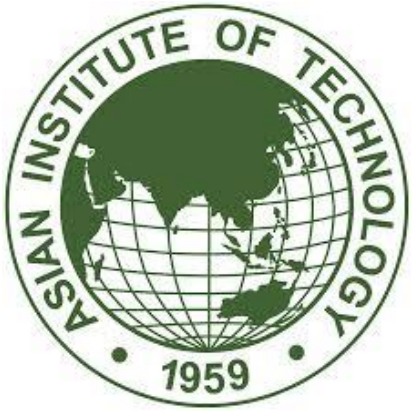


Asian Institute of Technology

School of Engineering and Technology

Computer Science and Information Management

AT70.07 – Programming Languages and Compilers

Project Report

Submitted to: Prof. Phan Min Dung

Submitted by: Group 05

B Jaelse Ronald (st119370)

C Venkata Sai Sravan Kumar (st119143)

P. Swathi Ramya (st119147)

Date: 02 May 2017

Compiler Project

ABSTRACT:-

This report covers the project on Compiler. The compiler is designed for a specific language with given constraints. In this report, a brief description about grammar, semantic rules, type checking and type binding is explained. Screenshots of the demonstration of all the features are shown along with the inputs and outputs given. In the end, it is concluded with the program features and its limitations.

Acquisition and user guidance:-

The source-code files can be found in the folder "[Proj]g05/CUP/ex5". The files are:

Ycalc.cup -- The CUP file where in the Grammar symbols and rules are declared.

Var.java

Astat.java

Lstat.java

SymbolTable.java

Lcalc.flex -- The JFlex file where the lex code for compilation is declared.

Main.java – The Java file where the parser is called.

Build.xml – The XML file.

Sample.inp – The input file where the input expression is read from.

Parser.java & sym.java – CUP generated file. These files get generated when CUP is build.

Lexer.java – Jflex generated file.

Table of contents

1. **Grammar And Semantic Rules**

   Following is the grammar:-

   program -> statement_type

   DECLARATION -> VAR ID_LIST COLON TYPE EQ logic
   | VAR ID_LIST COLON TYPE

   ID_LIST -> ID_LIST COMMA ID | ID

   TYPE -> INT | FLOAT| BOOL

   statement_type -> nstatements | procstatements

   nstatements -> statement_list

   procstatements -> mainproc | procd mainproc

   mainproc -> PROC MAIN LPAREN RPAREN statement_block ENDPROC

   procd -> PROC ID PARA statement_block ENDPROC
   |PROC ID LPAREN RPAREN statement_block ENDPROC

   PARA -> LPAREN PARADEC RPAREN

   PARADEC -> PARADEC COMMA ID COLON TYPE
   | ID COLON TYPE

statement_list -> statement_list statement_part
| statement_part

statement_part -> statement

statement -> DECLARATION |assignment | ifthen | print
| echo | while | for | proccall

statement_block -> statement_list

proccall -> ID LPAREN ARGS RPAREN | ID LPAREN RPAREN

ARGS -> ARGS COMMA logic | logic

while -> WHILE logic statement_block ENDWHILE

for -> FOR LPAREN DECLARATION SEMI logic SEMI statement
RPAREN statement_block ENDFOR

print -> PRINT logic

echo -> ECHO logic | ECHO NEWLINE

Ifthen -> IF logic statement_block ELSE statement_block ENDIF
| IF logic statement_block ENDIF

assignment -> ID EQ logic

logic -> logic LT expr | logic LTE expr | logic GT expr
| logic GTE expr | logic EQEQ expr | logic NEQ expr
| logic AND expr | logic OR expr | expr

expr -> expr PLUS term | expr MINUS term |term

term -> term TIMES factor | term DIVIDE factor | factor

factor -> NOT factor | LPAREN logic RPAREN | NUMBER | ID
| FLOATINGNUMBER | BOOLEAN

num -> 0 | [1-9][0-9]* | 0\.[0-9]* | [1-9][0-9]*\.[0-9]*

Semantic Rules:

Production	Semantic Rules
program -> statement_type	program.val = statement_type.val
DECLARATION -> VAR ID_LIST COLON TYPE EQ logic	DECLARATION.val = var ID_LIST.val ; TYPE.val = logic.val
DECLARATION -> VAR ID_LIST COLON TYPE	DECLARATION.val = var ID_LIST.val ; TYPE.val
ID_LIST -> ID_LIST COMMA ID	ID_LIST.val = ID_LIST.val , ID.val
ID_LIST -> ID	ID_LIST.val = ID.val
TYPE -> INT	TYPE.val = int
TYPE -> FLOAT	TYPE.val = float
TYPE - > BOOL	TYPE.val = bool
statement_type -> nstatements	Statement_type.val = nstatements.val
statement_type -> procstatements	statement_type.val =procstatements.val
nstatements -> statement_list	nstatements.val = statement_list.val
procstatements -> mainproc	procstatements.val = mainproc.val
procstatements -> procd mainproc	procstatements -> procd mainproc
mainproc -> PROC MAIN LPAREN RPAREN statement_block ENDPROC	Mainproc.val = proc main() statement_block.val endproc
procd -> PROC ID PARA statement_block ENDPROC	procd.val = proc ID.val PARA.val statement_block.val endproc
procd -> PROC ID LPAREN RPAREN statement_block ENDPROC	procd.val = proc ID.val () statement_block.val endproc
PARA -> LPAREN PARADEC RPAREN	PARA.val = (PARADEC.val)
PARADEC -> PARADEC COMMA ID COLON TYPE	PARADEC.val -> PARADEC.val ,ID.val ; TYPE.val
PARADEC -> ID COLON TYPE	PARADEC.val -> ID.val ; TYPE.val

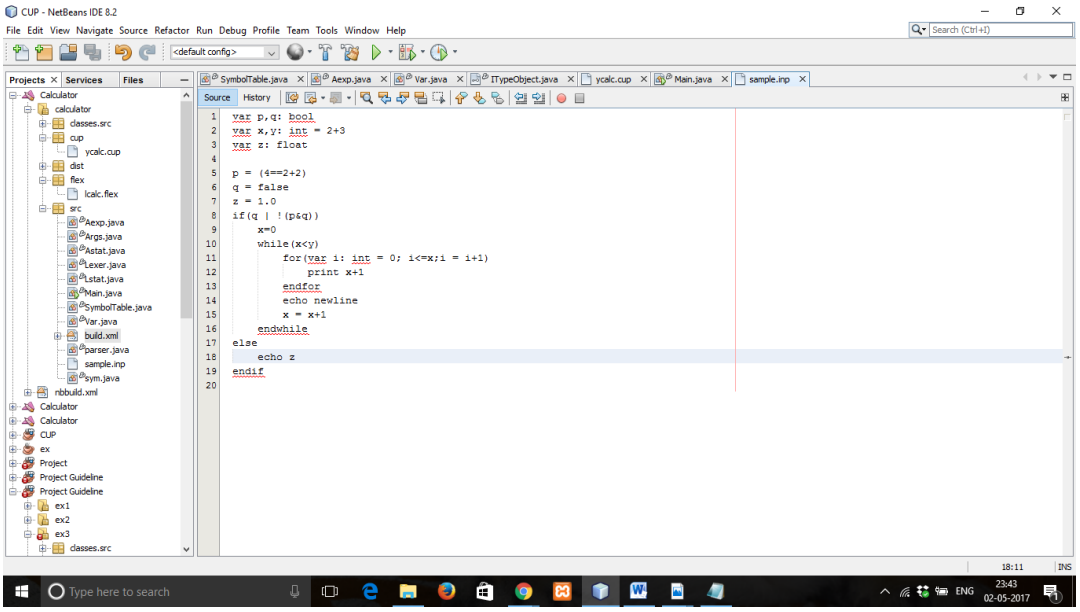
statement_list -> statement_list statement_part	statement_list.val -> statement_list.val statement_part.val
statement_list -> statement_part	statement_list.val -> statement_part.val
statement_part -> statement	statement_part.vall -> statement.val
statement -> DECLARATION	statement.val -> DECLARATION.val
statement -> assignment	statement.val -> assignment.val
statement -> ifthen	statement.val -> ifthen.val
statement -> print	statement.val -> print.val
statement -> echo	statement.val -> echo.val
statement -> while	statement.val -> while.val
statement -> for	statement.val -> for.val
statement -> proccall	statement.val -> proccall.val
statement_block -> statement_list	statement_block.val -> statement_list.val
proccall -> ID LPAREN ARGS RPAREN	proccall.val -> ID.val (ARGS.val)
proccall -> ID LPAREN RPAREN	proccall.val -> ID.val ()
ARGS -> ARGS COMMA logic	ARGS.val -> ARGS.val , logic.val
ARGS -> logic	ARGS.val -> logic.val
while -> WHILE logic statement_block ENDWHILE	while.val -> while logic.val statement_block.val endwhile
for -> FOR LPAREN DECLARATION SEMI logic SEMI statement RPAREN statement_block ENDFOR	for.val -> for (DECLARATION.val ; logic.val ; statement.val) statement_block.val endfor
print -> PRINT logic	print.val -> print logic.val
echo -> ECHO logic	echo.val -> echo logic.val
echo -> ECHO NEWLINE	echo.val -> echo newline
Ifthen -> IF logic statement_block ELSE statement_block ENDIF	Ifthen.val -> if logic.val statement_block.val else statement_block.val endif
Ifthen -> IF logic statement_block	Ifthen.val -> if logic.val

