MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent institution of MAHE, Manipal)

Compiler Design Project

Compiler Design Lab – CSE 3211

Mini Java Parser

Group members
Aditya Camarushy (49)
Rishit Ratan (54)
Swetha Sairamakrishnan (58)
Yogesh Lather (64)

Table of Contents

Objective

The objective of the projects is to implement a simplified model of a parser program that parses a subset of Java. While there exist many Java parsers, this project aims at parsing a subset of Java grammar using a flex and bison based compiler to identify regular expressions and generate a token stream along with a symbol table.

Grammar

SOURCE_CODE = IMPORT_STMT CLASS_DCLR | CLASS_DCLR

IMPORT_STMT = import #api_identifier * ;

CLASS_DCLR = class #identifier { CLASS_DEF }

CLASS_DEF = MAIN_METHOD | #e

MAIN_METHOD = ACCESS STATIC TYPE main (PARAM) EXCEPTIONS

{ STMT }

ACCESS = public | private | protected | #e

STATIC = static | #e

PARAM = TYPE PARAM` | #e

PARAM` = #identifier [ ] | [ ] #identifier

TYPE = int | void | String

EXCEPTIONS = throws #api_identifier | #e

STMT = VAR_DCLR ; STMT | CALL_STMT ; STMT | ASSIGN_STMT ; STMT | CTRL_STMT STMT | { STMT }

| #e

VAR_DCLR = TYPE #identifier VAR_INIT MORE_VAR_DCLR

VAR_INIT = #identifier = VAR_INITIAL_VAL | = #e

VAR_INITIAL_VAL = EXPR | NEW

NEW = new TYPE (NEW`

NEW` =) | NEW) | #api_identifier)

MORE_VAR_DCLR = , #identifier VAR_INIT MORE_VAR_DCLR| #e

CALL_STMT = METHOD_NAME (ARGS)

METHOD_NAME = #api_identifier

ARGS = ARG MORE_ARGS

MORE_ARGS = , ARGS | + ARGS | #e

ARG = CALL_STMT | #string | #identifier | #e

```
ASSIGN_STMT = #identifier ASSIGN_STMT`
ASSIGN_STMT`= #identifier = ASSIGN_SRC | ASSIGN_OP EXPR
ASSIGN_SRC = CALL_STMT | EXPR
ASSIGN_OP = += | -= | *= | /= | %=
EXPR = TERM EXPR`
EXPR` = ADD_OP TERM EXPR` | #e
ADD_OP = + | -
TERM = FACTOR TERM`
TERM` = MUL_OP FACTOR TERM` | #e
MUL_OP = * | / | %
FACTOR = ( EXPR ) | #identifier | #number
CTRL_STMT = IF_STMT | WHILE_STMT | FOR_STMT
IF_STMT = if ( TEST ) STMT_SINGLE ELSE_PART
ELSE_PART = else STMT_SINGLE | #e
STMT_SINGLE = VAR_DCLR ; | CALL_STMT ; | ASSIGN_STMT ; | CTRL_STMT ; | { STMT }
TEST = EXPR TEST_OP TEST_NORM
TEST_NORM = #identifier| #number | ( EXPR )
TEST_OP = < | > | >= | <= | ==
WHILE_STMT = while ( TEST ) STMT_SINGLE
FOR_STMT = for ( ASSIGN_STMT ; TEST ; U_EXPR ) STMT_SINGLE
U_EXPR = #identifier ++ | #identifier -- | ++ #identifier | -- #identifier
```

Languages used for implementation

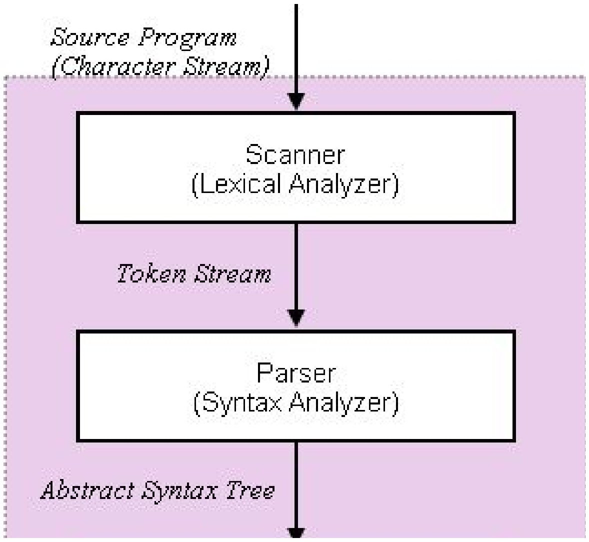
1. C
2. Flex
3. Bison

The type of parser

Bison is a general-purpose *parser generator* that converts a grammar description (Bison Grammar Files) for an LALR(1) context-free grammar into a C program to parse that grammar.

The Bison parser is a **bottom-up** parser. It tries, by shifts and reductions, to reduce the entire input down to a single grouping whose symbol is the grammar's start-symbol.

Methodology



The first phase of a compiler is called **lexical analysis/scanning**. The lexical analyser reads the stream of characters making up the source program and groups the characters into meaningful sequences called lexemes.
It passes on this stream of tokens to the subsequent phase, **syntax analysis**.
In the token, the first component token- name is an abstract symbol that is used during syntax analysis, and the second component attribute-value points to an entry in the symbol table for this token. Information from the symbol-table entry 'is needed for semantic analysis and code generation phases.

User Documentation / Readme text

Prerequisites - Flex and Bison installed prior to running.

