Perx

v1.0.0

SER – 502 Languages and Programming Paradigms

Team 33

Vihar Bhatt
Jay Patel
Meet Patel
Raghavendra

Overview

- Language Introduction
- Grammar, Lexical Analyzer and Parser
- Intermediate Code
- Runtime
- Demonstration of the language & sample programs.

Language Introduction

- Perx Programming Language
- Easy to code
- Mainly based on C
- Runtime written in Java
- Used prefix notation for Intermediate code
- Used stack machine mode for virtual machine

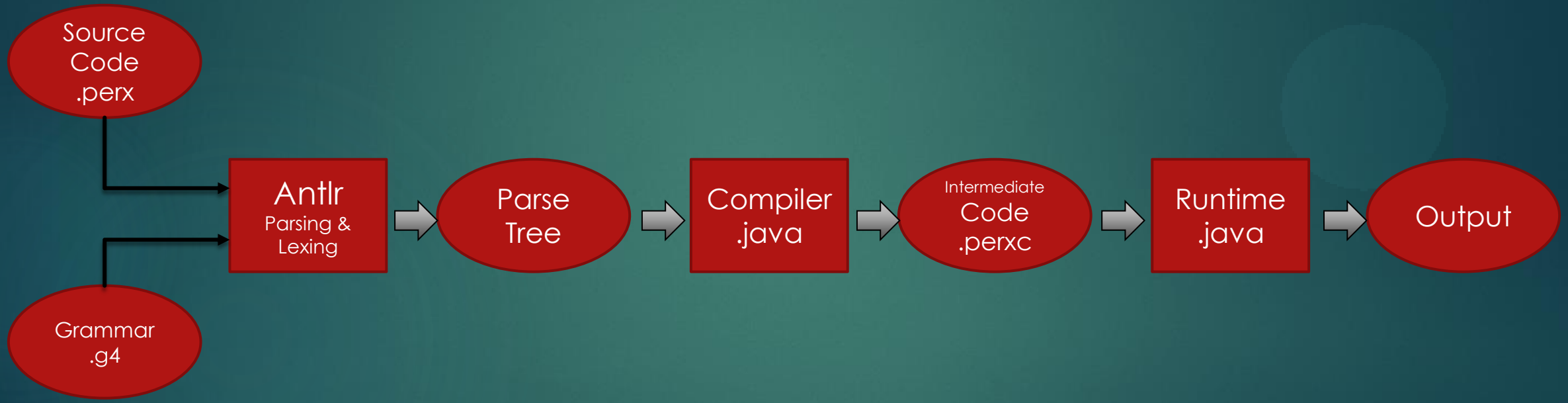
Language Features

- Simple Syntax
- Data types – boolean , int
- Arithmetic operations – addition, subtraction, division, multiplication, modulo.
- Boolean operations
- Relational operations – !=, ==, <,>,<=,>=
- Operator Precedence
- Decision statements – if – else
- Loop structures – while loop

Tools used to develop Perx

- Eclipse Oxygen
- IntelliJ Idea
- Antlr v4.7.1 – Takes the grammar as input, performs lexical analysis, generates tokens, parses the grammar and generates the parse tree.
- Antlr generates a listener class which helps us traverse all the nodes of the parse tree.

Language Design Flow



Grammar

Our Basic Grammar looks like

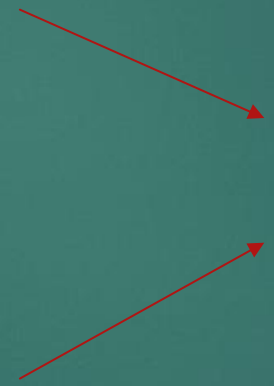
```
grammar Perx;

program: stmt_block
       ;

stmt_block: stmt';'stmt_block
          | stmt';'
          ;

stmt: decl_stmt
    | assign_stmt
    | ifelse_stmt
    | whileloop_stmt
    | print_stmt
    ;

decl_stmt: integer Identifier
         | bool Identifier
         ;
```

Basic Syntax and Equivalent Grammar Rule

Variable Declaration

integer a;

boolean a;



```
decl_stmt: integer Identifier
         | bool Identifier
         ;
Identifier: [a-z]+
          ;
integer: 'integer'
       ;
bool: 'boolean';
```

Assignment Statement

```
a = 5;
b = T;
```



```
assign_stmt: Identifier ASSIGN
expression
;
Identifier: [a-z]+
;
expression: expression DIV
expression
| expression MUL expression
| expression MOD expression
| expression ADD expression
| expression SUB expression
| '('expression')'
| Identifier
| Number
;
ASSIGN:'=';
```

