yytext); return LIT;}

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

%%

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

# THE PROGRAM LISTING

## The code of minij_parse.y

```
%{
    #include <stdio.h>
    #include <stdlib.h>
    #include "minij.h"
    #include "minij_parse.h"
    extern int yylineno;
%}

%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 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 NOT

%expect 32
```

```
%%
prog    :   mainc cdcls
        { printf("%d Program -> MainClass
 ClassDecl*\n", yylineno);
          printf("%d Parsed OK!\n", yylinen
o); }
    |
        { printf("%d ****** Parsing failed!
\n", yylineno); }
    ;

mainc   :   CLASS ID LBP PUB STATIC VOID MA
IN LP STR LSP RSP ID RP LBP stmts RBP RBP
        { printf("%d MainClass -> class id
 lbp public static void main lp string lsp
 rsp id rp lbp Statemet* rbp rbp\n", yyline
no); }
    ;

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

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

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

vdcl    :   type ID SEMI
        { printf("%d VarDecl -> Type id sem
i\n", yylineno); }
    ;
```

```
mdcls   :   mdcl mdcls
        { printf("%d (for MethodDecl*) mdcl
s : mdcl mdcls\n", yylineno); }
    |
        { printf("%d (for MethodDecl*) mdcl
s : \n", yylineno); }
    ;

mdcl    :   PUB type ID LP formals RP LBP v
dcls stmts RETURN exp SEMI RBP
        { printf("%d MethodDecl -> public T
ype id lp FormalList rp lbp Statements* ret
urn Exp semi rbp\n", yylineno); }
    ;

formals :   type ID frest
        { printf("%d FormalList -> Type id
 FormalRest*\n", yylineno); }
    |
        { printf("%d FormalList -> \n", yyl
ineno); }
    ;

frest   :   COMMA type ID frest
        { printf("%d FormalRest -> comma Ty
pe id FormalRest\n", yylineno); }
    |
        { printf("%d FormalRest -> \n", yyl
ineno); }
    ;

type    :   INT LSP RSP
        { printf("%d Type -> INT LSP RSP
 \n", yylineno); }
    |
            BOOLEAN
        { printf("%d Type -> BOOLEAN \n", y
ylineno); }
    |
            INT
        { printf("%d Type -> INT \n", yylin
eno); }
    |
            ID
        { printf("%d Type -> --%s-- \n", yy
lineno, $1); }
    ;
```

```
stmt    :   LBP stmts RBP
        { printf("%d stmt -> lbp Statement*
rbp \n", yylineno); }
    |
            IF LP exp RP stmt ELSE stmt
        { printf("%d stmt -> if lp Exp rp S
tatement else Statement \n", yylineno); }
    |
            WHILE LP exp RP stmt
        { printf("%d stmt -> while lp Exp r
p Statement \n", yylineno); }
    |
            PRINT LP exp RP SEMI
        { printf("%d stmt -> print lp Exp r
p semi \n", yylineno); }
    |
            ID ASSIGN exp SEMI
        { printf("%d stmt -> id assign Exp
 rp semi \n", yylineno); }
    |
            ID LSP exp RSP ASSIGN exp SEMI
        { printf("%d stmt -> id lsp Exp rsp
assign Exp semi \n", yylineno); }
    |
            vdcl
        { printf("%d stmt -> VarDecl \n", y
ylineno); }
    ;

stmts   :   stmt stmts
        { printf("%d stmts -> Statement*
 \n", yylineno); }
    |
        { printf("%d stmts -> \n", yylinen
o); }
    ;
```

```
exp     :   exp ADD exp
        { printf("%d exp -> Exp add Exp
 \n", yylineno); }
    |
            exp MINUS exp
        { printf("%d exp -> Exp minus Exp
 \n", yylineno); }
    |
            exp TIMES exp
        { printf("%d exp -> Exp times Exp
 \n", yylineno); }
    |
            exp AND exp
        { printf("%d exp -> Exp and Exp
 \n", yylineno); }
    |
            exp OR exp
        { printf("%d exp -> Exp or Exp \n",
yylineno); }
    |
            exp LT exp
        { printf("%d exp -> Exp lt Exp \n",
yylineno); }
    |
            exp LE exp
        { printf("%d exp -> Exp le Exp \n",
yylineno); }
    |
            exp EQ exp
        { printf("%d exp -> Exp eq Exp \n",
yylineno); }
    |
            ID LSP exp RSP
        { printf("%d exp -> id lsp Exp rsp
 \n", yylineno); }
    |
            ID LP expls RP
        { printf("%d exp -> id lp Explist r
p \n", yylineno); }
    |
            LP exp RP
        { printf("%d exp -> lp Exp rp \n",
 yylineno); }
    |
```

```
       exp DOT exp
        { printf("%d exp -> Exp dot Exp
 \n", yylineno); }
    |
           LIT
        { printf("%d exp -> --%s-- \n", yyl
ineno, $1); }
    |
           TRUE
        { printf("%d exp -> true \n", yylin
eno); }
    |
           FALSE
        { printf("%d exp -> false \n", yyli
neno); }
    |
           ID
        { printf("%d exp -> --%s-- \n", yyl
ineno, $1); }
    |
           THIS
        { printf("%d exp -> this \n", yylin
eno); }
    |
           NEW INT LSP exp RSP
        { printf("%d exp -> new int lsp exp
rsp \n", yylineno); }
    |
           NEW ID LP RP
        { printf("%d exp -> new id lp rp
 \n", yylineno); }
    |
           NOT exp
        { printf("%d exp -> Exp \n", yyline
no); }
    ;

expls   :   exp exprest
        { printf("%d expls -> Exp ExpRest*
 \n", yylineno); }
    |
        { printf("%d expls -> \n", yylinen
o); }
    ;
```

```
exprest : COMMA exp exprest
        { printf("%d exprests -> comma Exp
 \n", yylineno); }
    |
        { printf("%d exprests -> \n", yylin
eno); }
    ;

%%

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

# TEST RUN RESULTS

test1.mj