Type in the following commands in the linux terminal for within directory of the shell script ‘javacompl.sh’ and type the following -

To Run The Parser as it is :

```
chmod 777 javacompl.sh ./javacompl.sh
```

To recompile from scratch and run after some changes made to the program -

Step I) In directory Parser/

```
bison -d a.y g
cc lex.yy.c a.tab.c -o parser
./parser
```

Note: For the input file ‘first.java’ , change the path to your own in the ‘a.y’ file.

Step II) In directory Symtable/

```
flex tokgen.l
gcc lex.yy.c -o symtable
./symtable
```

Note: For the input file ‘first.java’ , change the path to your own in the ‘tokgen.l’ file.

Code

1) Code for the parser

a)Lex file ‘ first.l ’

```
%{
#include <stdio.h>
#include <stdlib.h>
#include "a.tab.h"
#include "variable.h"
lineno = 0;
%}
%%
\n {lineno++;}
"--"(.)*\n {printf("This is a single line comment and will be
ignored: %s\n", yytext); }
"--[["(.|[\n])*"]]" {printf("This is a multi line comment and will
be ignored: %s\n", yytext); }
\"(.)*\" {printf("%s\n", yytext);return STRING;}
[0-9]+ {printf("%s\n", yytext); return NUMBER;}
";" {printf("%s\n", yytext); return semi;}
"import" {printf("%s\n", yytext); return IMPORT;}
"main" {printf("%s\n", yytext); return MAIN;}
"public" {printf("%s\n", yytext); return PUBLIC;}
"private" {printf("%s\n", yytext); return PRIVATE;}
"protected" {printf("%s\n", yytext); return PROTECTED;}
"static" {printf("%s\n", yytext); return STATIC_KEY;}
"int" {printf("%s\n", yytext); return INT;}
"void" {printf("%s\n", yytext); return VOID;}
"String" {printf("%s\n", yytext); return STRING_KEY;}
"Scanner" {printf("%s\n", yytext); return SCANNER;}
"new" {printf("%s\n", yytext); return NEW_KEY;}
"throws" {printf("%s\n", yytext); return THROWS;}
"System.out.print" {printf("%s\n", yytext); return SYSTEMPRINT;}
"," {printf("%s\n", yytext); return COMMA;}
"=" {printf("=\n"); return EQUAL;}
```

```
"+=" {printf("%s\n", yytext); return SHORT_ADD;}
"-=" {printf("%s\n", yytext); return SHORT_MINUS;}
"*=" {printf("%s\n", yytext); return SHORT_MUL;}
"/=" {printf("%s\n", yytext); return SHORT_DIV;}
"%=" {printf("%s\n", yytext); return SHORT_MOD;}
"++" {printf("%s\n", yytext); return INCREMENT;}
"--" {printf("%s\n", yytext); return DECREMENT;}
"while" {printf("%s\n", yytext); return WHILE;};
"if" {printf("%s\n", yytext); return IF;};
"for" {printf("%s\n", yytext); return FOR;};
"else" {printf("%s\n", yytext); return ELSE;}
\[ {printf("[\n"); return SQUA_OPEN;}
\] {printf("]\n"); return SQUA_CLOSE;}
"-" {printf("-\n"); return MINUS;}
"+" {printf("+\n"); return PLUS;}
"*" {printf("*\n"); return MUL;}
"/" {printf("/\n"); return DIV;}
"%" {printf("%\n"); return MOD;}
">" {printf(">\n"); return GREATER_THAN;}
">=" {printf(">=\n"); return GREATER_THAN_EQUAL;}
"<" {printf("<\n"); return LESSER_THAN;}
"<=" {printf("<=\n"); return LESSER_THAN_EQUAL;}
"==" {printf("==\n"); return EQUALS;}
"!=" {printf("!=\n"); return NOT_EQUALS;}
"(" {printf("(\n"); return OPEN_BRAC;}
")" {printf(")\n"); return CLOSE_BRAC;}
"{" {printf("{\n"); return OPEN_FLOW;}
"}" {printf("}\n"); return CLOSE_FLOW;}
"class" {printf("%s\n", yytext); return CLASS;}
([a-zA-Z]*\.)+([a-zA-Z]*) {printf("%s\n", yytext); return API_REF;}
([a-zA-Z]*\.)+(\*) {printf("%s\n", yytext); return API_REF;}
[a-zA-Z_][a-zA-Z0-9_]* {printf("%s\n", yytext); return ID;}
%%
int yywrap(){
     return 1;
}
```