ENDIF	statement_block.val endif
assignment -> ID EQ logic	assignment.val -> ID.val = logic.val
logic -> logic LT expr	logic.val -> logic.val < expr.val
logic -> logic LTE expr	logic.val -> logic.val <= expr.val
logic -> logic GT expr	logic.val -> logic.val > expr.val
logic -> logic GTE expr	logic.val -> logic.val >= expr.val
logic -> logic EQEQ expr	logic.val -> logic.val == expr.val
logic -> logic NEQ expr	logic.val -> logic.val != expr.val
logic -> logic AND expr	logic.val -> logic.val & expr.val
logic -> logic OR expr	logic.val -> logic.val | expr.val
logic -> expr	logic.val -> expr.val
expr -> expr PLUS term	expr.val -> expr.val + term.val
expr -> expr MINUS term	expr.val -> expr.val - term.val
expr -> term	expr.val = term.val
term -> term TIMES factor	term.val = term.val * factor.val
term -> term DIVIDE factor	term.val = term.val / factor.val
term -> factor	term.val = factor.val
factor -> NOT factor	factor.val = ! factor.val
factor ->LPAREN logic RPAREN	factor.val = (logic.val)
factor -> NUMBER	factor.val = number
factor -> ID	factor.val = ID.val
factor -> FLOATINGNUMBER	factor.val = floatingnum
factor -> BOOLEAN	factor.val = booleannum

