SER 502 | KoDo

Team 7:

Nishant Jagadeesan 1210319341
Ruthvik Arya Manjunatha 1211286528
Smita Hirve 1210393740
Susmitha Kistamsetty 1211588271

Agenda

- Overview of KoDo
- Language elements
- Program Flow
- Language constructs
- Sample program output
- Demonstration

KoDo: A Simplified Programming Language

- ‘KoDo’ is a japanese word for Code and is easy to pronounce and remember.
- It is a high level language which is compiled to a simple assembly level interpreter language and then output is displayed .
- Design of the language is inspired from Visual Basic and Java.

Features: KoDo

- Simple syntax
- Easy to learn
- No End of statements
- No usage of parenthesis
- Limited keywords
- Supports operation precedence

Tools Used

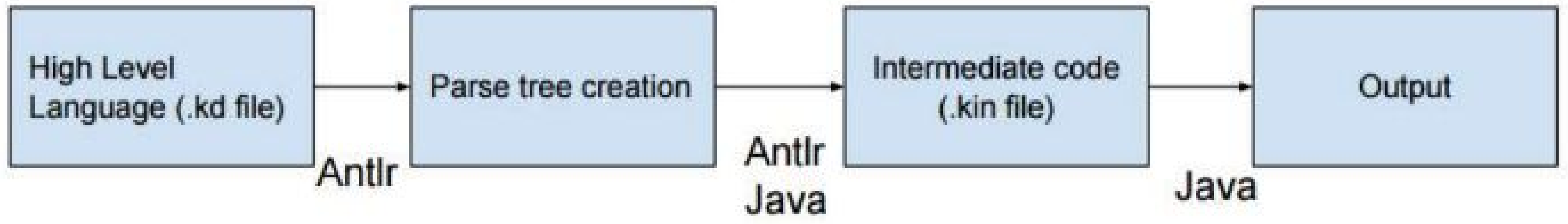
- **ANTLR:** Lexical analysis and Parse Tree generation
- **ANTLR & Java:** Intermediate code generation (Compilation)
- **Java:** Intermediate code parsing and runtime

Language Elements

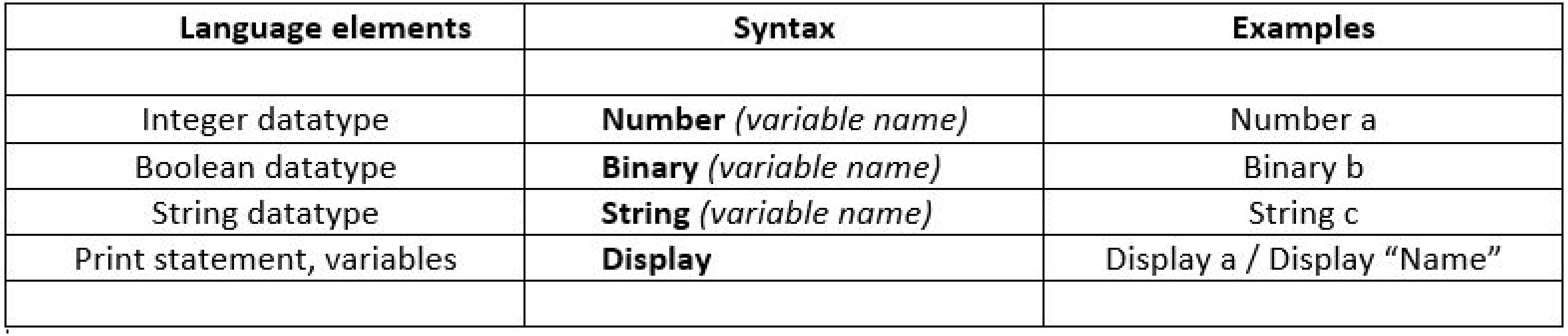
- **Operators:** Arithmetic, Comparison and Logical operators.
- **Data Types:** Integer, Boolean and String
- **Comments:** Single and Multiline
- **Decision Statement:** If then- else loop
- **Control flow statement:** While loop

KoDo: Work Flow

- High level language file - (.kd extension)
- Intermediate language file - (.kin extension)