b) Bison File ‘a.y’

```
%{
     #include <stdio.h>
     #include <stdlib.h>
     int yylex();
     int yyerror();
     extern FILE *yyin;
    #include "variable.h"
%}
%locations %define api.pure full
%token semi PUBLIC PRIVATE PROTECTED STATIC_KEY MAIN API_REF IMPORT
UNTIL ID INT VOID STRING_KEY
NUMBER STRING DO END WHILE REPEAT IF EQUAL FOR IN ELSE ELSEIF THEN
BREAK RETURN THROWS NEW_KEY
LOCAL COLON DOT COMMA MINUS PLUS GREATER_THAN GREATER_THAN_EQUAL
LESSER_THAN LESSER_THAN_EQUAL
EQUALS NOT_EQUALS MOD POWER AND OR OPEN_FLOW CLOSE_FLOW OPEN_BRAC
CLOSE_BRAC SQUA_OPEN CLASS SYSTEMPRINT
SQUA_CLOSE TRUE FALSE SHORT_ADD SHORT_MINUS SHORT_MUL SHORT_DIV
SHORT_MOD INCREMENT DECREMENT SCANNER
%left      OR
%left      AND
%left      LESSER_THAN LESSER_THAN_EQUAL GREATER_THAN
GREATER_THAN_EQUAL EQUALS NOT_EQUAL
%left      PLUS MINUS
%left      MUL DIV MOD
%right     NOT HASH
%right     POWER
%%
SOURCE_CODE     : IMPORT_STMT CLASS_DCLR
                | CLASS_DCLR
                ;

IMPORT_STMT     : IMPORT API_REF semi IMPORT_STMT
                |
                ;
```

```
CLASS_DCLR      : ACCESS CLASS ID OPEN_FLOW CLASS_DEF CLOSE_FLOW
CLASS_DCLR
                |
                ;

CLASS_DEF       : MAIN_METHOD
                |
                ;

MAIN_METHOD     : ACCESS STATIC TYPE MAIN OPEN_BRAC PARAM CLOSE_BRAC
EXCEPTIONS OPEN_FLOW STMT CLOSE_FLOW
                ;

ACCESS          : PUBLIC
                | PRIVATE
                | PROTECTED
                |
                ;

STATIC          : STATIC_KEY
                |
                ;

PARAM           : TYPE PARAM_DASH
                |
                ;

PARAM_DASH      : ID SQUA_OPEN SQUA_CLOSE
                | SQUA_OPEN SQUA_CLOSE ID
                ;

TYPE            : INT
                | VOID
                | STRING_KEY
                | SCANNER
                ;

EXCEPTIONS      : THROWS API_REF
                |
                ;

STMT            : VAR_DCLR semi STMT
                | CALL_STMT semi STMT
                | ASSIGN_STMT semi STMT
                | CTRL_STMT STMT
                | OPEN_FLOW STMT CLOSE_FLOW
                | OUTPUT_STATEMENT semi
                |
                ;

OUTPUT_STATEMENT: SYSTEMPRINT OPEN_BRAC STRING CSIDS CLOSE_BRAC
                ;

CSIDS           : COMMA ID CSIDS
                |
                ;

VAR_DCLR        : TYPE ID VAR_INIT MORE_VAR_DCLR
                ;

VAR_INIT        : EQUAL VAR_INITIAL_VAL
                |
                ;

VAR_INITIAL_VAL : EXPR
                | NEW
                ;

NEW             : NEW_KEY TYPE OPEN_BRAC NEW_DASH
                ;

NEW_DASH        : CLOSE_BRAC
                | NEW CLOSE_BRAC
                | API_REF CLOSE_BRAC
                ;

MORE_VAR_DCLR   : COMMA ID VAR_INIT MORE_VAR_DCLR
                |
                ;

CALL_STMT       : METHOD_NAME OPEN_BRAC ARGS CLOSE_BRAC
                ;

METHOD_NAME     : API_REF
                ;

ARGS            : ARG MORE_ARGS
                ;

MORE_ARGS       : COMMA ARGS
                | PLUS ARGS
                |
                ;

ARG             : CALL_STMT
                | STRING
                | ID
                |
                ;

ASSIGN_STMT     : ID ASSIGN_STMT_DASH
                ;

ASSIGN_STMT_DASH: EQUAL ASSIGN_SRC
                | ASSIGN_OP EXPR
                ;

ASSIGN_SRC      : CALL_STMT
                | EXPR
                ;

ASSIGN_OP       : SHORT_ADD
                | SHORT_MINUS
                | SHORT_MUL
                | SHORT_DIV
                | SHORT_MOD
                ;
```

```
EXPR            : TERM EXPR_DASH
                ;

EXPR_DASH       : ADD_OP TERM EXPR_DASH
                |
                ;

ADD_OP          : PLUS
                | MINUS
                ;

TERM            : FACTOR TERM_DASH
                ;

TERM_DASH       : MUL_OP FACTOR TERM_DASH
                |
                ;

MUL_OP  : MUL
                | DIV
                | MOD
                ;

FACTOR   : OPEN_BRAC EXPR CLOSE_BRAC
                | ID
                | NUMBER
                ;

CTRL_STMT  : IF_STMT
                | WHILE_STMT
                | FOR_STMT
                ;

IF_STMT : IF OPEN_BRAC TEST CLOSE_BRAC STMT_SINGLE ELSE_PART
                ;

ELSE_PART : ELSE STMT_SINGLE
                |
                ;

STMT_SINGLE     : VAR_DCLR semi
                | CALL_STMT semi
                | ASSIGN_STMT semi
                | CTRL_STMT semi
                | OPEN_FLOW STMT CLOSE_FLOW
                ;

TEST    : EXPR TEST_OP TEST_NORM
                ;

TEST_NORM  : ID
                | NUMBER
                | OPEN_BRAC EXPR CLOSE_BRAC
                ;

TEST_OP         : LESSER_THAN
                | GREATER_THAN
                | GREATER_THAN_EQUAL
                | LESSER_THAN_EQUAL
                | EQUALS
                | NOT_EQUALS
                ;

WHILE_STMT      :  OPEN_BRAC TEST CLOSE_BRAC STMT_SINGLE
                ;

FOR_STMT        : FOR OPEN_BRAC ASSIGN_STMT semi TEST semi U_EXPR
CLOSE_BRAC STMT_SINGLE
                ;

U_EXPR          : ID INCREMENT
                | ID DECREMENT
                | INCREMENT ID
                | DECREMENT ID
                ;

%%
```

```
int yyerror(YYLTYPE *locp, char const *msg){
     printf("Invalid Expression: %s \nAT LINE NUMBER %d \n", msg, 
lineno);
     return 1;
}
int main(){
     yyin = fopen(" ’Your path here’/JavaMiniParser/first.java", 
"r");
     do{
          if(yyparse()){
               exit(0);
          }
     }while(!feof(yyin));
     printf("No errors, Program successefully Parsed.\n");
    return 1;
}
```