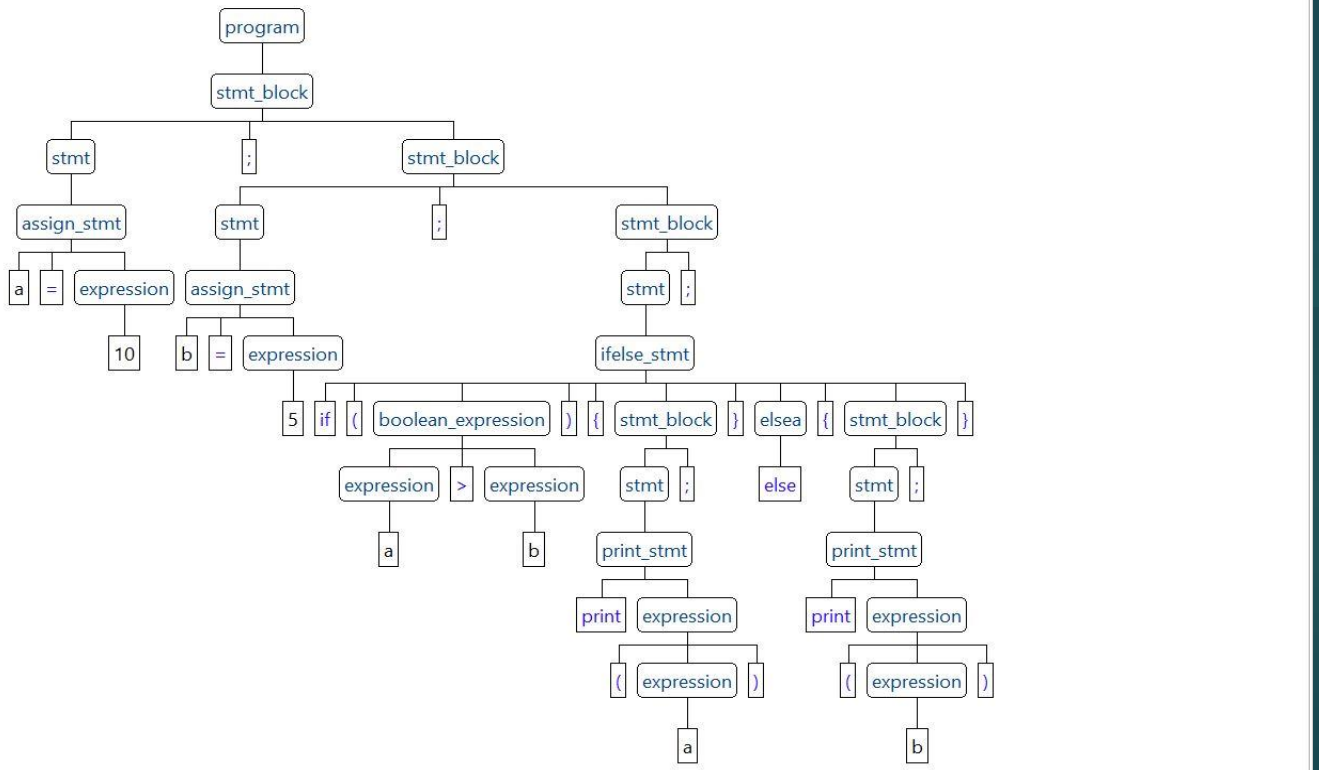
Parse Tree

- Antlr generates Parse tree
- Parse Tree helps in verifying the grammar.
- Walker class traverses the nodes of the parse tree to generate the Intermediate Code

Parse Tree for a Sample if-else program

Hello::program

```
a=10;
b=5;
if(a>b)
{
print(a);
}
else
{
print(b);
};
```



Intermediate Code Generation

- Antlr generates two listener classes –

1) BaseListener.java
2) Listener.java

- This classes help us to traverse nodes of the parse tree.
- Enter & Exit methods for each node.
- Fires action and generates the intermediate code.

Operations of Intermediate Code

- START
- END
- ASSIGN
- PRINT
- IF
- ELSE
- ADD
- SUM
- DIVIDE
- MULTIPLY
- MODULO
- WHILE
- INT
- BOOLEAN

Sample Intermediate Code

```
a = 5;
b = 6;
c = a+b;
print c;
```



```
ASSIGN a 5
ASSIGN b 6
ASSIGN c SUM a,b
PRINT c
```

```
a = 10;
b = 11;
if (a>b){
print a;
    }
 else {
print b;
    };
```



```
ASSIGN a 10
ASSIGN b 11
IF GREATER a,b
T START
PRINT a END
F START
PRINT b END
```

Runtime

- Written in Java
- Data structures

1)Hash map

2)Linked List

3)Stack

- Intermediate code is fed into the runtime to generate the output.