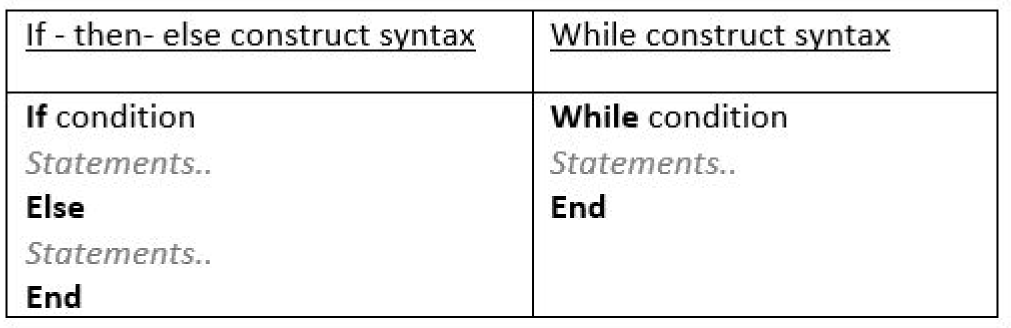
Language Constructs



Language elements	Syntax	Examples
Integer datatype	**Number** *(variable name)*	Number a
Boolean datatype	**Binary** *(variable name)*	Binary b
String datatype	**String** *(variable name)*	String c
Print statement, variables	**Display**	Display a / Display “Name”

Language Constructs Contd..

- Arithmetic operators : + , - ,/, *
- Comparison operators: >=,>, <=, < , ==
- Logical operators: &, !, |
- Looping statements:



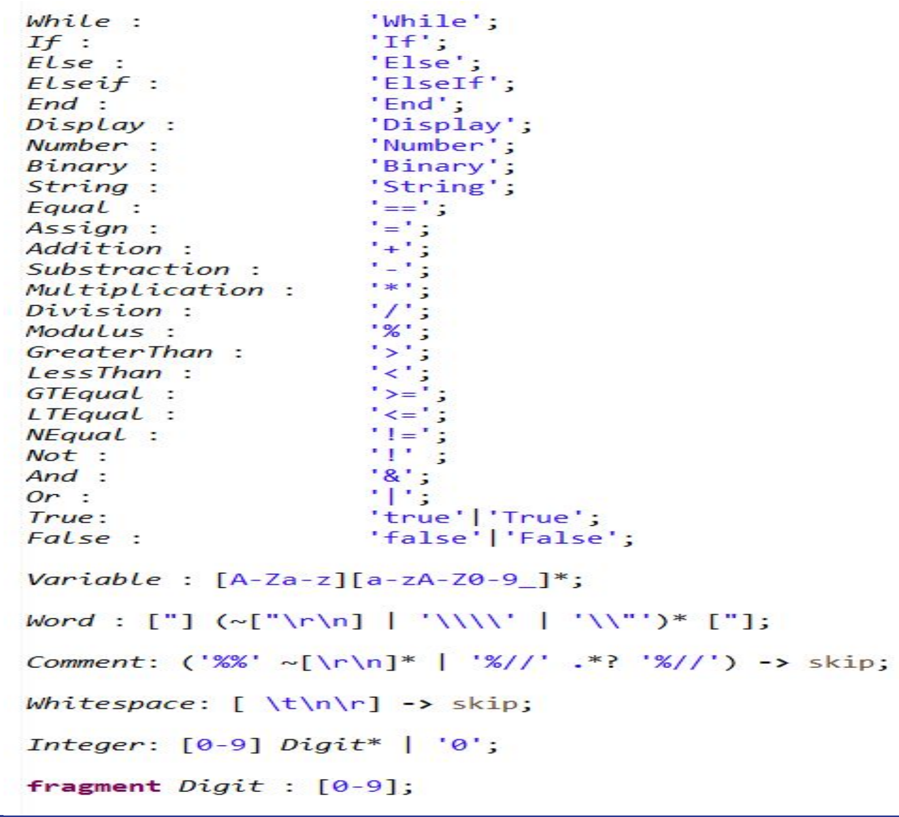
If - then- else construct syntax	While construct syntax
If condition *Statements..* **Else** *Statements..* **End**	**While** condition *Statements..* **End**