2) Code for Symbol Table generation

a) The Lex file ‘tokgen.l’

```
%{
     #include<stdio.h>
     #include<string.h>
     #include<stdlib.h>
     #include "symtab.h"
     #define YY_DECL Tokenptr yylex(void)

     int l=1, c=1, scope=0, fa=0;
     char dtype[10];
     Tokenptr tp;
     Tokenptr allocToken()
     {
          Tokenptr tp;
          tp = (Tokenptr)malloc(sizeof(struct Token));
          tp -> lexeme = (char*)malloc(20*sizeof(char));
          tp -> index = 0;
          tp -> type = EOFILE;
          return tp;
     }
     void setTokenArgs(Tokenptr tp, char *lexeme, int row, int col, 
enum tokenType type)
     {
          if(tp==NULL)
               return;
          strcpy(tp->lexeme, lexeme);
          tp->row = row;
          tp->col = col;
          tp->type = type;
     }
     char* getType(enum tokenType t)
     {
          switch(t)
          {
               case 0: return "LITERAL";
               case 1: return "KEYWORD";
               case 2: return "NUMBER";
               case 3: return "IDENTIFIER";
               case 4: return "SYMBOL";
               case 5: return "AOP";
               case 6: return "LOP";
               case 7: return "RELOP";
               case 8: return "FUNCTION";
               default: return "";
          }
     }
     void printToken(Tokenptr tp)
     {
          printf("<%s, %d, %d, %d, %s>\n", tp->lexeme, tp->row, 
tp->col, tp->index, getType(tp->type));
     }
%}
%%

"import"(.)*"\n"
{
     l++;
```

```
}
"//"(.)*"\n" {
     l++;
}
"/*"([^*]|"*"[^/])*"*/"  {
     for(int i=0; i<yyleng; i++)
          if(yytext[i]=='\n')
          {
               l++;
               c=1;
          }
}
\"[^\"]*\" {
     tp = allocToken();
     setTokenArgs(tp, yytext, l, c, LITERAL);
     c+=yyleng;
     return tp;
}
"public"|"private"|"class"|"String"|"static"|"int"|"char"|"if"|"else
"|"while"|"void"|"for"|"return"|"float"|"double" {
     if(strcmp(yytext, "int") == 0||strcmp(yytext, "char") ==
0||strcmp(yytext, "float") == 0||strcmp(yytext, "double") == 0
||strcmp(yytext, "void") == 0)
          strcpy(dtype, yytext);
     tp = allocToken();
     setTokenArgs(tp, yytext, l, c, KEYWORD);
     c+=yyleng;
     return tp;
}
[a-zA-Z_][a-zA-Z0-9_]* {
     tp = allocToken();
     setTokenArgs(tp, yytext, l, c, IDENTIFIER);
     c+=yyleng;
     return tp;
}

[-]?([0-9]*[.])?[0-9]+ {
     tp = allocToken();
     setTokenArgs(tp, yytext, l, c, NUMBER);
     c+=yyleng;
     return tp;
}

"+"|"="|"-"|"*"|"/"|"%"|"+="|"-="|"*="|"/="|"%="|"++"|"--" {
     tp = allocToken();
     setTokenArgs(tp, yytext, l, c, AOP);
     c+=yyleng;
     return tp;
}
"&"|"|"|"&&"|"||"|"!" {
     tp = allocToken();
     setTokenArgs(tp, yytext, l, c, LOP);
     c+=yyleng;
     return tp;
}
">"|"<"|"!="|">="|"<="|"==" {
     tp = allocToken();
     setTokenArgs(tp, yytext, l, c, RELOP);
     c+=yyleng;
     return tp;
}
\t|" " {
    c++;
}
\n {
     l++;
     c=1;
}
. {
     tp = allocToken();
     setTokenArgs(tp, yytext, l, c, SYMBOL);
     if(strcmp(yytext, ";")==0)
          dtype[0]='\0';
     else if(yytext[0] == '{')
          scope++;
     else if(yytext[0] == '}')
          scope--;
     c++;
     return tp;
}
%%
int main(int argc, char **argv)
{
     yyin = fopen(" ‘Your path here ’/JavaMiniParser/first.java",
"r");
     Tokenptr tk;
     printf("<Lexeme, Row, Col, Index, Type>\n");
     while(tk = yylex())
     {
          printToken(tk);
          if(tk -> type == IDENTIFIER)
          {
               char tempdt[10];
               strcpy(tempdt, dtype);
               Tokenptr temp = yylex();
               printToken(temp);
               if(strcmp(temp -> lexeme, "(") == 0)
               {
                    // if(dtype[0] == '\0')
                    //   break;
                    char args[10][100];
                    int i = 0;
                    while(strcmp(temp -> lexeme, ")")!=0)
                    {
                         temp = yylex();
                         printToken(temp);
                         if(temp -> type == IDENTIFIER || temp ->
type == LITERAL || temp -> type == NUMBER)
                         {
                              strcpy(args[i], temp ->lexeme);
                              i++;
                         }
                         if(temp -> type == KEYWORD)
                         {
                              strcpy(dtype, temp -> lexeme);
                         }
                         if(temp -> type == IDENTIFIER)
                         {
                              temp -> index = Insert(temp, 0, dtype,
scope, NULL, '\0');
                         }
                    }
                    tk -> index = Insert(tk, 1, tempdt, scope, args,
i);
                    dtype[0] = '\0';
               }
               else
                    tk -> index = Insert(tk, 0, tempdt, scope, NULL,
'\0');
          }
     }
     fclose(yyin);
     Display();
     return 0;
}
int yywrap()
{
     return 1;
}
```

b) The header 'file symtab.h'

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define tableLen 509

enum tokenType {EOFILE=-1, LITERAL, KEYWORD, NUMBER, IDENTIFIER,
SYMBOL, AOP, LOP, RELOP};

struct Token
{
     char *lexeme;
     int index;
     int row, col;
     enum tokenType type;
```

