J Compiler

Using flex and bison

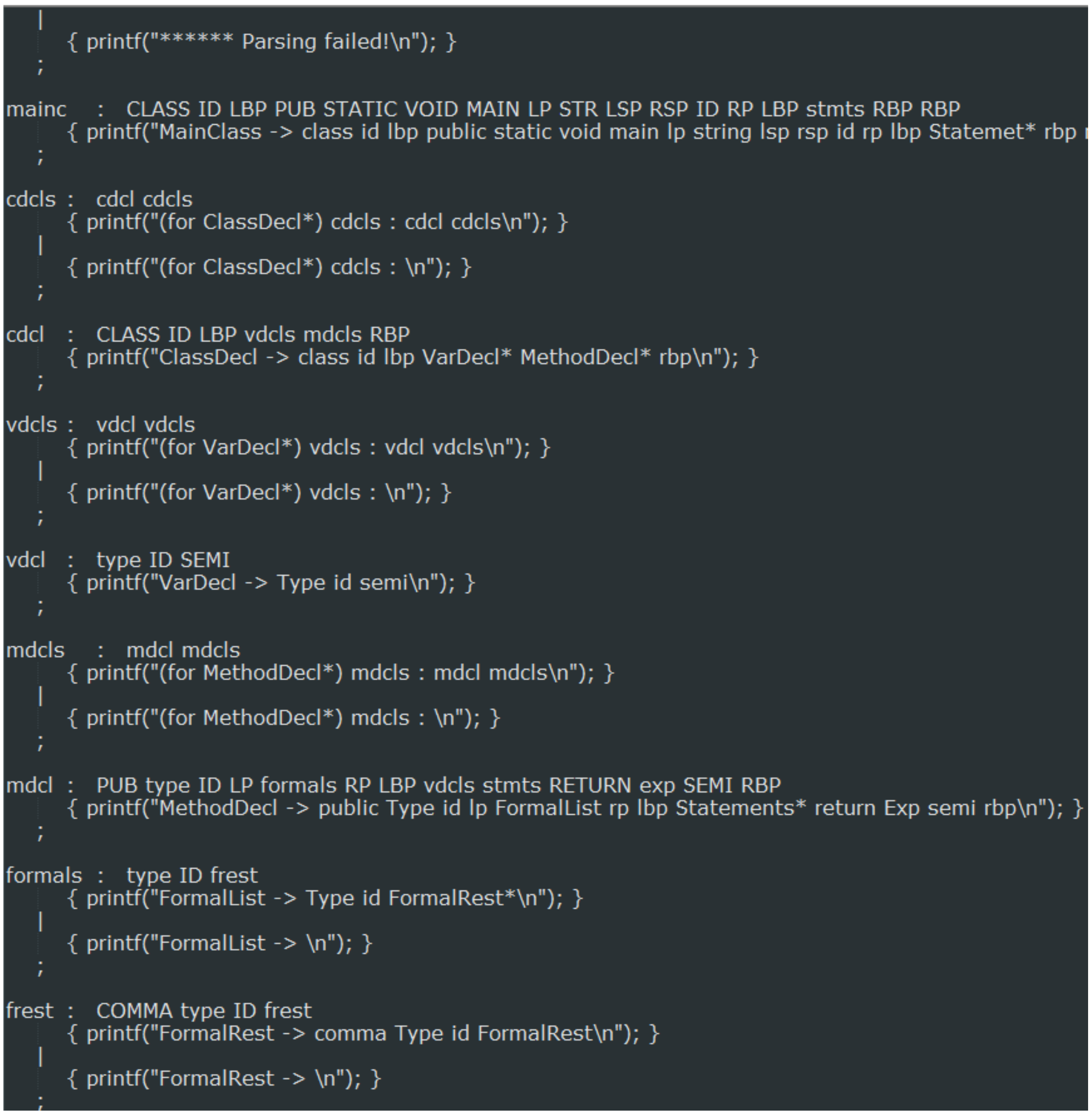
CSIE, NDHU 2018

高英皓

The problem description.

Use flex and bison to implement a front end (including a lexical analyzer and a syntax recognizer) of the compiler for the j programming language, which is a simplified version of Java and specially designed as a compiler construction project by Professor Chung Yung.

