IJAR Programming Language

(It's Just Another Random Programming Language)

Project Milestone 2 - SER502 - Team 13

Project Members:
Saloni Chudgar - 1219586082
Sanjana Mukundan - 1219492846
Chandra Sekhar Sai Sampath Swaroop Atkuri -1219630568
Sudhanva Hanumanth Rao - 1219246938

Language Design :

Data Type	Bool (true or false) ,int (Integers)
Assignment Operator	=
Logical Operator	&& , | | , !!
Conditional Operator	if (expr) {code block} else {code block}
Arithmetic Operator	+, -, *, /, ^, %
Ternary Operator	<expr> ? <code block> : <code block>
Comparison Operators	==, >, <, <=, >=, !=

Iteration	while(expr){code block}
	
	for(<declaration>; <condition>; <increment>) { code block }
	
	for(<declaration> in range (<initial value>, <final value>) { code block }

Standard output	print <expr>;
Reserved Keywords	bool, int, for, if, else, while, in, range, print, true, false, main
Identifier	Must begin with a non-integer character
Symbols	;, {, } ,(,)

Logical Operators:

Operator	Name	Operands	Logical Operation
&&	Logical AND	a&&b	Both expression a and b must be true
||	Logical OR	a||b	Either of expression a or b must be true
!!	Logical NOT	!!a	Negation of a

Assignment Operators:

Operator	Name	Operands	Arithmetic Operation
+	Addition	a+b	Value of b is added to value of a
-	Subtraction	a-b	Value of b is subtracted from value of a
*	Multiplication	a*b	Value of a is multiplied with value of a
/	Division	a/b	Value of a is divided by value of b
^	Power	a^b	Value of a raised to the value of b
%	Modulus	a%b	Remainder when a is divided by b

Comparison Operators:

Operator	Name	Operands	Conditional Evaluation
==	Is equal to	a == b	Checks if b is equal to a
>	Is greater than	a > b	Checks if a is greater than b
<	Is less than	a < b	Checks if a is less than b
<=	Is less than or equals to	a <= b	Checks if a is less than or equal to b
>=	Greater than or equal to	a >= b	Checks if the value of a is greater than or equal to value of b
!=	Is not equal to	a != b	Checks if b is not equal to a

Ternary Operator:

Operator	Name	Operands	Ternary Evaluation
? :	Ternary Operator	<expr> ? a : b	If expression is true, execute a, else execute b

Language grammar :

<program> := main‘(’ ‘)’ ‘{‘ <block> ‘}’

<block> := <block><statement>
| ε

<statement> := <expr>;
| <declaration>;
| <while_cond>
| <if_cond>
| ‘{‘ <block> ‘}’

<expr> := <identifier>
| <assign_expr>
| <arith_expr>
| <comp_expr>
| <logic_expr>
| <tern_expr>

<declaration> := <data_type><identifier>
| <data_type><assign_expr>

<while_cond> := while ‘(‘ <expr> ‘)’ <block>

<for_loop> := for <for_loop_statement> <block>

<if_cond> := if ‘(‘ <expr> ‘)’ <block>
| if ‘(‘ <expr> ‘)’ <block> ‘else’ <block>

<identifier> := <non_interger>
| <identifier><non_integer>
| <identifier><integer>

<assign_expr> := <identifier> ‘=’ <expr>

<arith_expr> := <arith_expr> ‘+’ <arith_expr>
| <arith_expr> ‘-’ <arith_expr>
| <arith_expr> ‘*’ <arith_expr>
| <arith_expr> ‘/’ <arith_expr>
| <arith_expr> ‘%’ <arith_expr>
| <arith_expr> ‘^’ <arith_expr>
| <integer>
| <non_integer>

<comp_expr> := <comp_phrase><comp_operator><comp_phrase>

<comp_phrase> := <identifier>
| <number>
| ‘(’ <arith_expr> ‘)’

<comp_operator> := ‘<’ | ‘<=’ | ‘>’ | ‘>=’ | ‘==’ | ‘!=’

<logic_expr> := <logic_phrase><logic_operator><logic_phrase>
<logic_phrase> := <identifier>
| <number>
| ‘(’ <arith_expr> ‘)’

<logic_operator> := ‘&&’ | ‘||’ | ‘!!’

