SER 502 - Final Project Presentation

Reported by: Team 1

Rishav Kumar (rkumar96)
Shengdong Chen (schen253)
Shivani Patil(spatil46)
Zhuoran Li(zhuoran3)
Zuha Shireen Ansari(zansari1)

MOCHA Overview

This language is designed by taking an inspiration from the Java programming language.

Mocha parses everything in the source file between “begin” and “end” keyword and have simpler print statements compared to JAVA.

Key Features of MOCHA

- Supports primitive data types such as int, float, Boolean and String
- Support for Arithmetic operations such as addition, subtraction, multiplication and division on the data types
- Support for Logical operations such as AND, OR and NOT.
- Supports looping structures like if-else, for, for in range and while constructs.
- Support for Relational operations such as ==, <, > , <=, >=.
- Support for Assignment operator which associates a value with an identifier.
- Support for ternary operator like ? :
- Supports precedence in operators.

SYNTAX

Variable Declaration:

```
int a=5

string z = “hello”

boolean b = true
```

If else syntax:

```
If (a >b)

{

.....

}

else

{

......

}
```

SYNTAX

While and for, for in range

```
while <condition>{
        cond = cond + 1
}
```

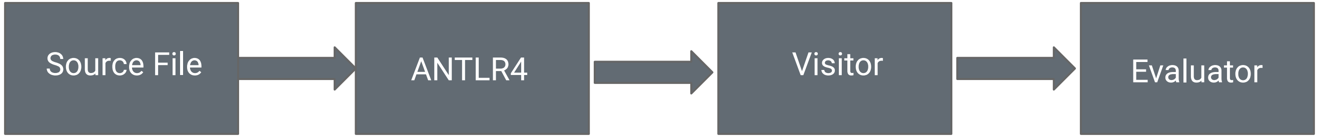
```
for i = 0 to 3

{

……

}
```

```
for i in 0, 3
{

……

 }
```

Print statement

```
print "<any string you like to print>"
```

Flowchart:



Grammar

The basic structure of our grammar looks like this:

```
program : 'begin' body 'end' ;
body : statement* ;
statement
    : variable_declaration
    | assignment_statement
    | if_else_statement
    | for_statement
    | while_statement
    | for_in_range_statement
    | print_statement ;
```

```
/* VARIABLE DECLARATION DEFINITION */
variable_declaration : data_type identifier_list ;
identifier_list : IDENTIFIER (OP_ASSIGN literal)?
(',' identifier_list)? ;

/* ASSIGNMENT STATEMENT DEFINITION */
assignment_statement : IDENTIFIER OP_ASSIGN
expression;

/* EXPRESSION DEFINITION */
expression
   : logical_expression
   | arithmetic_expression
   | relational_expression
   | ternary_expression ;
```

Grammar

```
arithmetic_expression :
       expression_term
       | '(' arithmetic_expression ')'
       | arithmetic_expression(OP_MUL | OP_DIV) arithmetic_expression
       | arithmetic_expression(OP_ADD | OP_SUB) arithmetic_expression
       ;

/* RELATIONAL EXPRESSION DEFINITION */
relational_expression : expression_term (OP_EQUALS | OP_SMALLER |
OP_GREATER | OP_SMALLER_EQUALS | OP_GREATER_EQUALS) expression_term
    | expression_term;

/* LOGICAL EXPRESSION DEFINITION */
logical_expression
    : expression_term (OP_LOGICAL_AND | OP_LOGICAL_OR) expression_term
    | OP_LOGICAL_NOT expression_term
    | expression_term;

/* TERNARY EXPRESSION DEFINITION */
ternary_expression : relational_expression OP_TERNARY_TRUE expression
OP_TERNARY_FALSE expression ;

expression_term : IDENTIFIER | literal | BOOLEAN_FALSE |
BOOLEAN_TRUE;
```

```
if_else_statement : 'if' if_condition '{' body '}' ('else' '{' body
'}')? ;
if_condition: relational_expression | logical_expression ;

for_statement : 'for' for_expression '{' body '}' ;
for_expression : IDENTIFIER OP_ASSIGN INTEGER_LITERAL 'to'
INTEGER_LITERAL ;

while_statement: 'while' while_condition '{' body '}' ;
while_condition: relational_expression | logical_expression ;

for_in_range_statement: 'for' IDENTIFIER 'in' range '{' body '}' ;
range : INTEGER_LITERAL ',' INTEGER_LITERAL ;

print_statement: 'print' print_argument_list ;
print_argument_list
   : literal (',' print_argument_list)?
   | IDENTIFIER (',' print_argument_list)? ;

literal : INTEGER_LITERAL | FLOATING_POINT_LITERAL | BOOLEAN_LITERAL
| STRING_LITERAL ;
```