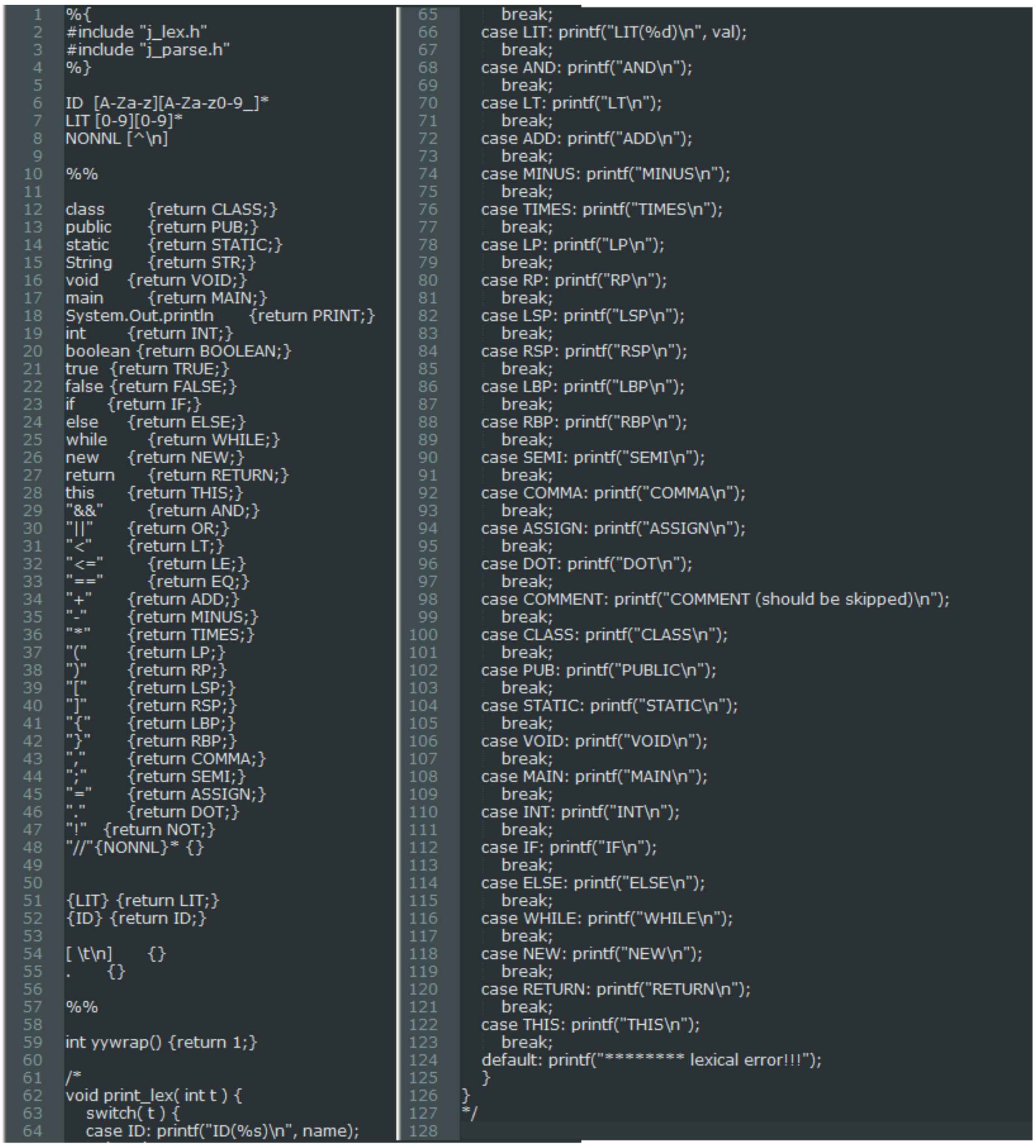
Program Highlight



```
    |
        { 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
    ;

