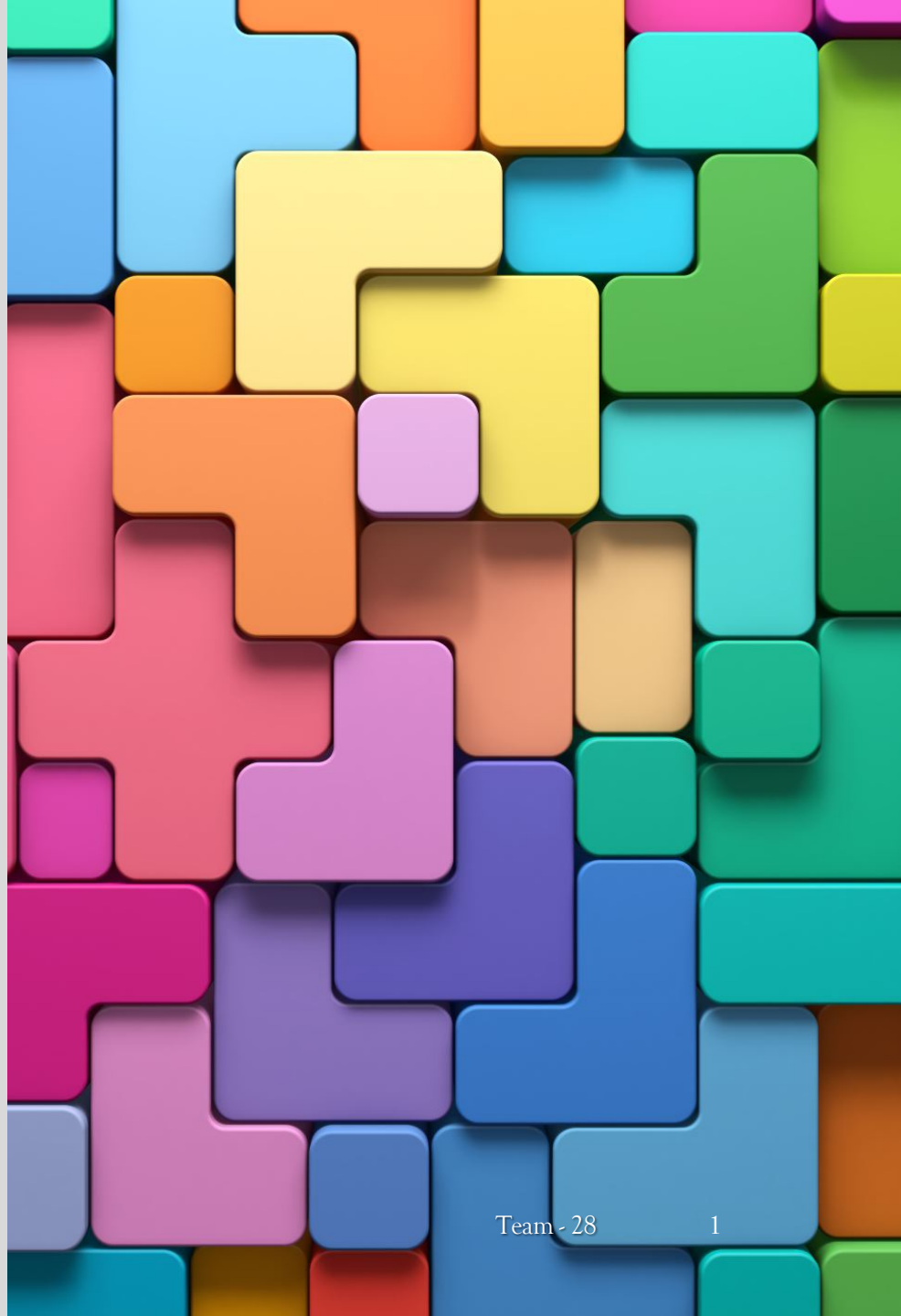
KVS LANGUAGE

SER 502 – Project – Team 28 – Spring 2023



SUMMARY

- Introduction
- Structure of the Plan
- Tools Used and Implemented
- KVS Grammar
- KVS Features
- Steps Involved in the Execution
- Execution Output