Grammar of Kodo

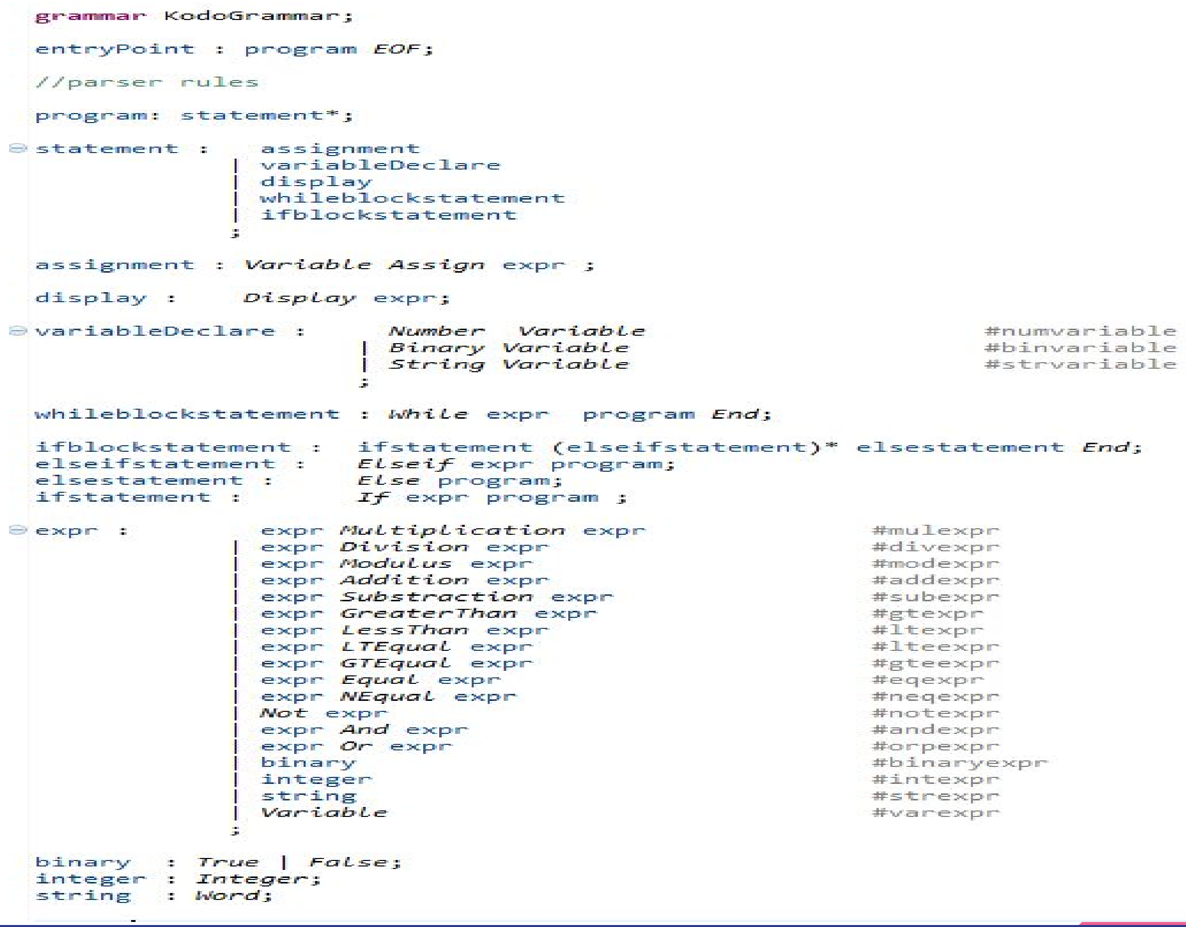
- Context free grammar (BNF) is used
- BaseListener is used for translation to intermediate code
- Enter and exit functions of every production rule in baseListener contains the necessary code.
- Sequence of operation

Read high level language file -> Tokens generation -> Parse tree generation -> Intermediate code file generation