<print> := <expr> | <string>

<tern_expr> := <expr> ‘?’ <block> ‘:’ <block>

<number> := <number> <integer>
| <integer>

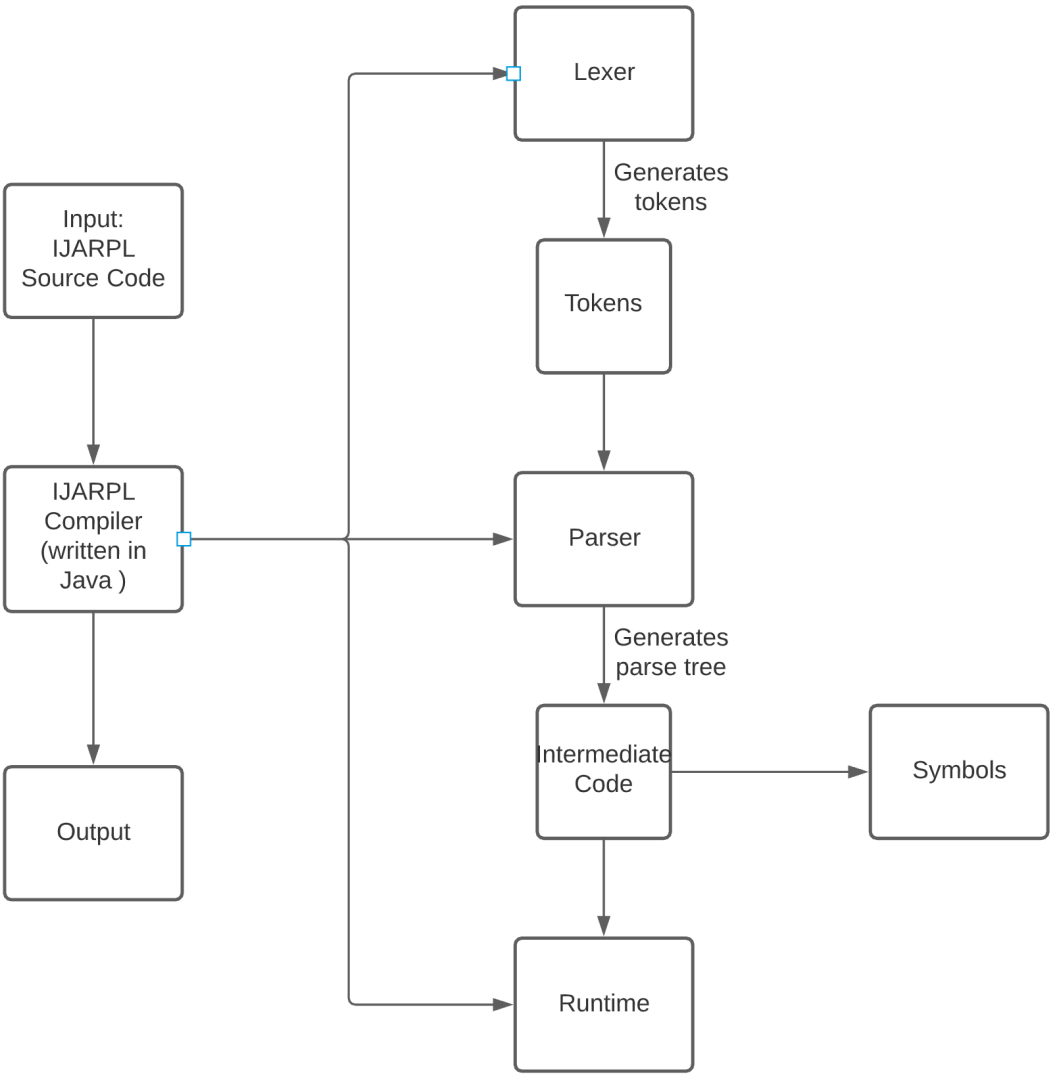
<data_type> := int | bool

<integer> := 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<non_integer> := a | b | c | d .....x | y | z