Intermediate  Runtime  Output

Sample Runtime execution and Output

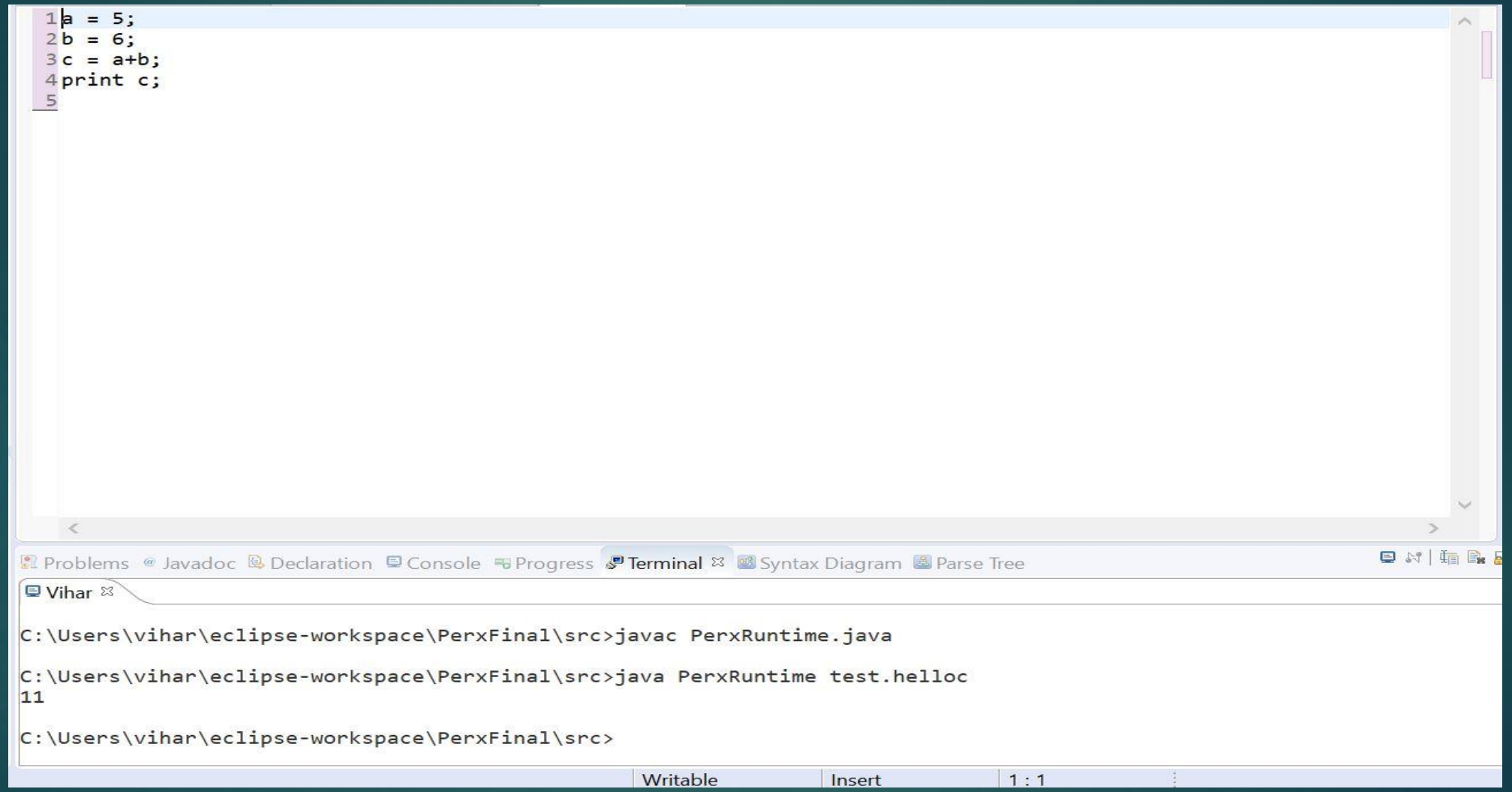
```
1 a = 10;
2 b = 11;
3 if (a>b){
4 print a;
5 }
6 else {
7 print b;
8 };
```

Problems Javadoc Declaration Console Progress Terminal Syntax Diagram Parse Tree

Vihar

```
C:\Users\vihar\eclipse-workspace\PerxFinal\src>javac PerxRuntime.java

C:\Users\vihar\eclipse-workspace\PerxFinal\src>java PerxRuntime test.helloc
11

C:\Users\vihar\eclipse-workspace\PerxFinal\src>
```

Writable Insert 1 : 1


a = 5;
b = 6;
c = a+b;
print c;
Problems
Javadoc
Declaration
Console
Progress
Terminal
Syntax Diagram
Parse Tree
Vihar
C:\Users\vihar\eclipse-workspace\PerxFinal\src>javac PerxRuntime.java
C:\Users\vihar\eclipse-workspace\PerxFinal\src>java PerxRuntime test.helloc
11
C:\Users\vihar\eclipse-workspace\PerxFinal\src>
Writable
Insert
1 : 1

Thank You