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"); }
    ;

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"); }
    ;
```

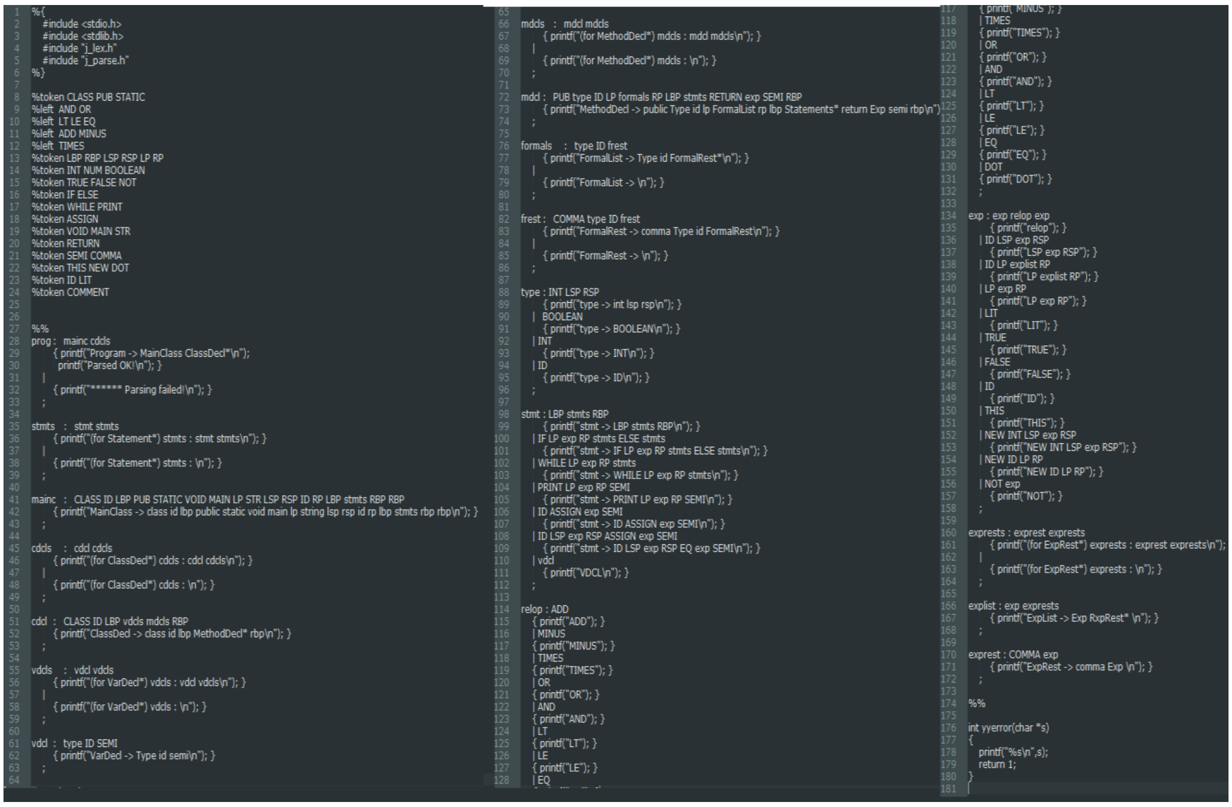
Program Listing:

J_lex.l



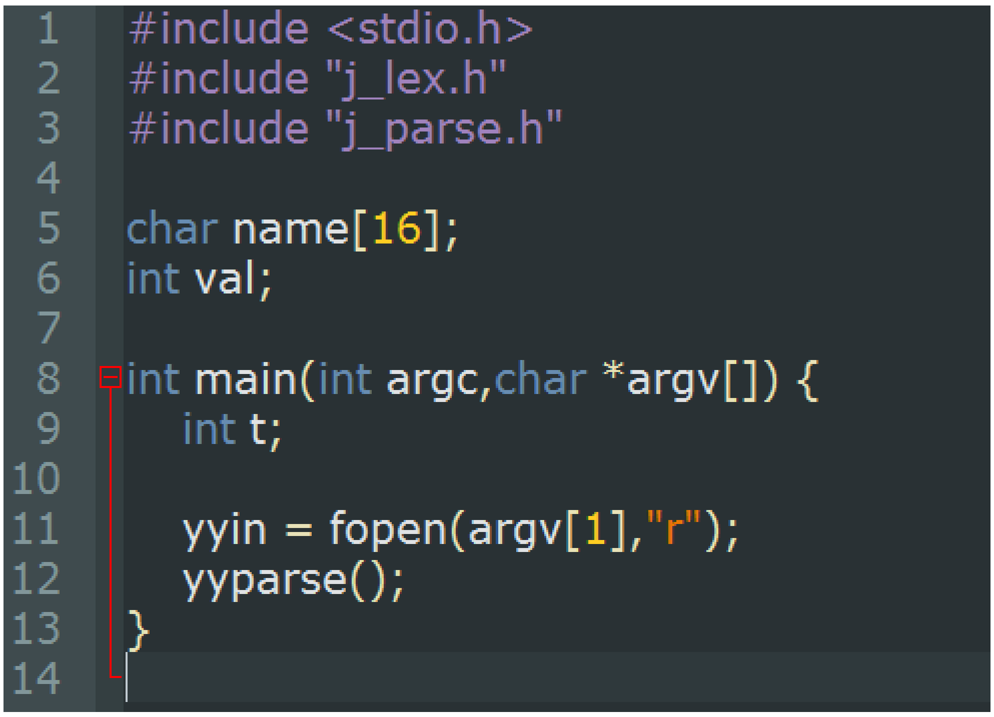
```
%{
#include "j_lex.h"
#include "j_parse.h"
%}

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;}
System.Out.println      {return PRINT;}
int     {return INT;}
boolean {return BOOLEAN;}
true  {return TRUE;}
false {return FALSE;}
if      {return IF;}
else    {return ELSE;}
while       {return WHILE;}
new     {return NEW;}
return      {return RETURN;}
this    {return THIS;}
"&&"        {return AND;}
"||"    {return OR;}
"<"     {return LT;}
"<="        {return LE;}
"=="        {return EQ;}
"+"     {return ADD;}
"-"     {return MINUS;}
"*"     {return TIMES;}
"("     {return LP;}
")"     {return RP;}
"["     {return LSP;}
"]"     {return RSP;}
"{"     {return LBP;}
"}"     {return RBP;}
","     {return COMMA;}
";"     {return SEMI;}
"="     {return ASSIGN;}
"."     {return DOT;}
"!"   {return NOT;}
"//"{NONNL}* {}


{LIT} {return LIT;}
{ID} {return ID;}

[ \t\n]     {}
.     {}

%%

int yywrap() {return 1;}

/*
void print_lex( int t ) {
   switch( t ) {
   case ID: printf("ID(%s)\n", name);
      break;
   case LIT: printf("LIT(%d)\n", val);
      break;
   case AND: printf("AND\n");
      break;
   case LT: printf("LT\n");
      break;
   case ADD: printf("ADD\n");
      break;
   case MINUS: printf("MINUS\n");
      break;
   case TIMES: printf("TIMES\n");
      break;
   case LP: printf("LP\n");
      break;
   case RP: printf("RP\n");
      break;
   case LSP: printf("LSP\n");
      break;
   case RSP: printf("RSP\n");
      break;
   case LBP: printf("LBP\n");
      break;
   case RBP: printf("RBP\n");
      break;
   case SEMI: printf("SEMI\n");
      break;
   case COMMA: printf("COMMA\n");
      break;
   case ASSIGN: printf("ASSIGN\n");
      break;
   case DOT: printf("DOT\n");
      break;
   case COMMENT: printf("COMMENT (should be skipped)\n");
      break;
   case CLASS: printf("CLASS\n");
      break;
   case PUB: printf("PUBLIC\n");
      break;
   case STATIC: printf("STATIC\n");
      break;
   case VOID: printf("VOID\n");
      break;
   case MAIN: printf("MAIN\n");
      break;
   case INT: printf("INT\n");
      break;
   case IF: printf("IF\n");
      break;
   case ELSE: printf("ELSE\n");
      break;
   case WHILE: printf("WHILE\n");
      break;
   case NEW: printf("NEW\n");
      break;
   case RETURN: printf("RETURN\n");
      break;
   case THIS: printf("THIS\n");
      break;
   default: printf("******** lexical error!!!");
   }
}
*/
```