```
};
typedef struct Token* Tokenptr;

struct ListElement
{
     Tokenptr tp;
     int hashval, argc;
     char name[10], type[10], scope, args[10][100], ret[10],
func[10];
     struct ListElement *next;
};
typedef struct ListElement* LEptr;

LEptr TABLE[tableLen];

void initialize()
{
     for(int i=0; i<tableLen; i++)
          TABLE[i] = NULL;
}

void Display()
{
     printf("HASH\tID/FUNC\t\tNAME\tTYPE\tSCOPE\tRET\tARG
C\tARGS\n");
     for(int i=0; i<tableLen; i++)
     {
          LEptr temp = TABLE[i];
          while(temp)
          {
               printf("%d\t%s\t%s\t%s\t%c\t%s\t", temp -> hashval,
temp -> func, temp -> name, temp -> type, temp -> scope, temp ->
ret);
               if(strcmp(temp -> func, "FUNCTION")==0)
                    printf("%d\t", temp -> argc);
               for(int i = 0; i<temp ->argc; i++)
               {
                    printf("%s", temp -> args[i]);
                    if(i!=temp->argc-1)
```

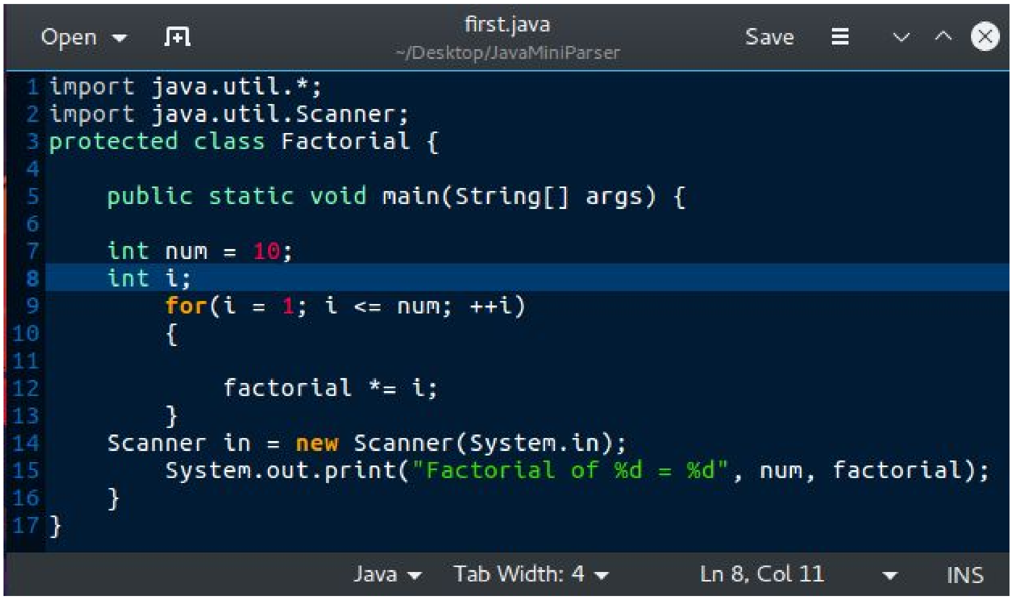
```
                         printf(", ");
               }
               printf("\n");
               temp = temp->next;
          }
     }
}

int Hash(char *str)
{
     int s=0, p=31, pw=1;
     for(int i=0; i<strlen(str); i++)
     {
          s = (s + (str[i])*pw) %tableLen;//-'a'+1
          pw = (pw*p)%tableLen;
     }
     return s;
}

int Search(char *str)
{
     int x = Hash(str);
     LEptr temp = TABLE[x];
     while(temp)
     {
          if(strcmp(temp->tp->lexeme, str)==0)
               return 1;
          temp = temp->next;
     }
     return 0;
}

int Insert(Tokenptr tp, int func, char* dtype, int scope, char
args[10][100], int argc)
{
     if(Search(tp->lexeme)==1)
          return Hash(tp->lexeme);
     int x = Hash(tp->lexeme);
     // printf("%d\n", x);
```

```
LEptr le = (LEptr)malloc(sizeof(struct ListElement));
le -> tp = tp;
strcpy(le -> name, tp->lexeme);
le -> hashval = x;
if(func == 0)
{
     strcpy(le -> func, "IDENTIFIER");
     le -> argc = 0;
     le -> ret[0] = '\0';
     if(scope == 0)
          le -> scope = 'G';
     else
          le -> scope = 'L';
     strcpy(le -> type, dtype);
}
else if(func == 1)
{
     strcpy(le -> func, "FUNCTION");
     le -> scope = ' ';
     le -> type[0] = '\0';
     strcpy(le -> ret, dtype);
     if(le -> ret[0] == '\0')
          strcpy(le -> ret, "void");
     le -> argc = argc;
     for(int i=0; i<argc; i++)
          strcpy(le -> args[i], args[i]);
}
le -> next = NULL;
if(TABLE[x]==NULL)
     TABLE[x] = le;
else
{
     LEptr temp = TABLE[x];
     x += tableLen;
     while(temp->next)
     {
          temp = temp->next;
          x += tableLen;
     }
```