<bool_val> := true | false

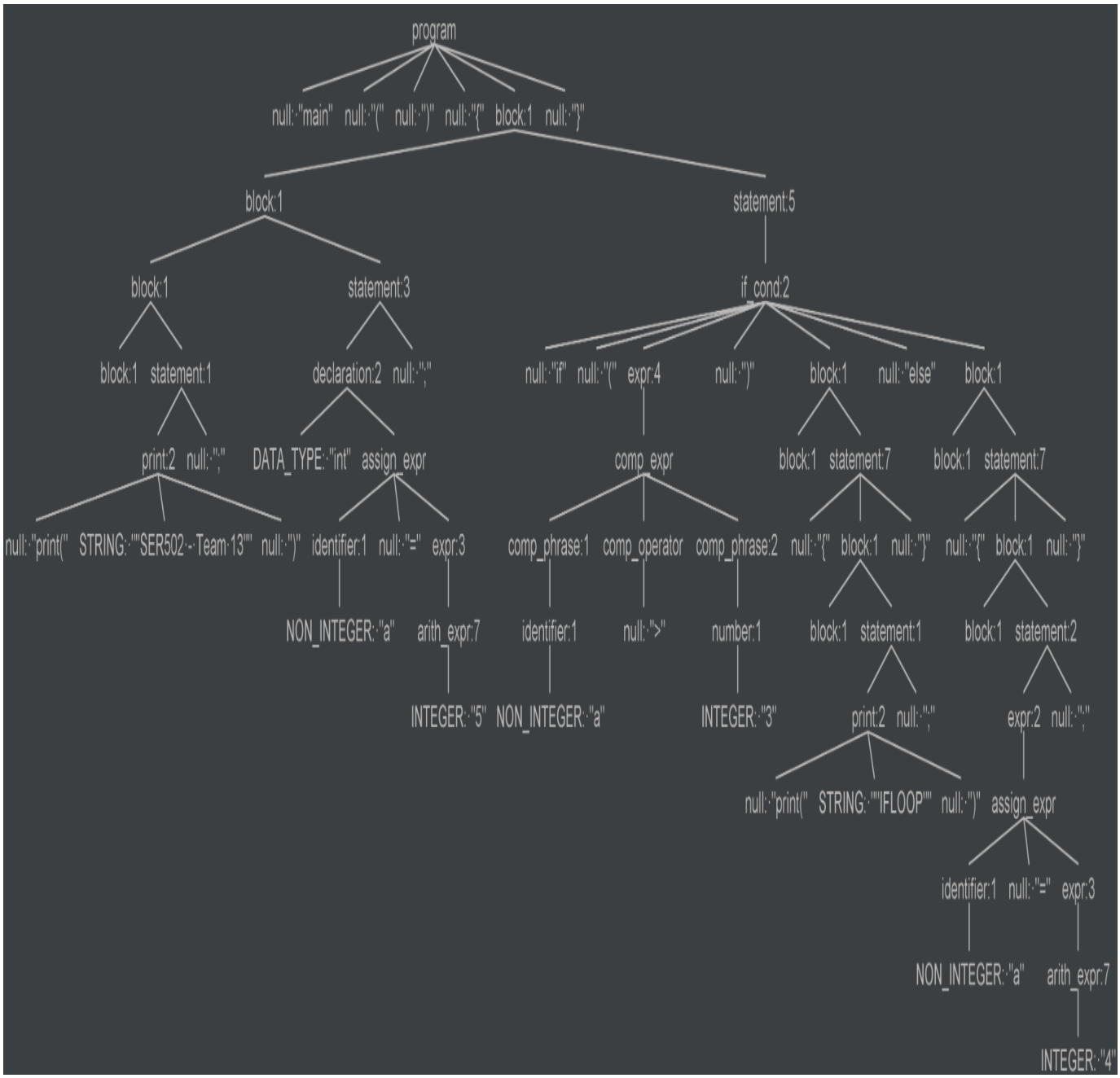
Architecture :