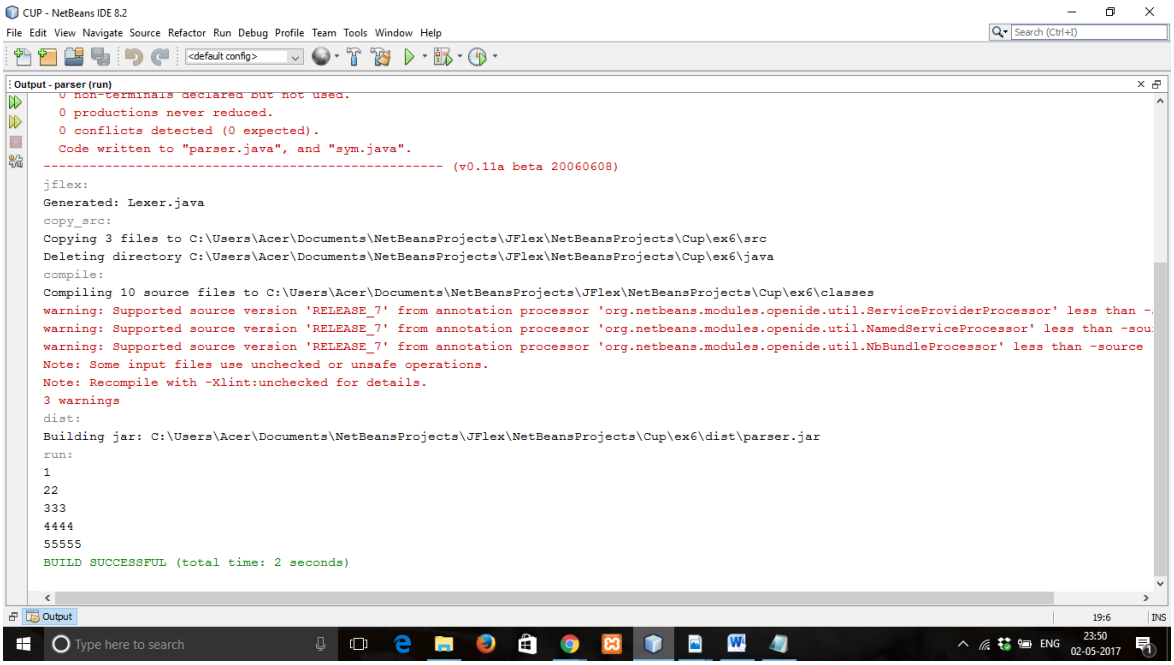
2. Screenshots

2.1 Inputfile 1

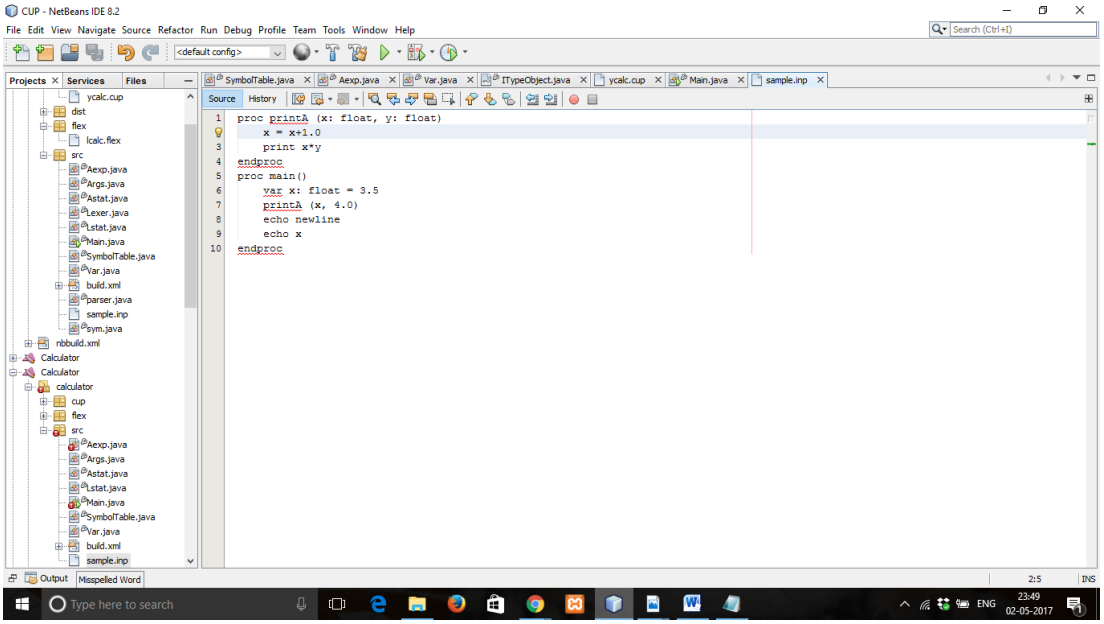
Features including variable declaration, initialization, if then, forloop, whileloop, conditions, printing and echo. Along with ending loops and if then.