KoDo Grammar: Lexer Rules & Parser Rules



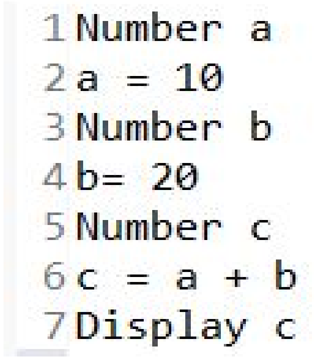
```
While :                    'While';
If :                       'If';
Else :                     'Else';
Elseif :                   'ElseIf';
End :                      'End';
Display :                  'Display';
Number :                   'Number';
Binary :                   'Binary';
String :                   'String';
Equal :                    '==';
Assign :                   '=';
Addition :                 '+';
Substraction :             '-';
Multiplication :           '*';
Division :                 '/';
Modulus :                  '%';
GreaterThan :              '>';
LessThan :                 '<';
GTEqual :                  '>=';
LTEqual :                  '<=';
NEqual :                   '!=';
Not :                      '!' ;
And :                      '&';
Or :                       '|';
True:                      'true'|'True';
False :                    'false'|'False';

Variable : [A-Za-z][a-zA-Z0-9_]*;

Word : ["] (~["\r\n] | '\\\\' | '\\"')* ["];

Comment: ('%%' ~[\r\n]* | '%//' .*? '%//') -> skip;

Whitespace: [ \t\n\r] -> skip;

Integer: [0-9] Digit* | '0';

fragment Digit : [0-9];
```



```
grammar KodoGrammar;

entryPoint : program EOF;

//parser rules

program: statement*;

statement :    assignment
             | variableDeclare
             | display
             | whileblockstatement
             | ifblockstatement
             ;

assignment : Variable Assign expr ;

display :    Display expr;

variableDeclare :    Number  Variable                          #numvariable
                   | Binary Variable                           #binvariable
                   | String Variable                           #strvariable
                   ;

whileblockstatement : While expr  program End;

ifblockstatement :  ifstatement (elseifstatement)* elsestatement End;
elseifstatement :   Elseif expr program;
elsestatement :     Else program;
ifstatement :       If expr program ;

expr :         expr Multiplication expr                 #mulexpr
             | expr Division expr                       #divexpr
             | expr Modulus expr                        #modexpr
             | expr Addition expr                       #addexpr
             | expr Substraction expr                   #subexpr
             | expr GreaterThan expr                    #gtexpr
             | expr LessThan expr                       #ltexpr
             | expr LTEqual expr                        #lteexpr
             | expr GTEqual expr                        #gteexpr
             | expr Equal expr                          #eqexpr
             | expr NEqual expr                         #neqexpr
             | Not expr                                 #notexpr
             | expr And expr                            #andexpr
             | expr Or expr                             #orexpr
             | binary                                   #binaryexpr
             | integer                                  #intexpr
             | string                                   #strexpr
             | Variable                                 #varexpr
             ;

binary   : True | False;
integer  : Integer;
string   : Word;
```

ANTLR Intermediate code generation

- High Level Code:

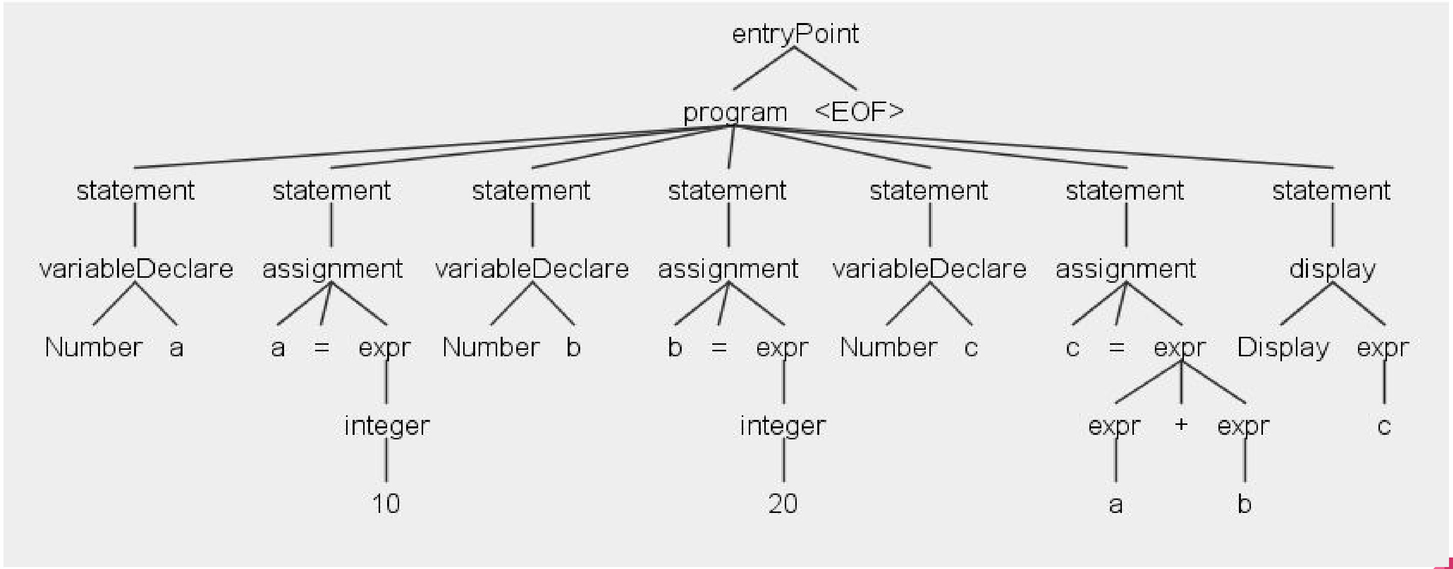


```
Number a
a = 10
Number b
b= 20
Number c
c = a + b
Display c
```