INTRODUCTION

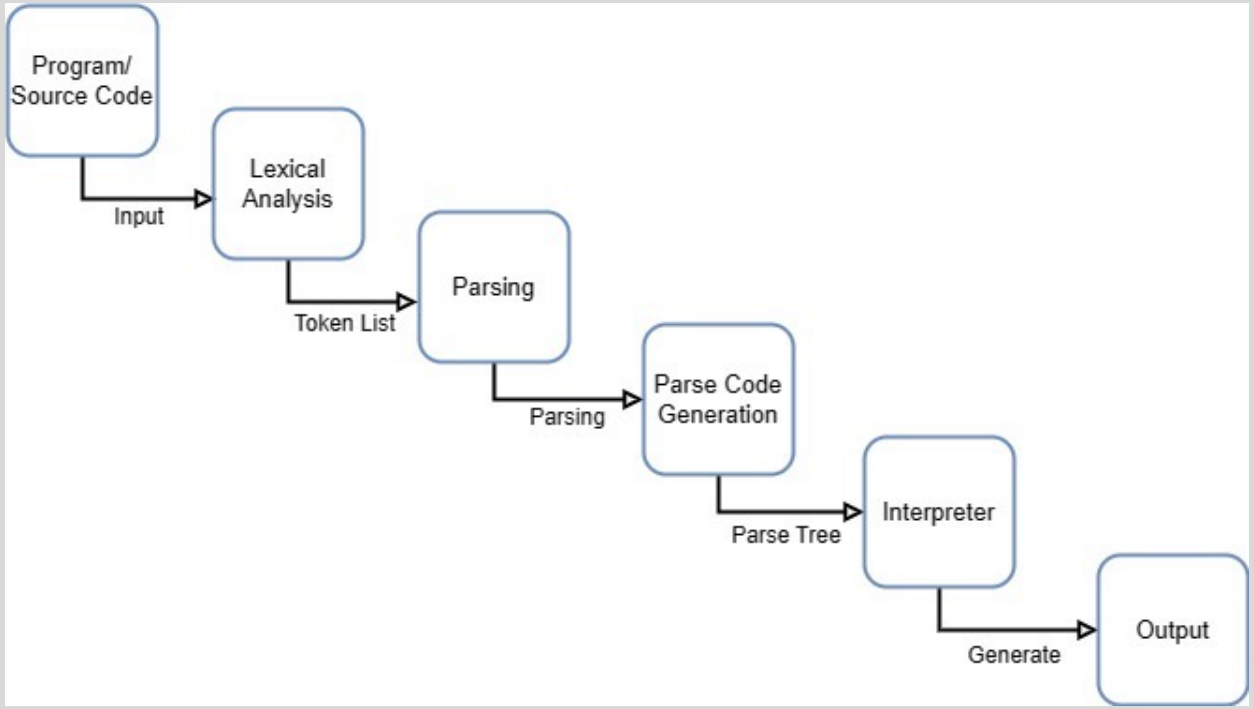
KVS Language, named after the Initials of the team members Kavya, Veda, Satvik, and Sunil, is a simple programming language with all the necessary functionalities created using Prolog and Python. Token generation was performed using Prolog.

Characteristics:

- Input file extension '.kvs'.
- Lexer.py is used to generate a list of tokens. These tokens are then parsed using Prolog to give the outcome.

STRUCTURE OF THE PLAN

- The KVS language has a straightforward, modular structure.
- The initial program we create is scanned by the lexical analyzer, producing tokens as the parser needs to produce intermediate code.
- After evaluating the program, the interpreter checks the code to see if it is semantically accurate before producing the output.



TOOLS USED

- SWI Prolog (Compilation)
- Python 3.9 (Tokens)
- SWI Prolog (Parser)
- SWI Prolog (Interpreter)

KVS GRAMMAR -- FUNDAMENTAL DEFINITIONS

```
NUM: = /^[0-9]+$/
BOOL: = / 'True' | 'False'/
STR: = /\" [\x00-\x7F]*\"/
CHAR: = /'[\x00-\x7F]'/
DATATYPE: = 'int' | 'bool' | 'string'
IDENTIFIER: = /^[a-z, A-Z_$][a-zA-Z_$0-9]*$/
ADDITION: = '+'
SUBTRACTION: = '-'
MULTIPLICATION: = '*'
DIVISION: = '/'
AND OPERATOR: = 'and'
OR OPERATOR: = 'or'
NOT OPERATOR: = 'not'
ASSIGNING VALUE: = '='
COMPARING VALUES: = '>' | '<' | '<=' | '>=' | '==' | '!='
CONDITIONAL OPERATORS: = and | or | not
IF-BLOCK: = 'if'
ELSE-BLOCK: = 'else'
```