Grammar

```
/* DATA TYPE DEFINITIONS */
data_type : DATA_TYPE_INT | DATA_TYPE_FLOAT |
DATA_TYPE_BOOLEAN | DATA_TYPE_STRING;
DATA_TYPE_INT        : 'int';
DATA_TYPE_FLOAT      : 'float';
DATA_TYPE_BOOLEAN    : 'boolean';
DATA_TYPE_STRING     : 'string';

/* LITERAL DEFINITION */

INTEGER_LITERAL : [+-]? [0-9] | [+-]? [1-9][0-9]+ ;
FLOATING_POINT_LITERAL : INTEGER_LITERAL '.' [0-9]+ ;
BOOLEAN_LITERAL : BOOLEAN_TRUE | BOOLEAN_FALSE ;
STRING_LITERAL : '"' .*? '"' ;
BOOLEAN_LITERAL | STRING_LITERAL ;
BOOLEAN_TRUE : 'true' ;
BOOLEAN_FALSE : 'false' ;

WHITESPACE : [ \t\r\n]+ -> skip ;
```

```
/* RELATIONAL OPERATOR DEFINITIONS */

OP_SMALLER_EQUALS | OP_GREATER_EQUALS;
OP_EQUALS : '==';
OP_SMALLER : '<';
OP_GREATER : '>';
OP_SMALLER_EQUALS : '<=';
OP_GREATER_EQUALS : '>=';

/* LOGICAL OPERATOR DEFINITIONS */

OP_LOGICAL_AND : '&&';
OP_LOGICAL_OR  : '||';
OP_LOGICAL_NOT : '!';

/* TERNARY OPERATOR DEFINITIONS */

OP_TERNARY_TRUE  : '?' ;
OP_TERNARY_FALSE : ':' ;

/* IDENTIFIER DEFINITION */

IDENTIFIER : [a-zA-Z_][a-zA-Z0-9_]*;
```

Interpreter

ANTLR4

ANTLR is another word for Another Tool For Language Recognition. This takes the grammar as an input and generates an output source code. It is a powerful parser generator for reading, processing, executing, or translating structured text/binary files[1].

An implementation of a tree listener is provided to traverse the abstract syntax tree(AST), which the ANTLR parser provides to us, to generate **recursive descent** parsers for the parsing technique we followed.

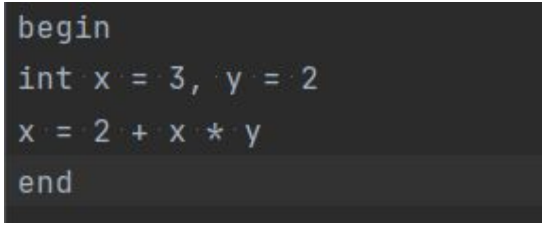
The .g4 files generated by ANTLR are well structured, concise, with good readability. It handles edges cases, as will as supports good error reporting.

It auto-generates classes of lexers, parsers, interpreters, visitor that are well-tested and robust.

