SOPA LANGUAGE

SER 502 : LANGUAGES AND PROGRAMMING PARADIGMS

DR. AJAY BANSAL

HTTPS://GITHUB.COM/SHAHSHAILEE/SER502-SOPA-TEAM5.GIT

HTTPS://YOUTU.BE/6YLUF7SXN50

TEAM - 5

MEMBERS

Vaishnavi Bhalodi
Shailee Shah
Satyam Shekhar
Aniket Patil

WHY SOPA?

“WHY DOES PROGRAMMING HAVE TO BE SO COMPLEX ?”

SOPA was designed with simplicity and ease of use in mind. It is specifically tailored for non-programmers while retaining essential programming constructs.

- **Simple Syntax**: Easy to read and write.
- **Intuitive Structure**: Programs flow naturally.
- **Accessible**: Ideal for beginners and professionals alike.

HIGHLIGHTS OF SOPA

- Readability: Human-friendly syntax with minimal boilerplate.
- Core Constructs: Includes essential programming structures:
  - Variable declarations and assignments.
  - Conditional blocks (if-else, ternary operators).
  - Loops (for, while).
  - Basic mathematical and logical operations.
- Ease of Learning: Designed for rapid understanding and use.

GRAMMAR

```
program ::= block
block ::= start decl; command stop
decl ::= decl ; decl | const identifier = assignments_allowed | var identifier
command ::= command ; command | identifier := expression | if statement_list then command else command endif | repeat
repeat_statement command endrepeat | unless statement_list command until | print_statement | condition ? command : command | block
assignments_allowed ::= number | '"' (.)*? '"'
statement_list ::= '(' condition ')'
repeat_statement ::= '(' identifier ',' number ',' number ')'
print_statement ::= show expression | show '"' (.)*? '"'
condition ::= expression operators expression | boolen
expression ::= expression + expression | expression - expression | expression * expression | expression / expression | expression %
expression | expression ^ number | sqrt expression | (expression) | identifier := expression | identifier | number
boolen ::= true | false | expression = expression | not boolean
operators ::= '==' | '<' | '>' | '<=' | '>=' | '!=' | '&&' | '||' | '!'
identifier ::= lowerCase (letters | _ | digit)*
letters ::= (lowerCase | upperCase)+
lowerCase ::= 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' | 'v' | 'w' | 'x' | 'y' | 'z'
upperCase ::= 'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' | 'H' | 'I' | 'J' | 'K' | 'L' | 'M' | 'N' | 'O' | 'P' | 'Q' | 'R' | 'S' | 'T' | 'U' | 'V' | 'W' | 'X' | 'Y' | 'Z'
digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
number ::= '-' (digit)+ | (digit)+
```

KEY FEATURES

01

BASIC DATA TYPES

- Integer, Boolean, and String.
- Declaration and assignment.

02

OPERATIONS

- Arithmetic: Addition, Subtraction, Multiplication, Division, Modulus.
- Relational: >, <, >=, <=, ==, !=.
- Logical: and, or, not.

03

CONTROL STRUCTURES

- Conditional Blocks: If-else and ternary.
- Loops: repeat (for loop) and unless (while loop).

ADDITIONAL FUNCTIONALITIES

- MODULUS -

```
start
var a;
var b;
a := 8;
b := 2;
show a%b;
stop
```

- SQRT -

```
start
var b;
b := 4;
show sqrt(b);
stop
```

SYNTAX OVERVIEW

Program Structure

- All programs must start with start and end with stop.

Variable names:

- Must start with an alphabet.
- Can include underscores and digits.