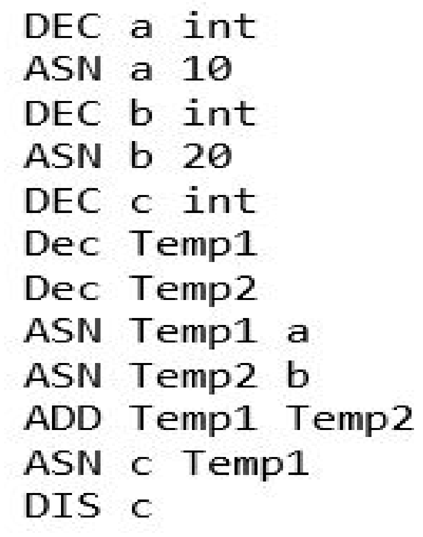
- Parsed output via Antlr:

(entryPoint (program (statement (variableDeclare Number a)) (statement (assignment a = (expr (integer 10)))) (statement (variableDeclare Number b)) (statement (assignment b = (expr (integer 20)))) (statement (variableDeclare Number c)) (statement (assignment c = (expr (expr a) + (expr b)))) (statement (display Display (expr c)))) <EOF>)

Parse Tree generation



Intermediate code generated:



```
DEC a int
ASN a 10
DEC b int
ASN b 20
DEC c int
Dec Temp1
Dec Temp2
ASN Temp1 a
ASN Temp2 b
ADD Temp1 Temp2
ASN c Temp1
DIS c
```

Runtime Overview

- Implemented using Java
- Symbol table - Used hashmap to maintain symbol table.
- Implementation of stack model: Used Stack data structure to execute intermediate code.
- Block structure maintained for if..else and while loop

KoDo sample program implementation

High Level Code

(Kodo_Program.kd)

```
Number a
Number b
a = 10
b = 5
If a > b
Display a
Else
Display b
End
```

Intermediate code

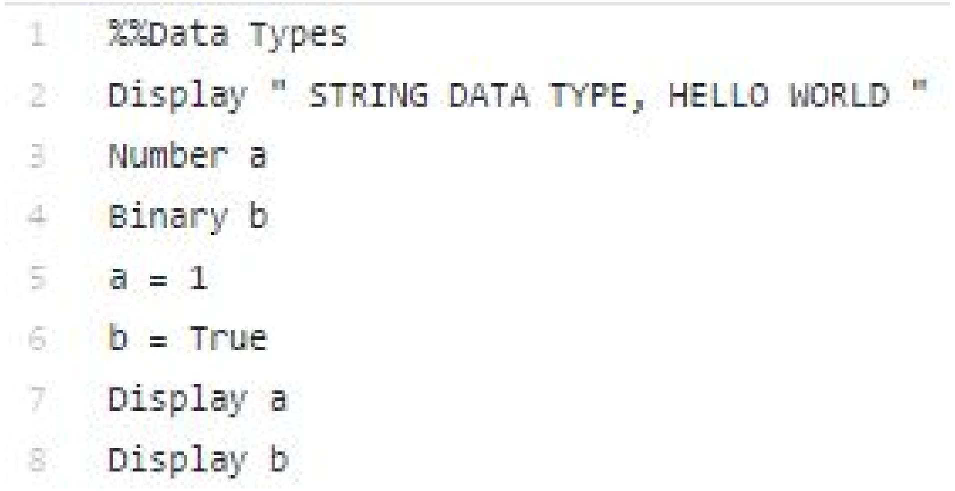
(Kodo_Program.kin)

```
DEC a int
DEC b int
ASN a 10
ASN b 5
Dec Temp1
Dec Temp2
ASN Temp1 a
ASN Temp2 b
GT Temp1 Temp2
If Temp1
DIS a
ENDif
else
DIS b
ENDelse
```