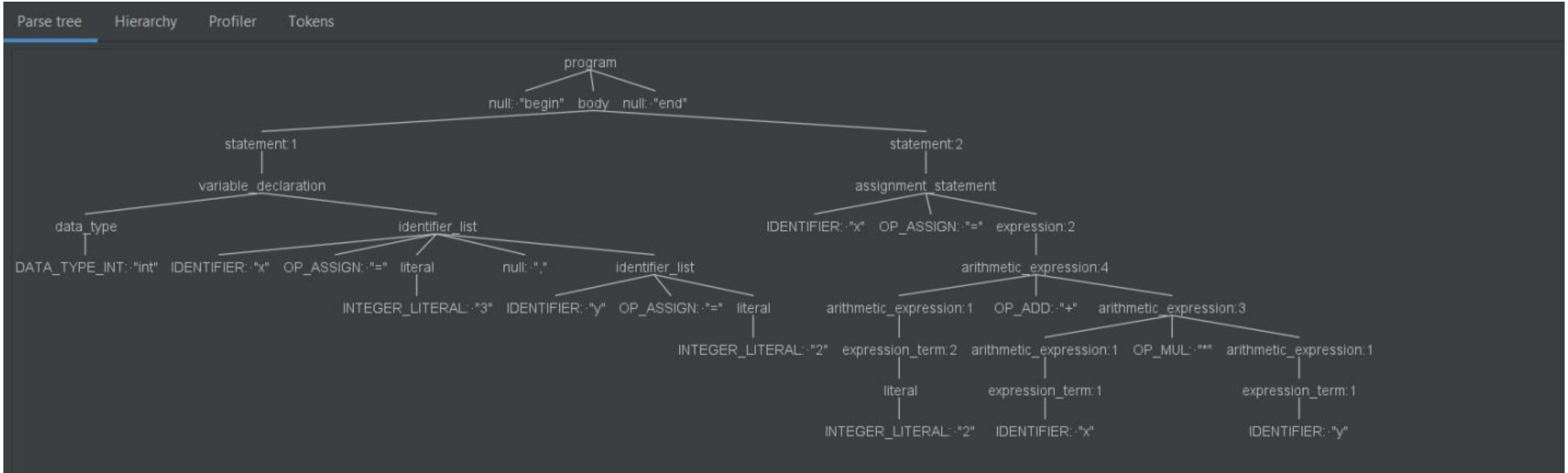
Upon changing the target languages too, ANTLR takes care of portability and there is no need for re-implementing the algorithms.

References: ANTLR

Parse Tree



```
begin
int x = 3, y = 2
x = 2 + x * y
end
```



Programming Language

JAVA

Java is used as it is a semantically safe language and is machine efficient, as well as offers strong type checking mechanism, and automatic garbage collection. All methods written in this language can only be called via a class or an object. Java also offers **generality**, **security, extensibility.**

Java provides support for extending the **visitor base class** generated by **ANTLR4. In our case that is**

```
MyMochaVisitor extends
MochaBaseVisitor<Object>
```

Our evaluator functions are written in **MyMochaVisitor.java** class

Data Structures

- **List**

```
private final List<SemanticError>
semanticErrorList;
```

(This captures all the semantic errors that arises upon execution of the code.)

- **HashMap**

```
private final Map<String, Variable>
variableMap;
```

(This stores the identifier and variable in the form of key-value pairs)

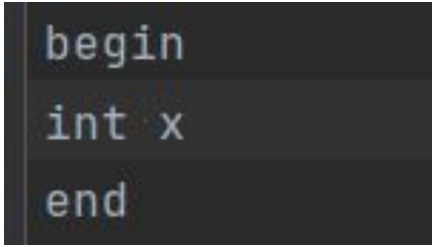
Runtime Evaluator

The evaluation is done in the **MyMochaVisitor.java** file

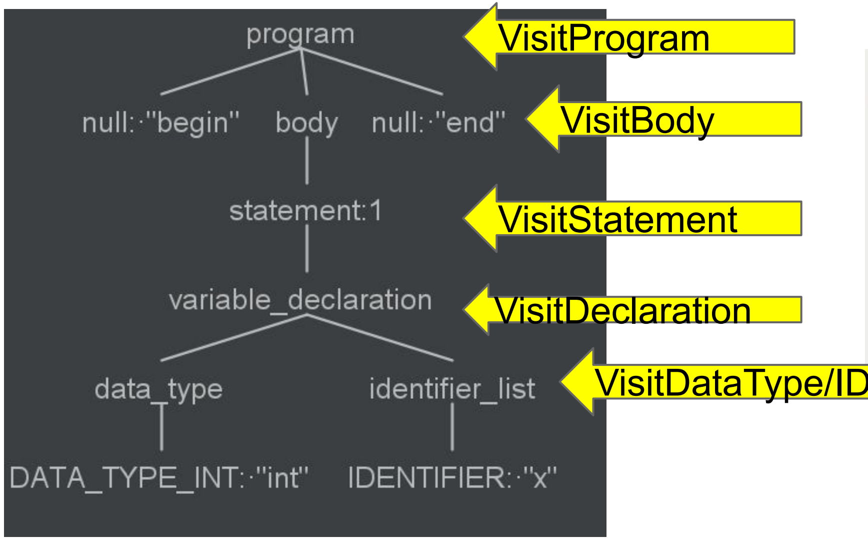
The functions implemented for semantic evaluation are as shown below:

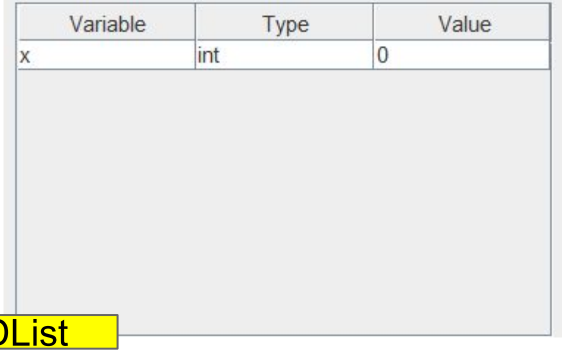
```
@Override public Object
visitIdentifier_list(MochaParser.Identifier_listContext ctx)
@Override public Object
visitAssignment_statement(MochaParser.Assignment_statementContext ctx)
@Override public Object
visitArithmetic_expression(MochaParser.Arithmetic_expressionContext ctx)
@Override public Object
visitRelational_expression(MochaParser.Relational_expressionContext ctx)
@Override public Object
visitLogical_expression(MochaParser.Logical_expressionContext ctx)
@Override public Object
visitTernary_expression(MochaParser.Ternary_expressionContext ctx)
@Override public Object
visitExpression_term(MochaParser.Expression_termContext ctx)
@Override public Object
visitIf_else_statement(MochaParser.If_else_statementContext ctx)
@Override public Object
visitIf_condition(MochaParser.If_conditionContext ctx)
@Override public Object
visitFor_statement(MochaParser.For_statementContext ctx)
@Override public Object
visitWhile_statement(MochaParser.While_statementContext ctx)
@Override public Object
visitWhile_condition(MochaParser.While_conditionContext ctx)
@Override public Object
visitFor_in_range_statement(MochaParser.For_in_range_statementContext ctx)
@Override public Object
visitRange(MochaParser.RangeContext ctx)
@Override public Object
visitPrint_argument_list(MochaParser.Print_argument_listContext ctx)
@Override public Object
visitLiteral(MochaParser.LiteralContext ctx)
public void printEvaluationResults()
public Map<String, Variable> getVariableMap()
```