```
          // temp -> hashval = x;
          temp->next = le;
     }
     return x;
                                                                          }
//name, type, size, scope, no of args, args, ret type
```

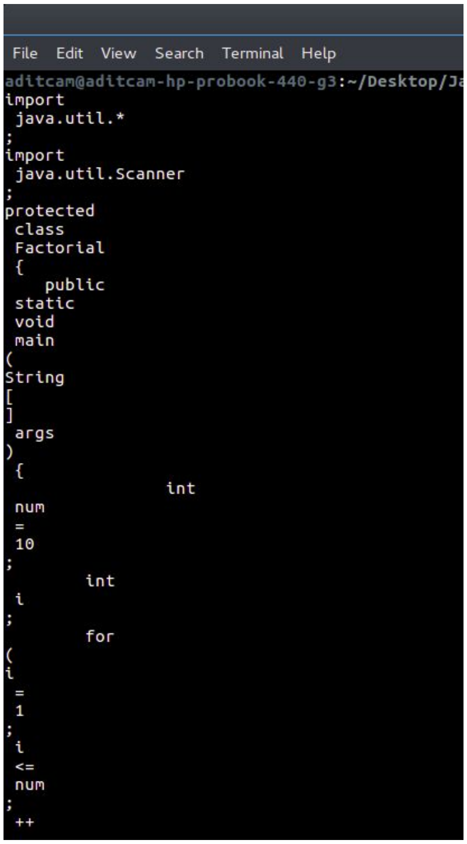
Input and Output samples

1)Sample Input file and corresponding output (For correct input)



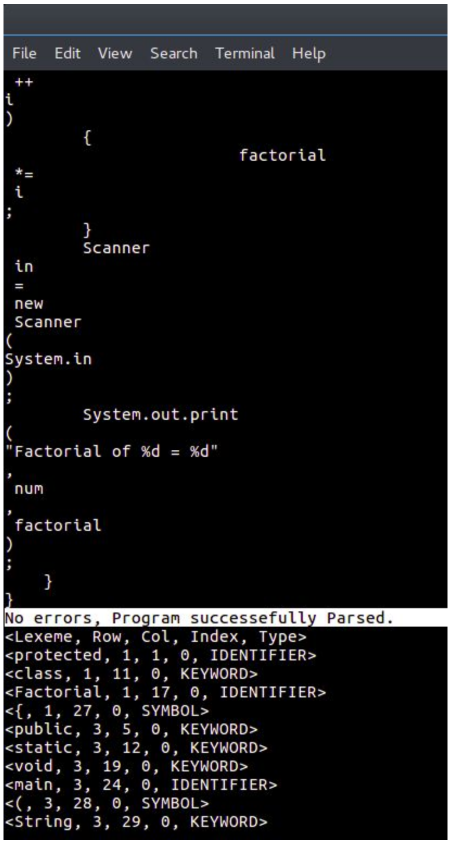
```
import java.util.*;
import java.util.Scanner;
protected class Factorial {