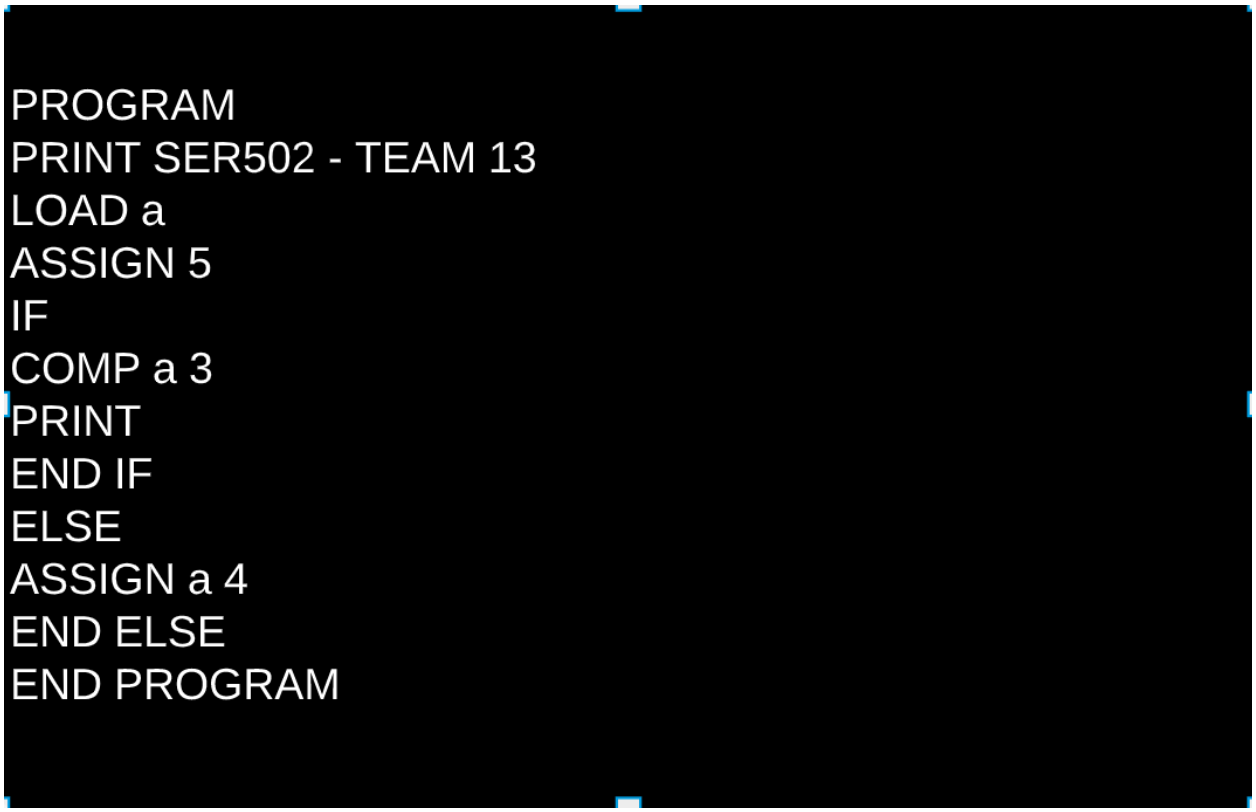
Example Code:

```
main()
{
	print("SER502 - Team 13");
	int a = 5;
	if(a > 3){
	print("IFLOOP");
}
	Else
	{
		a = 4;
	}
}
```

Parse Tree:


program
null:"main" null:"(" null:")" null:"{" block:1 null:"}"
block:1
statement:5
block:1
statement:3
if_cond:2
block:1 statement:1
declaration:2 null:";"
null:"if" null:"(" expr:4 null:")" block:1 null:"else" block:1
print:2 null:";"
DATA_TYPE:"int" assign_expr
comp_expr
block:1 statement:7
block:1 statement:7
null:"print(" STRING:""SER502 - Team 13"" null:")" identifier:1 null:"=" expr:3
comp_phrase:1 comp_operator comp_phrase:2 null:"{" block:1 null:"}"
null:"{" block:1 null:"}"
NON_INTEGER:"a" arith_expr:7
identifier:1 null:">" number:1
block:1 statement:1
block:1 statement:2
INTEGER:"5" NON_INTEGER:"a"
INTEGER:"3"
print:2 null:";"
expr:2 null:";"
null:"print(" STRING:""IFLOOP"" null:")" assign_expr
identifier:1 null:"=" expr:3
NON_INTEGER:"a" arith_expr:7
INTEGER:"4"

Intermediate Code :



ANTLR with Java using IntelliJ

In our code, we integrated Java with Antlr to generate two major aspects of our project :

1. Lexical Analysis
2. Parse Tree Generation

The grammar written will be inputted to the ANTLR and it gives out two output files namely the Lexer and the Parser.

The Lexer generated tokens which are passed to the parser which generates the parse tree. Listener functions are used to traverse over the parse tree.

Then functions in the Listener are edited to implement things to do inside the enter and the exit functions.
Similarly, the IntermediateCodeGenerator is written in a similar manner. The runtime and the compiler are then written in Java code. The output is then given back as a normal java output and displayed on the console.