J_parse.y



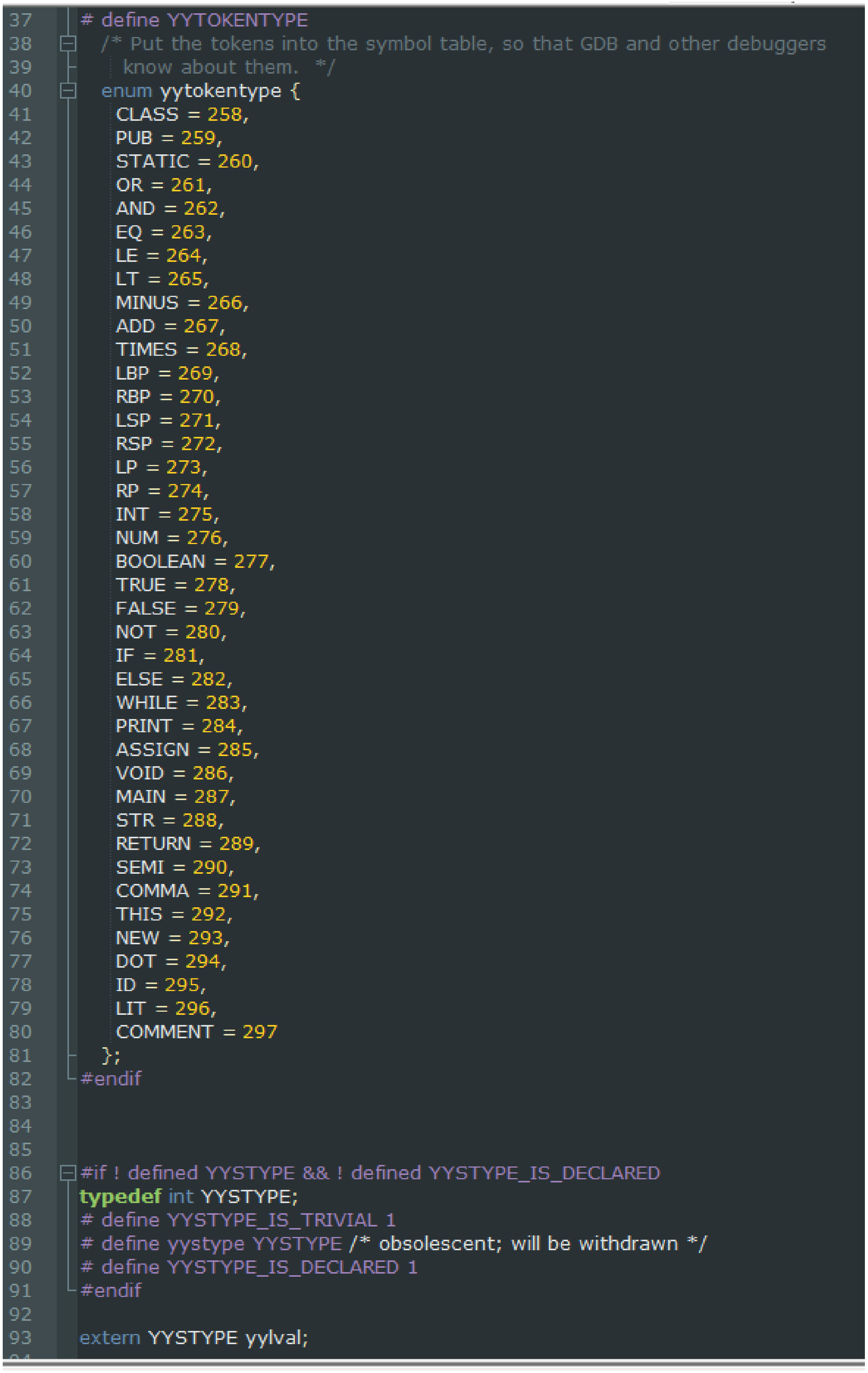
```
%{
   #include <stdio.h>
   #include <stdlib.h>
   #include "j_lex.h"
   #include "j_parse.h"
%}

%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 NUM BOOLEAN
%token TRUE FALSE NOT
%token IF ELSE
%token WHILE PRINT
%token ASSIGN
%token VOID MAIN STR
%token RETURN
%token SEMI COMMA
%token THIS NEW DOT
%token ID LIT
%token COMMENT


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

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

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 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"); }
   ;

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"); }
   ;

stmt : LBP stmts RBP
      { printf("stmt -> LBP stmts RBP\n"); }
   | IF LP exp RP stmts ELSE stmts
      { printf("stmt -> IF LP exp RP stmts ELSE stmts\n"); }
   | WHILE LP exp RP stmts
      { printf("stmt -> WHILE LP exp RP stmts\n"); }
   | PRINT LP exp RP SEMI
      { printf("stmt -> PRINT LP exp RP SEMI\n"); }
   | ID ASSIGN exp SEMI
      { printf("stmt -> ID ASSIGN exp SEMI\n"); }
   | ID LSP exp RSP ASSIGN exp SEMI
      { printf("stmt -> ID LSP exp RSP EQ exp SEMI\n"); }
   | vdcl
      { printf("VDCL\n"); }
   ;

relop : ADD
   { printf("ADD"); }
   | MINUS