Run Time Flow



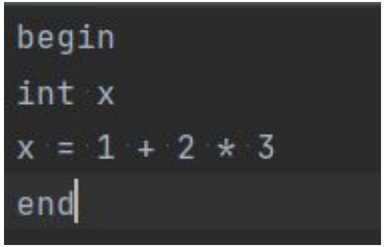
```
begin
int x
end
```



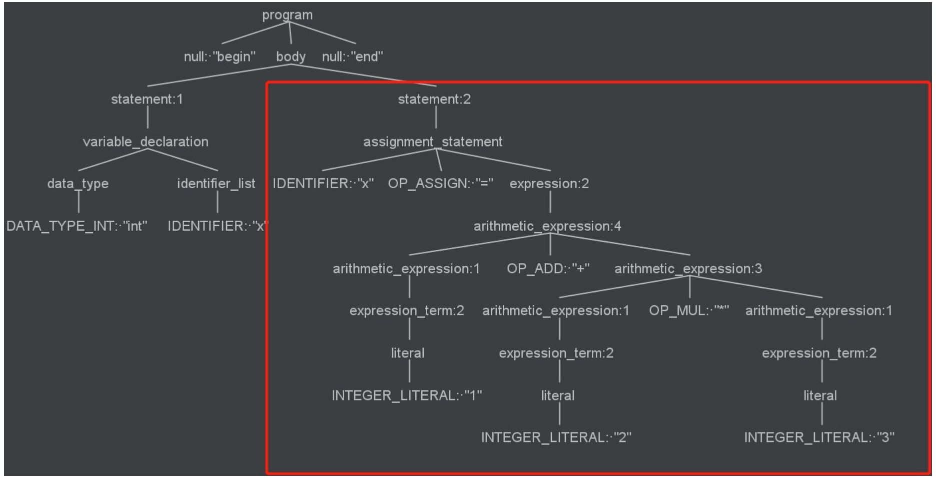


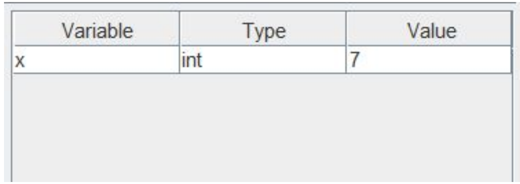
Variable	Type	Value
x	int	0

Run Time Flow (Continue)



```
begin
int x
x = 1 + 2 * 3
end
```





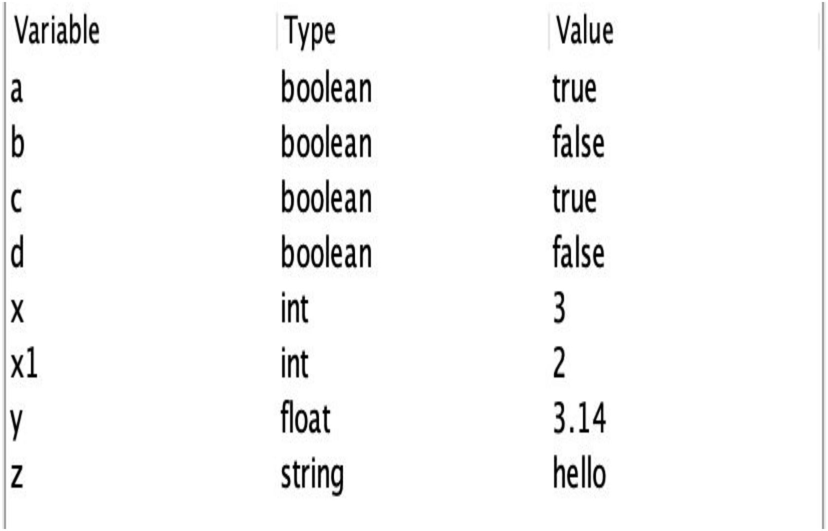
Variable	Type	Value
x	int	7

Sample Test Cases

TEST CASE 1:

```
begin
     int x = 3, x1 = 2
     float y = 3.14
     string z = "hello"
     boolean a = true
     boolean b
     b = true && false
     boolean c
     c = true || false
     boolean d
     d = !true
end
```

Output



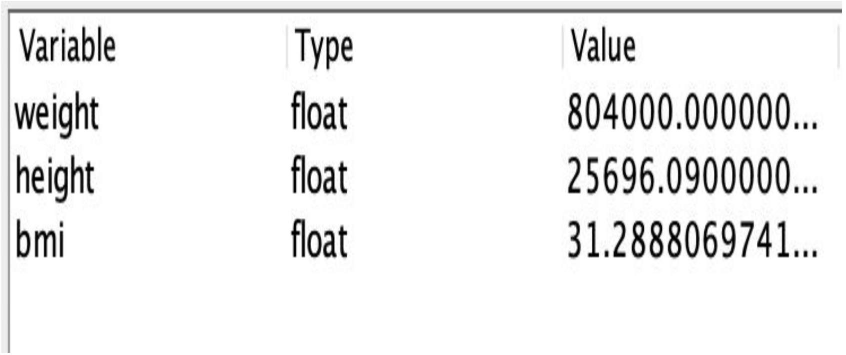
Variable	Type	Value
a	boolean	true
b	boolean	false
c	boolean	true
d	boolean	false
x	int	3
x1	int	2
y	float	3.14
z	string	hello

Sample Test Cases

TEST CASE 2:

```
begin
  float weight = 80.4, height = 160.3
  float bmi
  weight = weight * 100 * 100
  height = height * height
  bmi = weight / height
  if bmi > 30 {
     print "You are overweight"
  } else {
     print "You are normal"
  }
end
```