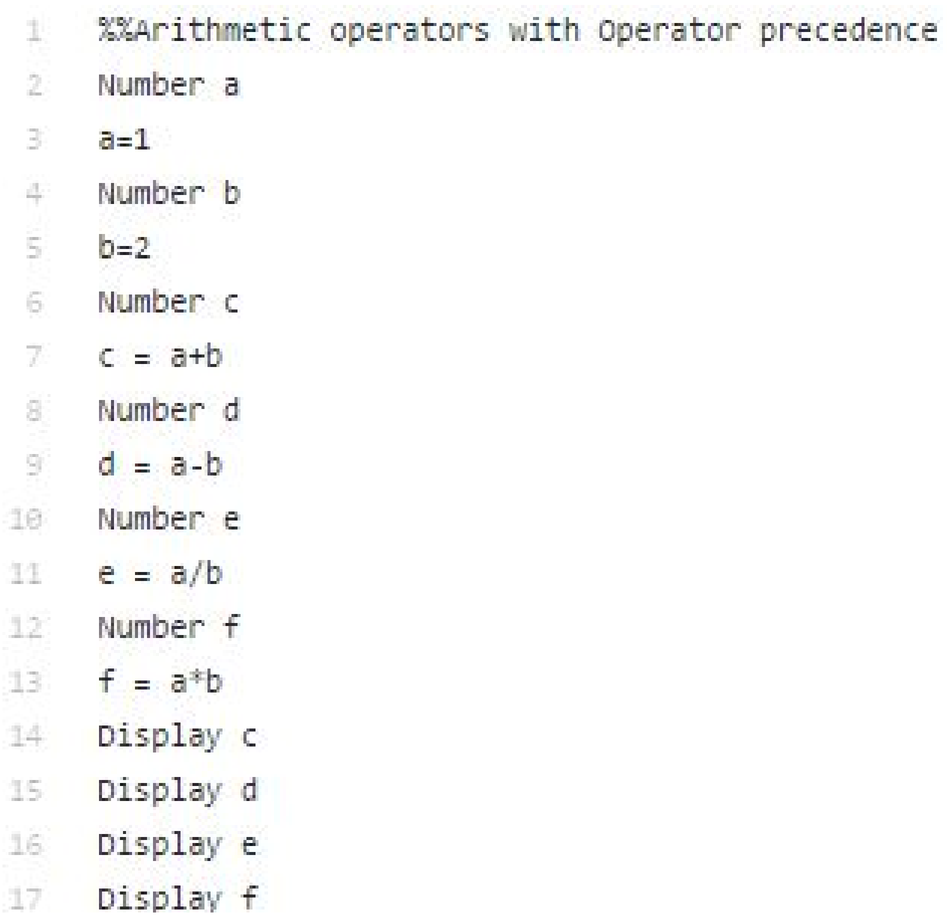
Output:

```
10
```

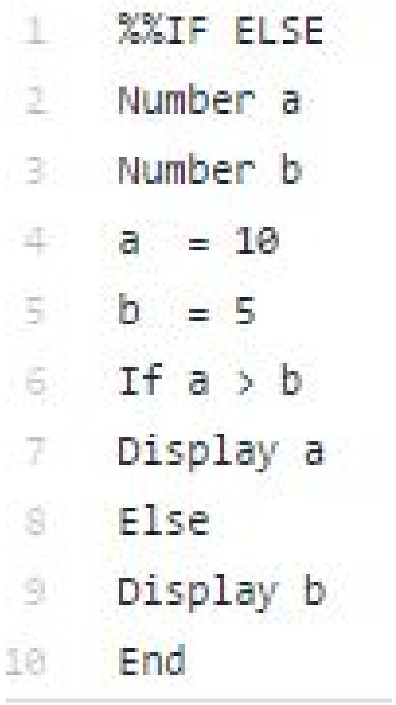
Sample programs



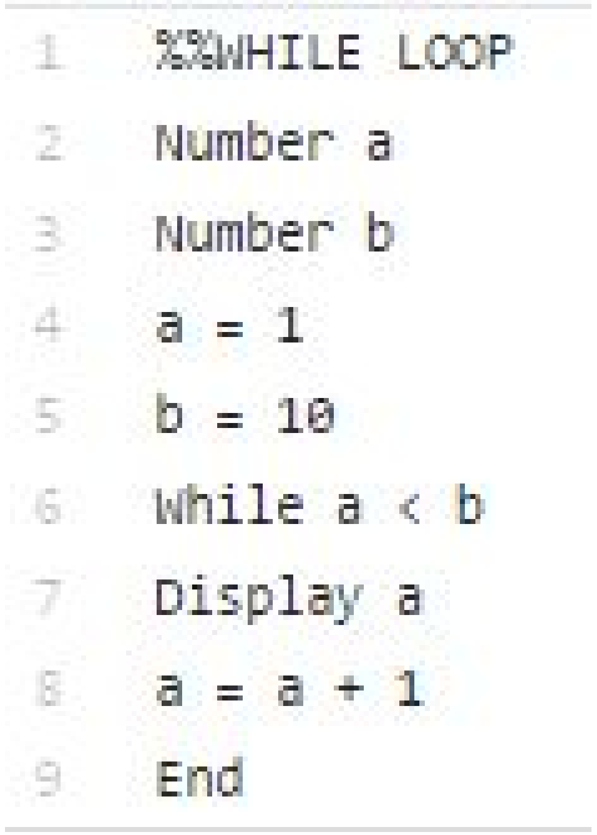
```
%%Data Types
Display " STRING DATA TYPE, HELLO WORLD "
Number a
Binary b
a = 1
b = True
Display a
Display b
```



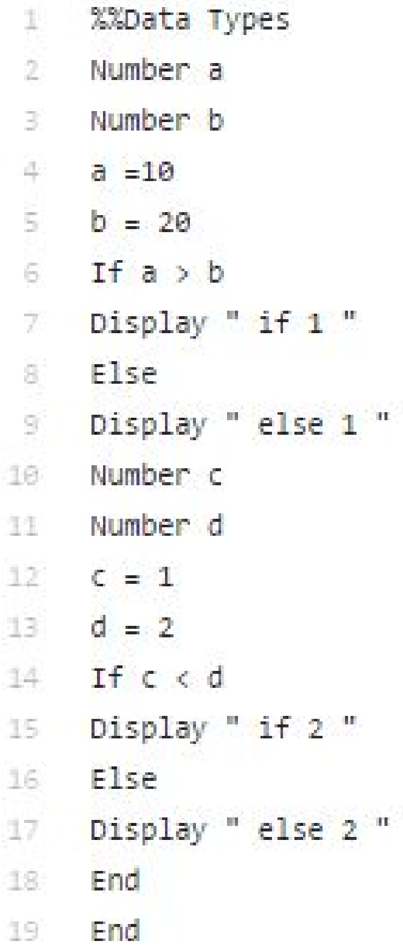
```
%%Arithmetic operators with Operator precedence