   { printf("MINUS"); }
   | TIMES
   { printf("TIMES"); }
   | OR
   { printf("OR"); }
   | AND
   { printf("AND"); }
   | LT
   { printf("LT"); }
   | LE
   { printf("LE"); }
   | EQ
   { printf("EQ"); }
   | DOT
   { printf("DOT"); }
   ;

exp : exp relop exp
      { printf("relop"); }
   | ID LSP exp RSP
      { printf("LSP exp RSP"); }
   | ID LP explist RP
      { printf("LP explist RP"); }
   | LP exp RP
      { printf("LP exp RP"); }
   | LIT
      { printf("LIT"); }
   | TRUE
      { printf("TRUE"); }
   | FALSE
      { printf("FALSE"); }
   | ID
      { printf("ID"); }
   | THIS
      { printf("THIS"); }
   | NEW INT LSP exp RSP
      { printf("NEW INT LSP exp RSP"); }
   | NEW ID LP RP
      { printf("NEW ID LP RP"); }
   | NOT exp
      { printf("NOT"); }
   ;

exprests : exprest exprests
      { printf("(for ExpRest*) exprests : exprest exprests\n");
   |
      { printf("(for ExpRest*) exprests : \n"); }
   ;

explist : exp exprests
      { printf("ExpList -> Exp RxpRest* \n"); }
   ;

exprest : COMMA exp
      { printf("ExpRest -> comma Exp \n"); }
   ;

%%

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

Main.c



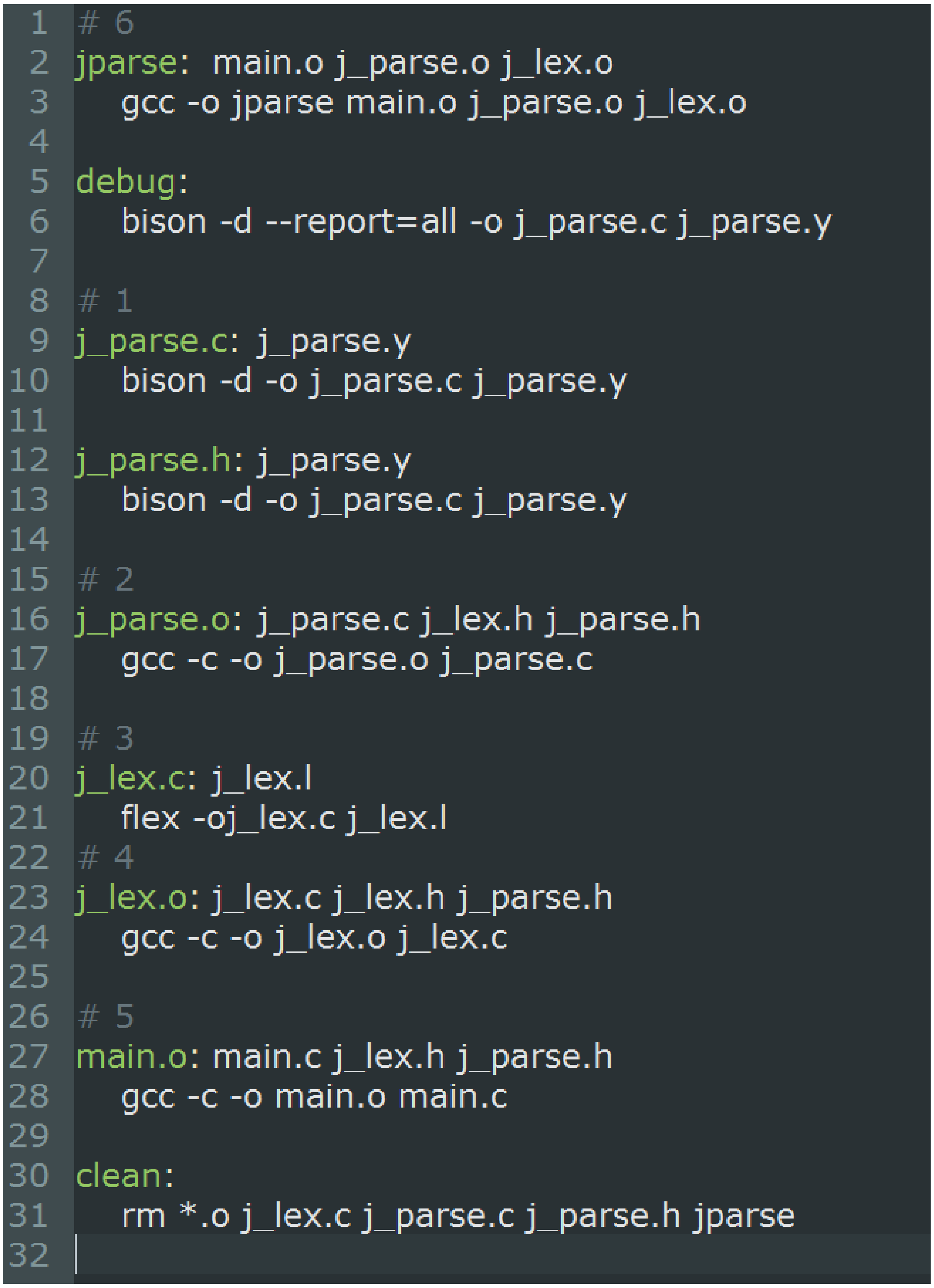
```
#include <stdio.h>
#include "j_lex.h"
#include "j_parse.h"