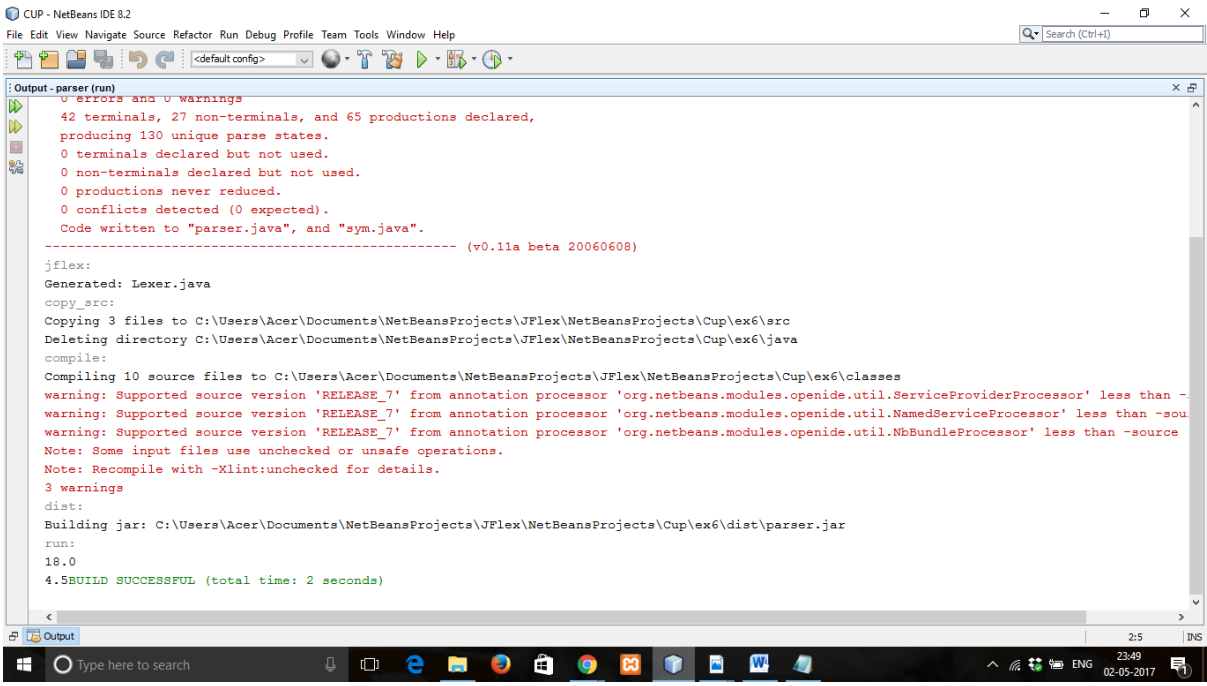
2.2 Output 1



2.3 Input file 2



2.4 Output 2



3. Program Features

This program helps in evaluating expressions like addition, subtraction, multiplication and division. Apart from just mathematical operations, this program also evaluates comparison operators like less than, less than equal, greater than, greater than equal, equality and inequality and logical operators like Conjunction (&), Disjunction (|), Negation (!).

Other features included are: -

- Print statement
- Variable declaration statement
- Assignment statement
- Loop (Both For and While)
- Conditional statement (If-Else)
- Function declaration and Function call
- Type checking

▪ Boolean cannot operate with any other types and error handling is expected.
▪ Integer and float may operate with each other. Type conversion mechanism is provided if the operations between integer and float is possible.
▪ The resulting type of arithmetic expression between int and int is int, while the result of int and float is float. For example, 5/3 = 1 and 5.0/3 = 1.6666.
- Statement block and environment checking
▪ The variable declared within a child environment should not be used in any higher level environment.

4. **Limitations**

   This program is only limited to the above described features. This program does not hold for scope variables. Also it does not have do while loop.

5. **Member Responsibility: -**

   - Grammar Rules Production – Done by all of us.
   - Type checking – Sravan
   - Grammar Checking- Jaelse
   - Report – Swathi Ramya
   - *.java files – Equally distributed among ourselves.
   - While loop – Jaelse