    public static void main(String[] args) {

    int num = 10;
    int i;
        for(i = 1; i <= num; ++i)
        {

            factorial *= i;
        }
    Scanner in = new Scanner(System.in);
        System.out.print("Factorial of %d = %d", num, factorial);
    }
}
```

1)



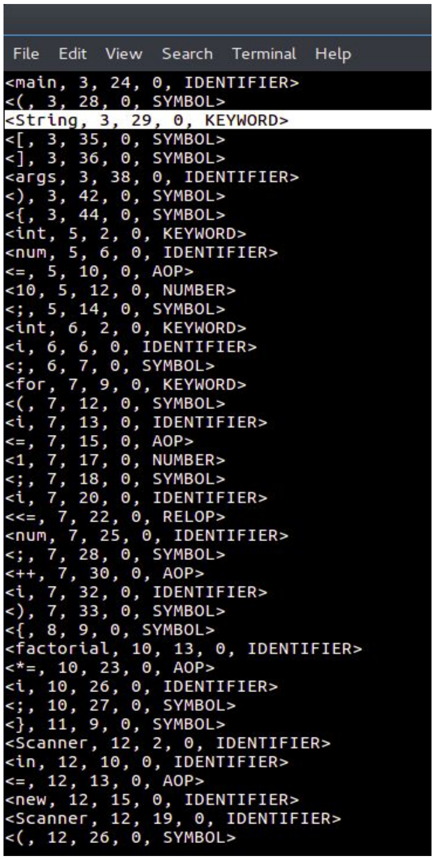
```
aditcam@aditcam-hp-probook-440-g3:~/Desktop/Ja
import
 java.util.*
;
import
 java.util.Scanner
;
protected
 class
 Factorial
 {
    public
 static
 void
 main
(
String
[
]
 args
)
 {
                int
 num
 =
 10
;
        int
 i
;
        for
(
i
 =
 1
;
 i
 <=
 num
;
 ++
```

2)



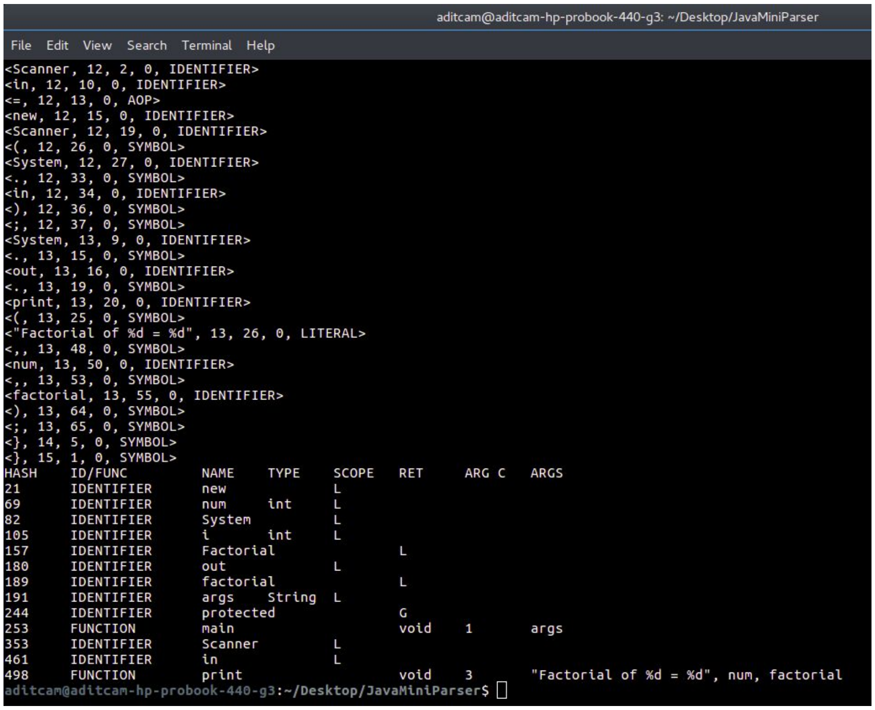
```
 ++
i
)
        {
                        factorial
 *=
 i
;
        }
        Scanner
 in
 =
 new
 Scanner
(
System.in
)
;
        System.out.print
(
"Factorial of %d = %d"
,
 num
,
 factorial
)
;
    }
}
No errors, Program successefully Parsed.
<Lexeme, Row, Col, Index, Type>
<protected, 1, 1, 0, IDENTIFIER>
<class, 1, 11, 0, KEYWORD>
<Factorial, 1, 17, 0, IDENTIFIER>
<{, 1, 27, 0, SYMBOL>
<public, 3, 5, 0, KEYWORD>
<static, 3, 12, 0, KEYWORD>
<void, 3, 19, 0, KEYWORD>
<main, 3, 24, 0, IDENTIFIER>
<(, 3, 28, 0, SYMBOL>
<String, 3, 29, 0, KEYWORD>
```

3)



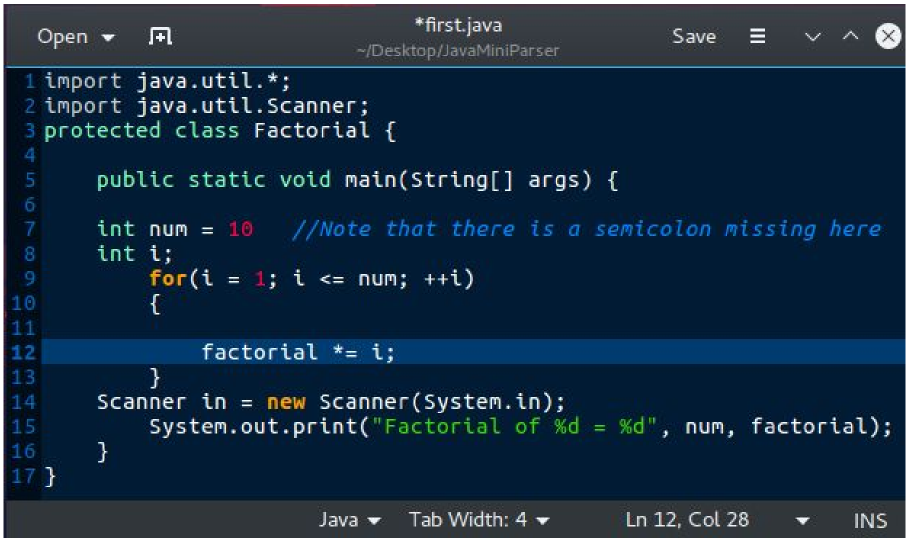
```
<main, 3, 24, 0, IDENTIFIER>
<(, 3, 28, 0, SYMBOL>
<String, 3, 29, 0, KEYWORD>
<[, 3, 35, 0, SYMBOL>
<], 3, 36, 0, SYMBOL>
<args, 3, 38, 0, IDENTIFIER>
<), 3, 42, 0, SYMBOL>
<{, 3, 44, 0, SYMBOL>
<int, 5, 2, 0, KEYWORD>
<num, 5, 6, 0, IDENTIFIER>
<=, 5, 10, 0, AOP>
<10, 5, 12, 0, NUMBER>
<;, 5, 14, 0, SYMBOL>
<int, 6, 2, 0, KEYWORD>
<i, 6, 6, 0, IDENTIFIER>
<;, 6, 7, 0, SYMBOL>
<for, 7, 9, 0, KEYWORD>
<(, 7, 12, 0, SYMBOL>
<i, 7, 13, 0, IDENTIFIER>
<=, 7, 15, 0, AOP>
<1, 7, 17, 0, NUMBER>
<;, 7, 18, 0, SYMBOL>
<i, 7, 20, 0, IDENTIFIER>
<<=, 7, 22, 0, RELOP>
<num, 7, 25, 0, IDENTIFIER>
<;, 7, 28, 0, SYMBOL>
<++, 7, 30, 0, AOP>
<i, 7, 32, 0, IDENTIFIER>
<), 7, 33, 0, SYMBOL>
<{, 8, 9, 0, SYMBOL>
<factorial, 10, 13, 0, IDENTIFIER>
<*=, 10, 23, 0, AOP>
<i, 10, 26, 0, IDENTIFIER>
<;, 10, 27, 0, SYMBOL>
<}, 11, 9, 0, SYMBOL>
<Scanner, 12, 2, 0, IDENTIFIER>
<in, 12, 10, 0, IDENTIFIER>
<=, 12, 13, 0, AOP>
<new, 12, 15, 0, IDENTIFIER>
<Scanner, 12, 19, 0, IDENTIFIER>
<(, 12, 26, 0, SYMBOL>
```

4)


aditcam@aditcam-hp-probook-440-g3: ~/Desktop/JavaMiniParser
File Edit View Search Terminal Help

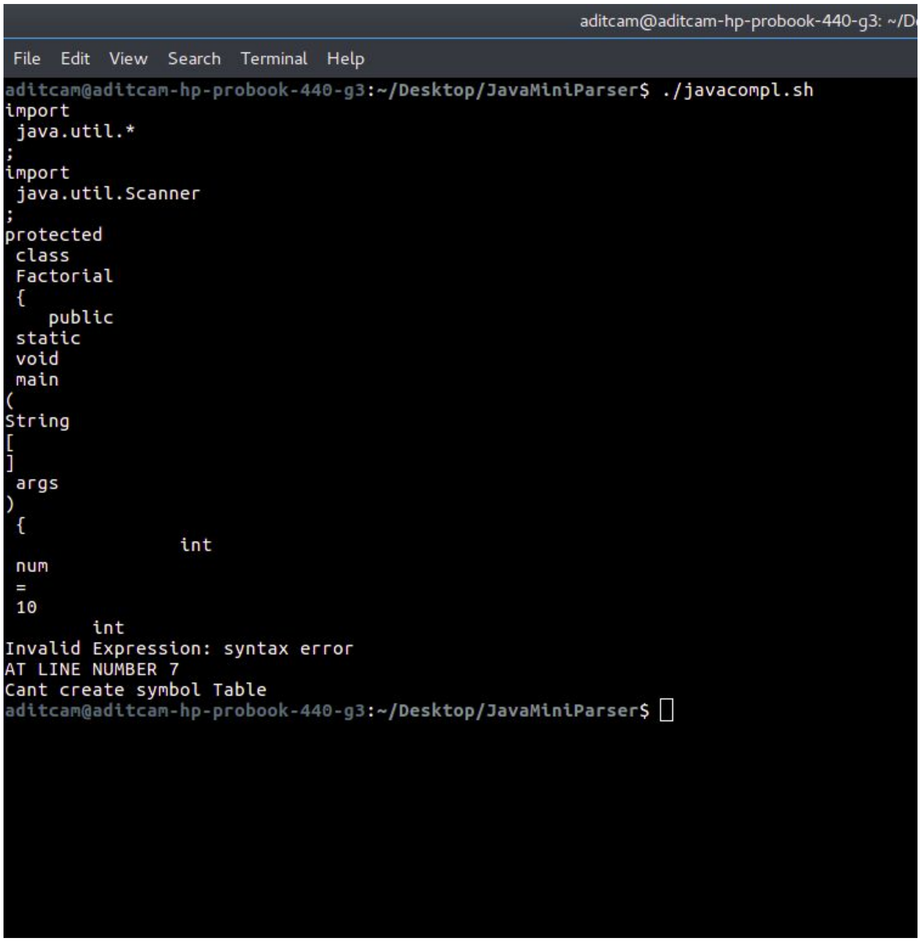
```