char name[16];
int val;

int main(int argc,char *argv[]) {
    int t;

    yyin = fopen(argv[1],"r");
    yyparse();
}
```

J_parse.h



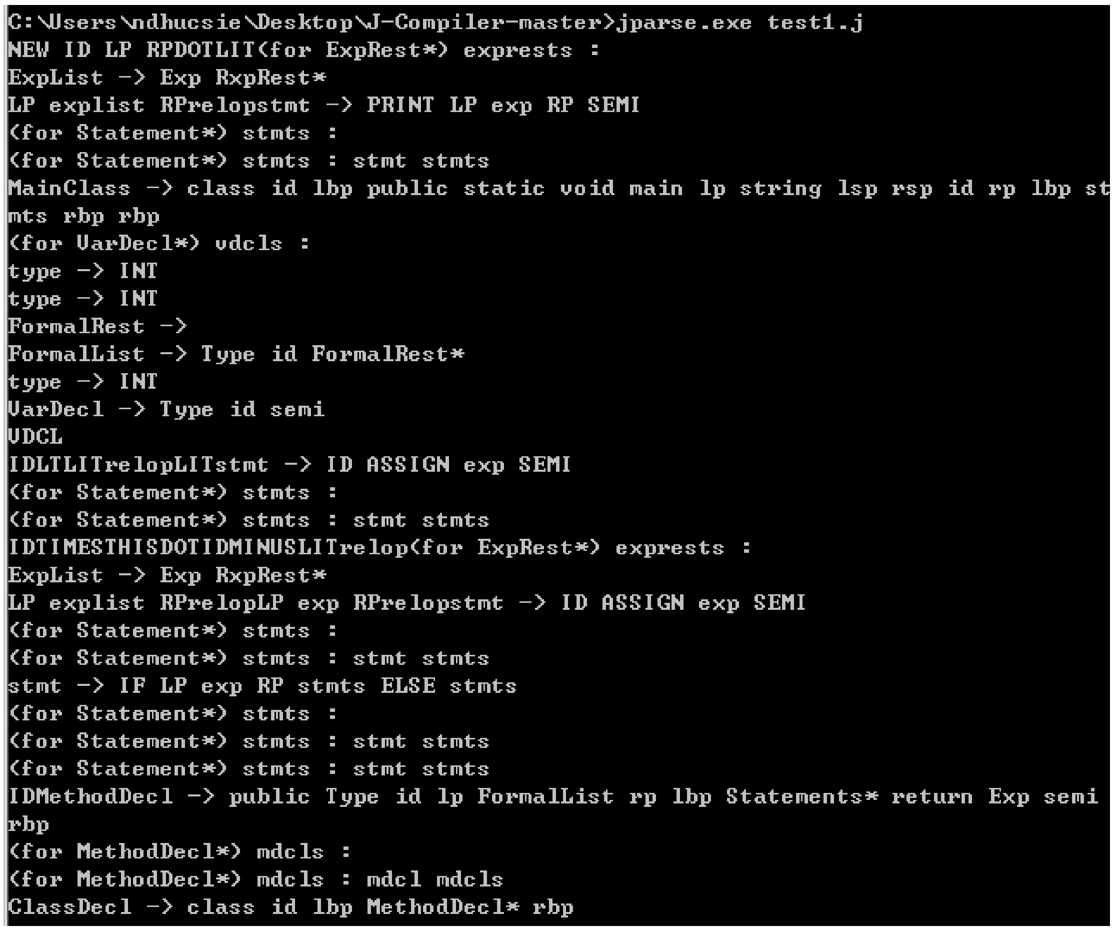
```
# define YYTOKENTYPE
  /* Put the tokens into the symbol table, so that GDB and other debuggers
     know about them.  */
  enum yytokentype {
    CLASS = 258,
    PUB = 259,
    STATIC = 260,
    OR = 261,
    AND = 262,
    EQ = 263,
    LE = 264,
    LT = 265,
    MINUS = 266,
    ADD = 267,
    TIMES = 268,
    LBP = 269,
    RBP = 270,
    LSP = 271,
    RSP = 272,
    LP = 273,
    RP = 274,
    INT = 275,
    NUM = 276,
    BOOLEAN = 277,
    TRUE = 278,
    FALSE = 279,
    NOT = 280,
    IF = 281,
    ELSE = 282,
    WHILE = 283,
    PRINT = 284,
    ASSIGN = 285,
    VOID = 286,
    MAIN = 287,
    STR = 288,
    RETURN = 289,
    SEMI = 290,
    COMMA = 291,
    THIS = 292,
    NEW = 293,
    DOT = 294,
    ID = 295,
    LIT = 296,
    COMMENT = 297
  };
#endif



#if ! defined YYSTYPE && ! defined YYSTYPE_IS_DECLARED
typedef int YYSTYPE;
# define YYSTYPE_IS_TRIVIAL 1
# define yystype YYSTYPE /* obsolescent; will be withdrawn */
# define YYSTYPE_IS_DECLARED 1
#endif