Number a
a=1
Number b
b=2
Number c
c = a+b
Number d
d = a-b
Number e
e = a/b
Number f
f = a*b
Display c
Display d
Display e
Display f
```



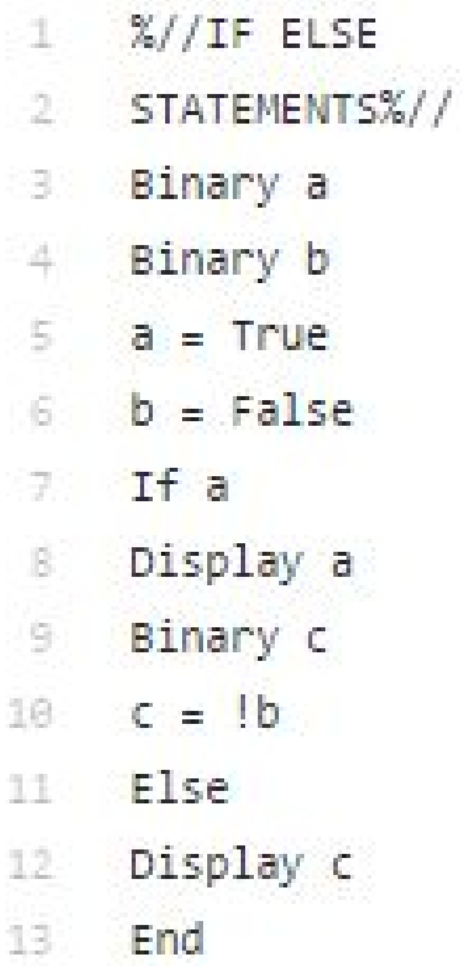
```
%%IF ELSE
Number a
Number b
a  = 10
b  = 5
If a > b
Display a
Else
Display b
End
```



```
%%WHILE LOOP
Number a
Number b
a = 1
b = 10
While a < b
Display a
a = a + 1
End
```