Output



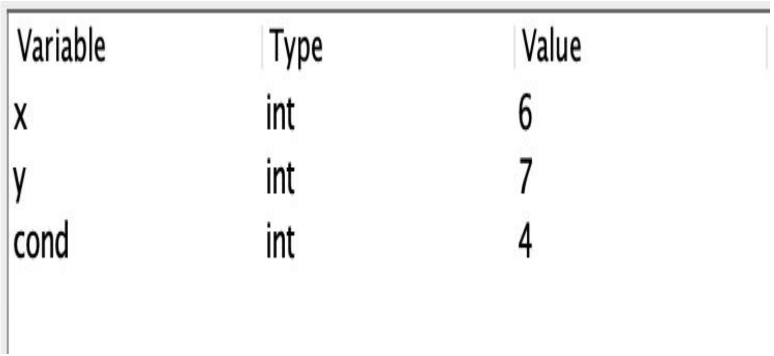
Variable	Type	Value
weight	float	804000.000000...
height	float	25696.0900000...
bmi	float	31.288069741...

Sample Test Cases

TEST CASE 3:

```
begin
  int x = 3, y = 4
  for i = 0 to 3 { x = x + i }

  for i in 0, 3 { y = y + i }

  int cond = 0
  while cond <= 3 {
     cond = cond + 1
  }
end
```

Output

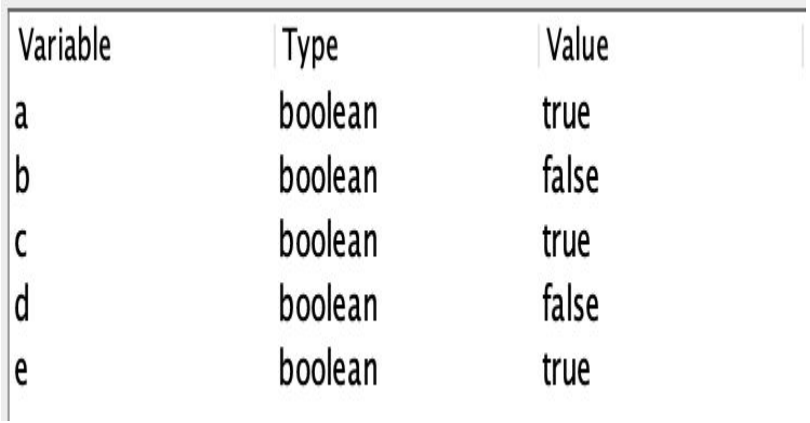


Variable	Type	Value
x	int	6
y	int	7
cond	int	4

Sample Test Cases

```
TEST CASE 4:

begin
  boolean a, b, c, d, e
  a = 1 == 1
  b = 1 >= 2
  c = 1 <= 2
  d = 1 > 2
  e = 1 < 2
  print "a -> ", a, " b -> ", b, " c
-> ", c, " d -> ", d, " e -> ", e
end
```

Output



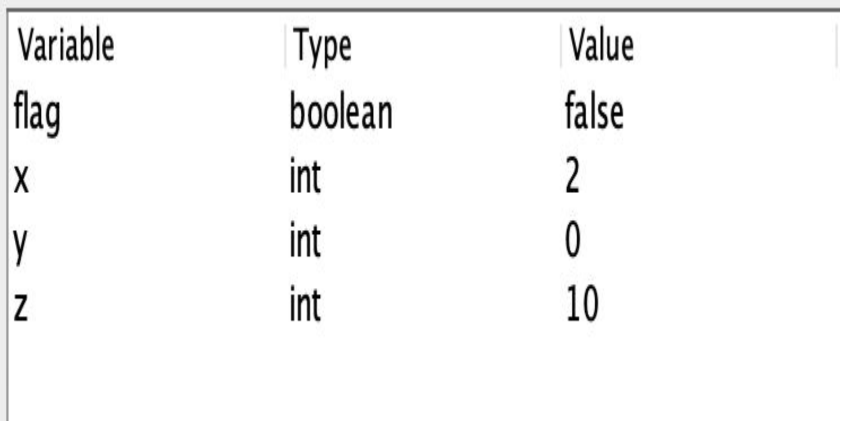
Variable	Type	Value
a	boolean	true
b	boolean	false
c	boolean	true
d	boolean	false
e	boolean	true

Sample Test Cases

TEST CASE 5:

```
begin
  int x, y, z
  boolean flag = false;
  x = flag ? 1 : 2
  if flag { y = 10 } else { z = 10 }
end
```

Output



Variable	Type	Value
flag	boolean	false
x	int	2
y	int	0
z	int	10

Thank-You :)

—Team 1-Mocha