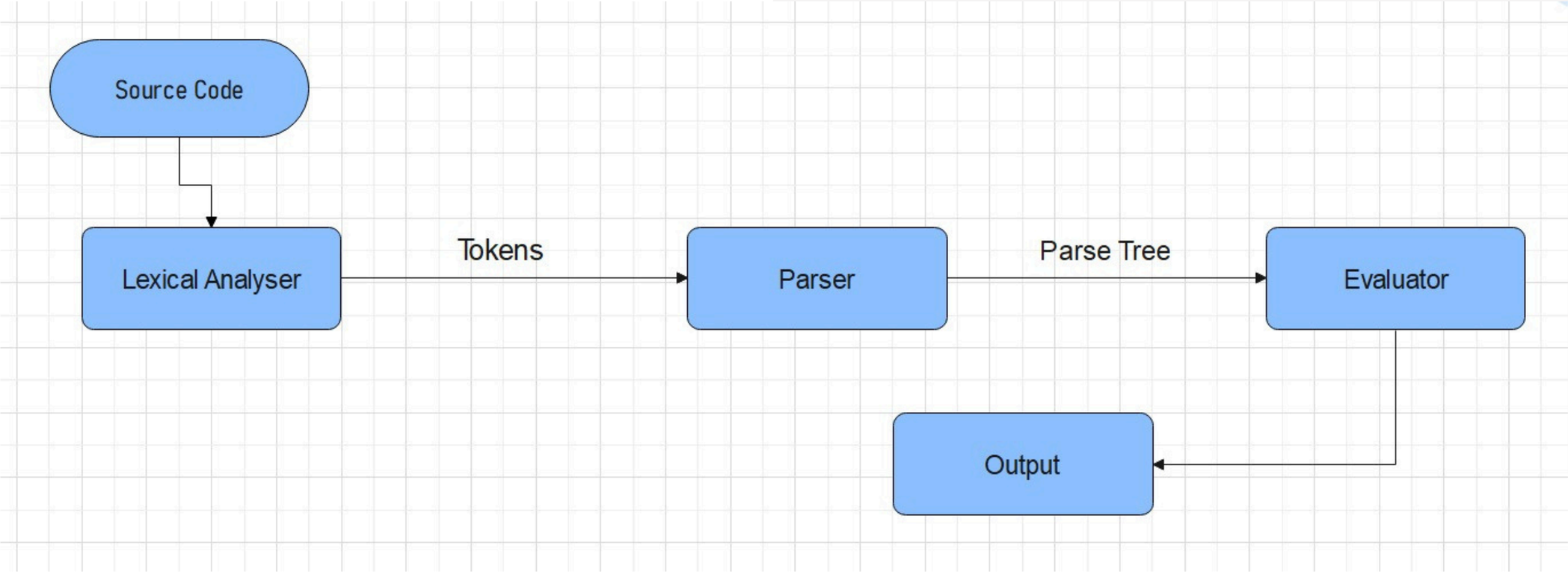
```
x := condition ? true_value : false_value;
```

```
var x;
const y = 10;
const z = "hello";
```

```
x := 5;
y := "world";
z := true;
```

```
repeat (i, 1, 10)
  show i;
endrepeat;
```

Compilation flow



COMPILATION FLOW

1. Lexical Analyzer

Purpose: Scans the SOPA source code and breaks it into tokens.

Output: A series of tokens representing keywords, identifiers, operators, etc.

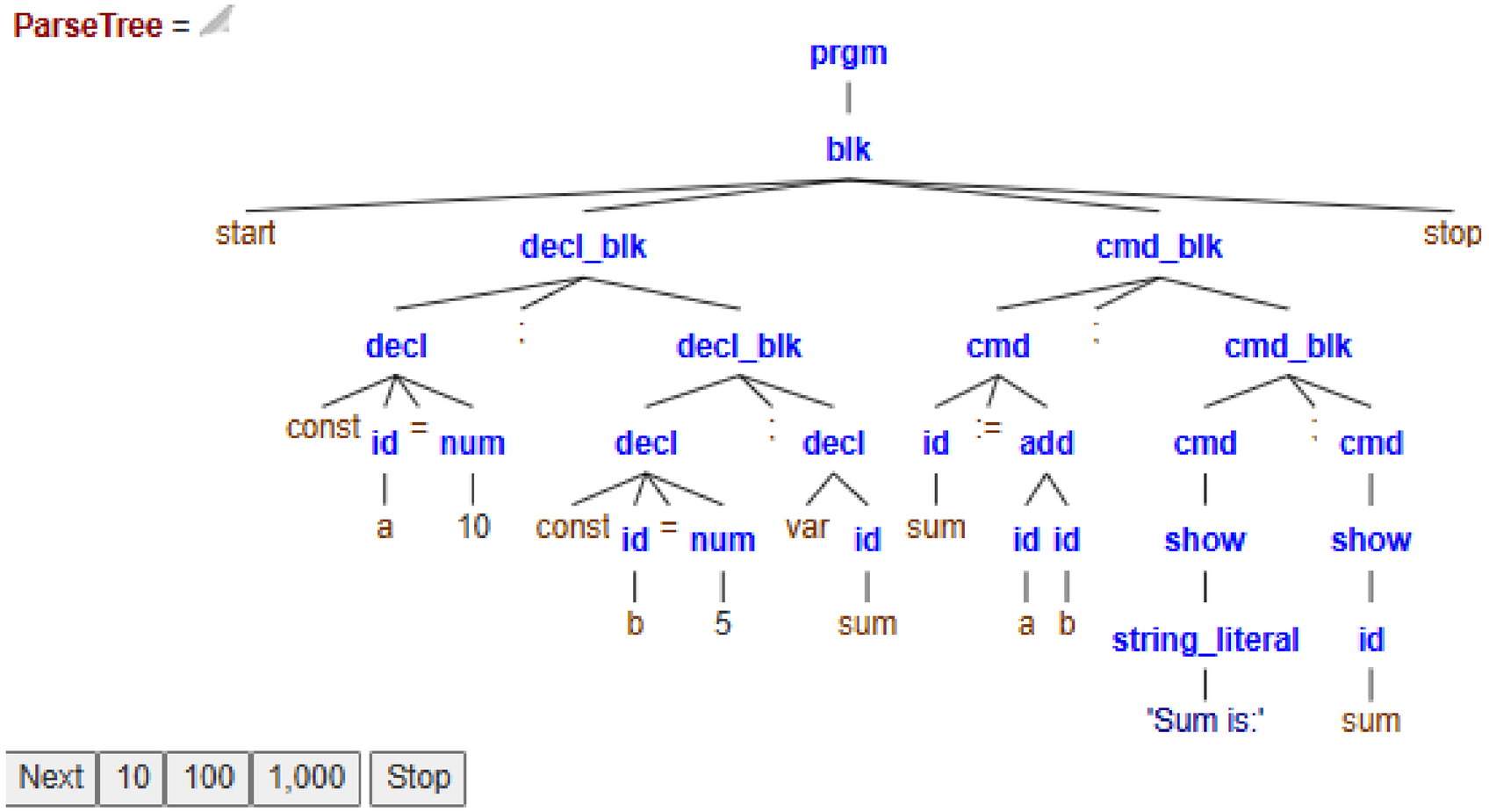
```
?- InputString = 'start
     const a = 10;
     const b = 5;
     var sum;
     sum := a + b;
     show "Sum is:";
     show sum;
   stop',
   tokenize(InputString, Tokens).
```