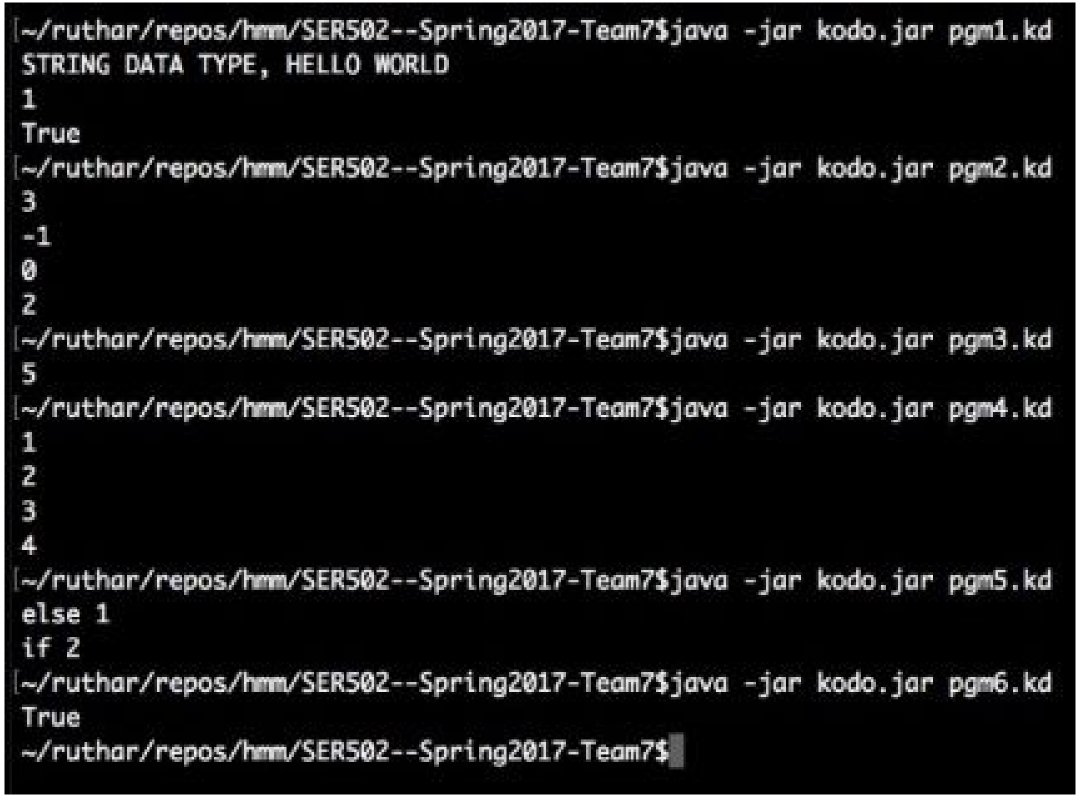


```
%%Data Types
Number a
Number b
a =10
b = 20
If a > b
Display " if 1 "
Else
Display " else 1 "
Number c
Number d
c = 1
d = 2
If c < d
Display " if 2 "
Else
Display " else 2 "
End
End
```



```
%//IF ELSE
STATEMENTS%//
Binary a
Binary b
a = True
b = False
If a
Display a
Binary c
c = !b
Else
Display c
End
```

Output of Sample Programs



```
[~/ruthar/repos/hmm/SER502--Spring2017-Team7$java -jar kodo.jar pgm1.kd
STRING DATA TYPE, HELLO WORLD
1
True
[~/ruthar/repos/hmm/SER502--Spring2017-Team7$java -jar kodo.jar pgm2.kd
3
-1
0
2
[~/ruthar/repos/hmm/SER502--Spring2017-Team7$java -jar kodo.jar pgm3.kd
5
[~/ruthar/repos/hmm/SER502--Spring2017-Team7$java -jar kodo.jar pgm4.kd
1
2
3
4
[~/ruthar/repos/hmm/SER502--Spring2017-Team7$java -jar kodo.jar pgm5.kd
else 1
if 2
[~/ruthar/repos/hmm/SER502--Spring2017-Team7$java -jar kodo.jar pgm6.kd
True
~/ruthar/repos/hmm/SER502--Spring2017-Team7$
```

Thank you !!