<Scanner, 12, 2, 0, IDENTIFIER>
<in, 12, 10, 0, IDENTIFIER>
<=, 12, 13, 0, AOP>
<new, 12, 15, 0, IDENTIFIER>
<Scanner, 12, 19, 0, IDENTIFIER>
<(, 12, 26, 0, SYMBOL>
<System, 12, 27, 0, IDENTIFIER>
<., 12, 33, 0, SYMBOL>
<in, 12, 34, 0, IDENTIFIER>
<), 12, 36, 0, SYMBOL>
<;, 12, 37, 0, SYMBOL>
<System, 13, 9, 0, IDENTIFIER>
<., 13, 15, 0, SYMBOL>
<out, 13, 16, 0, IDENTIFIER>
<., 13, 19, 0, SYMBOL>
<print, 13, 20, 0, IDENTIFIER>
<(, 13, 25, 0, SYMBOL>
<"Factorial of %d = %d", 13, 26, 0, LITERAL>
<,, 13, 48, 0, SYMBOL>
<num, 13, 50, 0, IDENTIFIER>
<,, 13, 53, 0, SYMBOL>
<factorial, 13, 55, 0, IDENTIFIER>
<), 13, 64, 0, SYMBOL>
<;, 13, 65, 0, SYMBOL>
<}, 14, 5, 0, SYMBOL>
<}, 15, 1, 0, SYMBOL>
HASH    ID/FUNC         NAME    TYPE    SCOPE   RET     ARG C   ARGS
21      IDENTIFIER      new             L
69      IDENTIFIER      num     int     L
82      IDENTIFIER      System          L
105     IDENTIFIER      i       int     L
157     IDENTIFIER      Factorial               L
180     IDENTIFIER      out             L
189     IDENTIFIER      factorial               L
191     IDENTIFIER      args    String  L
244     IDENTIFIER      protected               G
253     FUNCTION        main                    void    1       args
353     IDENTIFIER      Scanner         L
461     IDENTIFIER      in              L
498     FUNCTION        print                   void    3       "Factorial of %d = %d", num, factorial
aditcam@aditcam-hp-probook-440-g3:~/Desktop/JavaMiniParser$
```

2)Sample Input file (For incorrect input)



```java
import java.util.*;
import java.util.Scanner;
protected class Factorial {

    public static void main(String[] args) {

    int num = 10    //Note that there is a semicolon missing here
    int i;
        for(i = 1; i <= num; ++i)
        {

            factorial *= i;
        }
    Scanner in = new Scanner(System.in);
        System.out.print("Factorial of %d = %d", num, factorial);
    }
}
```

Output :


aditcam@aditcam-hp-probook-440-g3: ~/D
File Edit View Search Terminal Help

```
aditcam@aditcam-hp-probook-440-g3:~/Desktop/JavaMiniParser$ ./javacompl.sh
import
 java.util.*
;
import
 java.util.Scanner
;
protected
 class
 Factorial
 {
    public
 static
 void
 main
(
String
[
]
 args
)
 {
                int
 num
 =
 10
        int
Invalid Expression: syntax error
AT LINE NUMBER 7
Cant create symbol Table
aditcam@aditcam-hp-probook-440-g3:~/Desktop/JavaMiniParser$
```