extern YYSTYPE yylval;
```

MAKEFILE

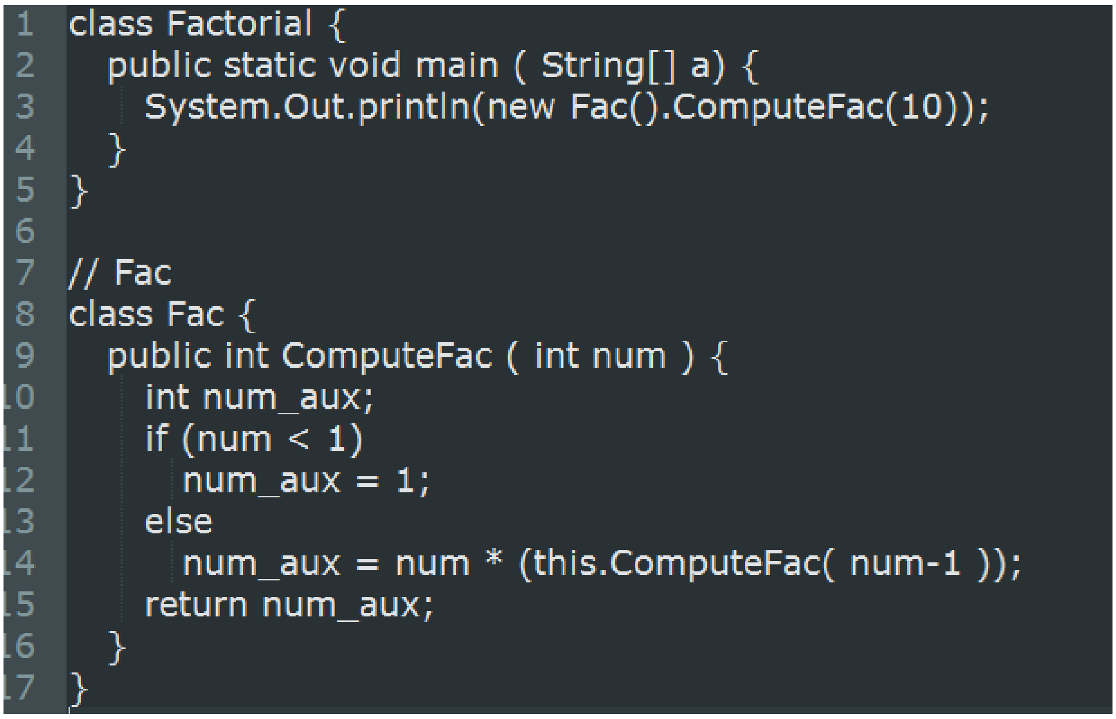


```makefile
# 6
jparse:  main.o j_parse.o j_lex.o
	gcc -o jparse main.o j_parse.o j_lex.o

debug:
	bison -d --report=all -o j_parse.c j_parse.y

# 1
j_parse.c: j_parse.y
	bison -d -o j_parse.c j_parse.y

j_parse.h: j_parse.y
	bison -d -o j_parse.c j_parse.y

# 2
j_parse.o: j_parse.c j_lex.h j_parse.h
	gcc -c -o j_parse.o j_parse.c

# 3
j_lex.c: j_lex.l
	flex -oj_lex.c j_lex.l
# 4
j_lex.o: j_lex.c j_lex.h j_parse.h
	gcc -c -o j_lex.o j_lex.c

# 5
main.o: main.c j_lex.h j_parse.h
	gcc -c -o main.o main.c

clean:
	rm *.o j_lex.c j_parse.c j_parse.h jparse

```

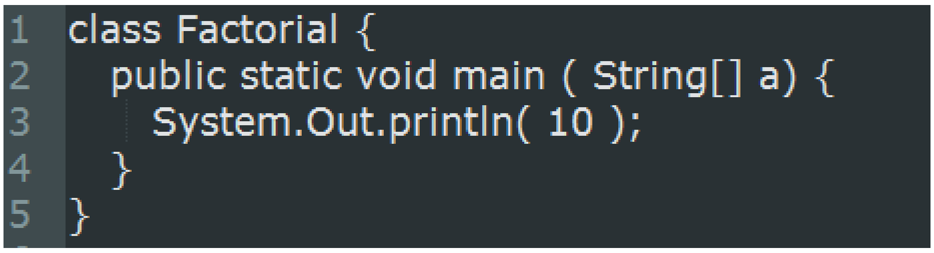
Test and Result:


```
C:\Users\ndhucsie\Desktop\J-Compiler-master>jparse.exe test1.j
NEW ID LP RPDOTLIT(for ExpRest*) exprests :
ExpList -> Exp RxpRest*
LP explist RPrelopstmt -> 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 st
mts rbp rbp
(for VarDecl*) vdcls :
type -> INT
type -> INT
FormalRest ->
FormalList -> Type id FormalRest*
type -> INT
VarDecl -> Type id semi
VDCL
IDLTLITrelopLITstmt -> ID ASSIGN exp SEMI
(for Statement*) stmts :
(for Statement*) stmts : stmt stmts
IDTIMESTHISDOTIDMINUSLITrelop(for ExpRest*) exprests :
ExpList -> Exp RxpRest*
LP explist RPrelopLP exp RPrelopstmt -> ID ASSIGN exp SEMI
(for Statement*) stmts :
(for Statement*) stmts : stmt stmts
stmt -> IF LP exp RP stmts ELSE stmts
(for Statement*) stmts :
(for Statement*) stmts : stmt stmts
(for Statement*) stmts : stmt stmts
IDMethodDecl -> public Type id lp FormalList rp lbp Statements* return Exp semi
rbp
(for MethodDecl*) mdcls :
(for MethodDecl*) mdcls : mdcl mdcls
ClassDecl -> class id lbp MethodDecl* rbp
```

test1.j


```
class Factorial {
   public static void main ( String[] a) {
      System.Out.println(new Fac().ComputeFac(10));
   }
}

// Fac
class Fac {
   public int ComputeFac ( int num ) {
      int num_aux;
      if (num < 1)
         num_aux = 1;
      else
         num_aux = num * (this.ComputeFac( num-1 ));
      return num_aux;
   }
}
```

test2.j



```
class Factorial {
  public static void main ( String[] a) {
    System.Out.println( 10 );
  }
}
```

Discussion:
在實作這次的程式之前，都要把所有 ERROR 和 Warning 全部處理，不論是 C/C++ 或 Python 等等。這次 BISON 中 shift/reduce 和 reduce/reduce 的 Warning 可以 expect，真是覺得通體舒暢。這次除了更熟練 BISON 和 FLEX，也更了解正規語言的細節。最後感謝助教上機的時候不斷解決我的疑惑，尤其是系統環境變數的地方，如果不做修改，甚至連 BISON 都沒辦法編譯。