```
3 exp -> --10--
3 stmt -> print lp Exp rp semi
4 stmts ->
4 stmts -> Statement*
5 MainClass -> class id lbp public static void main lp string lsp rsp id rp lbp Statemet* rbp rbp
7 (for ClassDecl*) cdcls :
7 Program -> MainClass ClassDecl*
7 Parsed OK!
```

# test2.mj

```
3 Type -> INT
3 VarDecl -> Type id semi
3 stmt -> VarDecl
4 exp -> --10--
4 stmt -> id assign Exp rp semi
5 exp -> --i--
5 exp -> --1--
5 exp -> Exp lt Exp
6 exp -> --0--
6 stmt -> print lp Exp rp semi
8 exp -> --1--
8 stmt -> print lp Exp rp semi
8 stmt -> if lp Exp rp Statement else Statement
9 stmts ->
9 stmts -> Statement*
9 stmts -> Statement*
9 stmts -> Statement*
10 MainClass -> class id lbp public static void main lp string lsp rsp id rp lbp Statemet* rbp rbp
12 (for ClassDecl*) cdcls :
12 Program -> MainClass ClassDecl*
12 Parsed OK!
```

# test3.mj

```
3 exp -> new id lp rp
3 exp -> --10--
3 exprests ->
3 expls -> Exp ExpRest*
3 exp -> id lp Explist rp
3 exp -> Exp dot Exp
3 stmt -> print lp Exp rp semi
4 stmts ->
4 stmts -> Statement*
5 MainClass -> class id lbp public static void main lp string lsp rsp id rp lbp Statemet* rbp rbp
```

```
9 (for VarDecl*) vdcls :
9 Type -> INT
9 Type -> INT
9 FormalRest ->
9 FormalList -> Type id FormalRest*
10 Type -> INT
10 VarDecl -> Type id semi
11 (for VarDecl*) vdcls :
11 (for VarDecl*) vdcls : vdcl vdcls
11 exp -> --num--
11 exp -> --l--
11 exp -> Exp lt Exp
12 exp -> --l--
12 stmt -> id assign Exp rp semi
14 exp -> --num--
14 exp -> this
14 exp -> --num--
14 exp -> --l--
14 exp -> Exp minus Exp
14 exprests ->
14 expls -> Exp ExpRest*
14 exp -> id lp Explist rp
14 exp -> Exp dot Exp
14 exp -> lp Exp rp
14 exp -> Exp times Exp
14 stmt -> id assign Exp rp semi
14 stmt -> if lp Exp rp Statement else Statement
15 stmts ->
15 stmts -> Statement*
15 exp -> --num_aux--
16 MethodDecl -> public Type id lp FormalList rp lbp Statements* return Exp semi rbp
17 (for MethodDecl*) mdcls :
17 (for MethodDecl*) mdcls : mdcl mdcls
17 ClassDecl -> class id lbp VarDecl* MethodDecl* rbp
18 (for ClassDecl*) cdcls :
18 (for ClassDecl*) cdcls : cdcl cdcls
18 Program -> MainClass ClassDecl*
18 Parsed OK!
```

# DISCUSSION

很難debug，不知道bug從哪裡來，網路上能找到的資源有點少，和其他人討論很重要，例如會遇到諸如此類的問題☹ : (這時候就需要有人幫忙debug)

```
1 Type -> --(null)--
syntax error
```

```
1 Type -> INT
VarDecl -> Type id semi
1 stmt -> VarDecl
1 exp -> --(null)--
1 stmt -> id assign Exp rp semi
1 exp -> --(null)--
1 exp -> --(null)--
1 exp -> Exp lt Exp
1 Type -> --(null)--
syntax error
```