Tokens =
[*kw*(start), *kw*(const), *id*(a), =, *num*(10), ;, *kw*(const), *id*(b), =, *num*(5), ;, *kw*(var), *id*(sum), ;, *id*(sum), :=, *id*(a), +, *id*(b), ;, *kw*(show), *string*('Sum is:'), ;, *kw*(show), *id*(sum), ;, *kw*(stop)]

2. Parser

Purpose: Analyzes the sequence of tokens to ensure the syntax is correct.

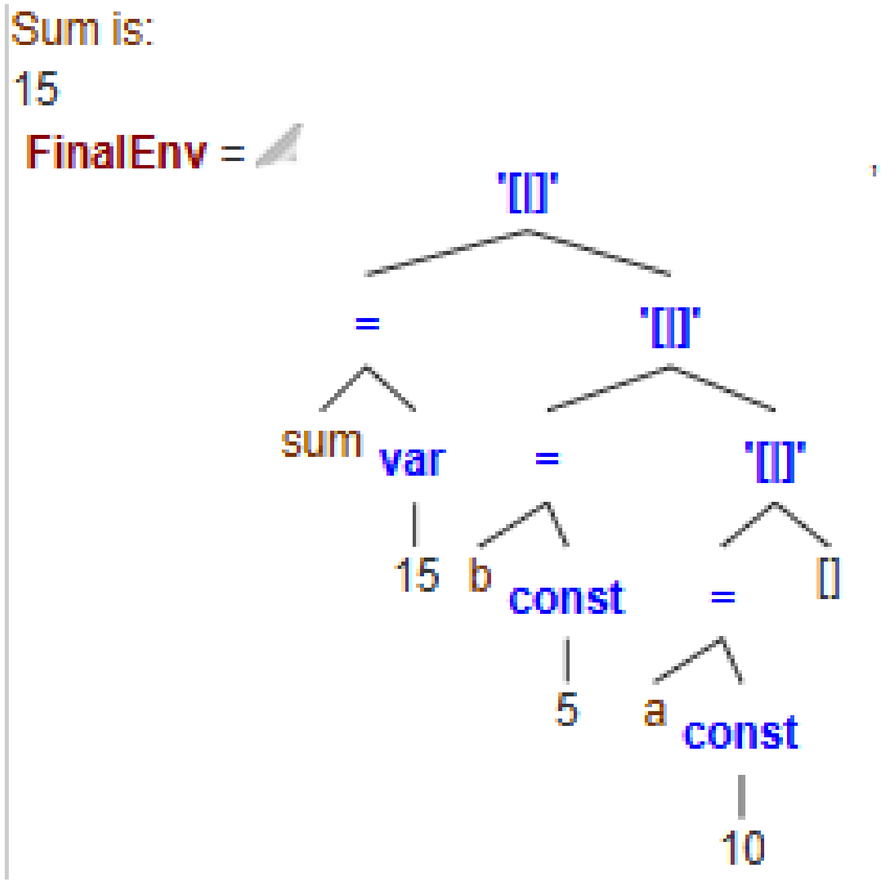
Output: A parse tree representing the program's structure.



3. Semantics

Purpose: Defines the logical meaning of each construct in the language.

Output: Ensures program behavior matches expected outcomes.



SAMPLE CODE

ADDITION

```
start
const x=8;
var y;
y:= x+2;
show y;
stop
```

NESTED FOR LOOP

```
start
repeat(i,2,10)
start
repeat(j,1,3)
show j;
endrepeat
stop
endrepeat
stop
```

SAMPLE CODE

WHILE LOOP

```
start
var a;
a := 1;
unless(a<10)
show "You can do this !!!" ;
a := a + 1;
until
stop
```

SQUARE ROOT

```
start
var b;
b := 4;
show sqrt(b);
stop
```

IF - ELSE

```
start
var a;
var b;
a := 10;
b := 20;
if(a>20)
show "a is greater";
else
show " b is greater";
endif
stop
```

FUTURE SCOPE

- Adding support for:
- Lists and Arrays: Enhance data management capabilities.
- Advanced Data Structures: Incorporate trees, queues, and stacks for more complex programs.
- String Manipulation: Extend operations to allow robust handling of textual data.

THANK YOU