KVS GRAMMAR -- FUNDAMENTAL DEFINITIONS

```
ELSEIF-BLOCK: = 'elseif'
WHILE-BLOCK: = 'while'
FOR-BLOCK: = 'for'
IN-LOOP: = 'in'
RANGE-CONDITION: = 'range'
COMMENTING: = '#'
START BLOCK: = '{'
END BLOCK: = '}'
DLR: = ';'
COMMA: = ','
STARTING QUOTE: = '''
ENDING QUOTE: = '''
BEGIN: = 'start'
END: = 'terminate'
OPAR: = '('
CPAR: = ')'
DISPLAY:= 'disp'
```

KVS GRAMMAR – PROGRAM RULES

```
program: = START statements END with TERMINATE | comment statements with START s
statement: = all statements DLR statements | statements
every statement: = disp | declaration | assign | if-else | while | for
declaration: = DATATYPE SPACE IDENTIFIER ASSIGN data | DATATYPE SPACE IDENTIFIER
assign: = IDENTIFIER ASSIGN expression
print: = DISP SPACE STARTING QUOTE STRING ENDING QUOTE | DISP SPACE IDENTIFIER |
STARTING QUOTE STRING ENDING QUOTE | DISP IDENTIFIER
if else: = IF OPAR condition CPAR STARTING BLOCK statements
DLR END BLOCK | IF OPAR condition CPAR STARTING BLOCK
statements DLR END BLOCK DLR, else if Loop DLR ELSE
STARTING BLOCK statements DLR END BLOCK | IF OPAR condition
CPAR STARTING BLOCK statements DLR END BLOCK DLR ELSE
OPAR statements DLR CPAR
else if Loop:= elseifLoop1 DLR else if Loop | elseifLoop1
elseifLoop1: = ELSEIF OPAR condition CPAR STARTING BLOCK
```

KVS GRAMMAR – PROGRAM RULES

```
statements DLR END BLOCK
while: = WHILE OPAR condition CPAR STARTING BLOCK statements
END BLOCK
for: = FOR for Range STARTING BLOCK statements END BLOCK
forRange: = OPAR IDENTIFIER ASSIGN expression DLR IDENTIFIER
COMPARING VALUES expression DLR CLOSEPARANTHESIS | OPAR IDENTIFIER
ASSIGN expression DLR IDENTIFIER COMPARING VALUES expression DLR expression.
CPAR | IDENTIFIER IN RANGE OPAR expression COMMA expression CPAR
condition: = IDENTIFIER SPACE COMPARING VALUES SPACE expression | IDENTIFIER SPA
COMPARING VALUES expression CONDITIONAL OPERATORS condition | BOOL
comment: = COMMENT STRING
expression: = value ADDITION expression | value SUBTRACTION expression | value
value: = factor MULTIPLY value | factor DIVISON value | factor
factor: = OPAR expression CPAR | data | IDENTIFIER
data: = INTEGER | BOOL | STRING
```

KVS FEATURES

- DataTypes
  - Num - 1-9
  - Bool - true/false
  - Str - “String”
- Arithmetic Operations
  - Addition:- ‘+’
  - Subtraction:- ’-’
  - Multiplication:- ’*’
  - Division:- ’/’
- Increment Operator:- ++
- Decrement Operator:- –
- Relational Operators
  - Equal to:- ==
  - Not equal to:- !=
  - Greater than:- >
  - Lesser than:- <
  - Greater than or equal to:- >=
  - Lesser than or equal to:- <=
- Logical Operators
  - AND:- and (&&)
  - OR:- or (||)
  - NOT:- not (!!)

KVS FEATURES

Statement Declarations

Assignment Declaration
bool flag=true;

Print Declaration
disp x;

Declaration Declaration
Num a;

If condition

```
start
{
        int a = 5;
        int b = 18;

        if(a != b)
        {
                print a;
        }
        else
        {
                print b;
        }
}
terminate
```

NOTE: else is optional.

KVS FEATURES

For loop

```
start
{
        for(int i=1;i<=5;i++)
        {
                print i;
        }

}
terminate
```

While loop

```
start
{
   while(res <= 500)
      {
          y = y * 2;
          x = x + 2;
          }
}
terminate
```

Ternary Operator

```
x > y ? print x ; : print y;;
```

STEPS INVOLVED IN EXECUTION

The first step in the execution by KVS Language is creating an input file containing the program with the .kvs extension using any standard text editor.

- This input file is used as input for the Lexer for parsing.
- After creating the input file, the next step would be to open Swipl on the terminal.
- Compilation of the kvs.pl using the command: ?-[‘path to the kvs.pl file’].
- Running the input file containing program ?-kvs (' path to lexer.py file',' path to input file with .kvs extension').

SAMPLE PROGRAM

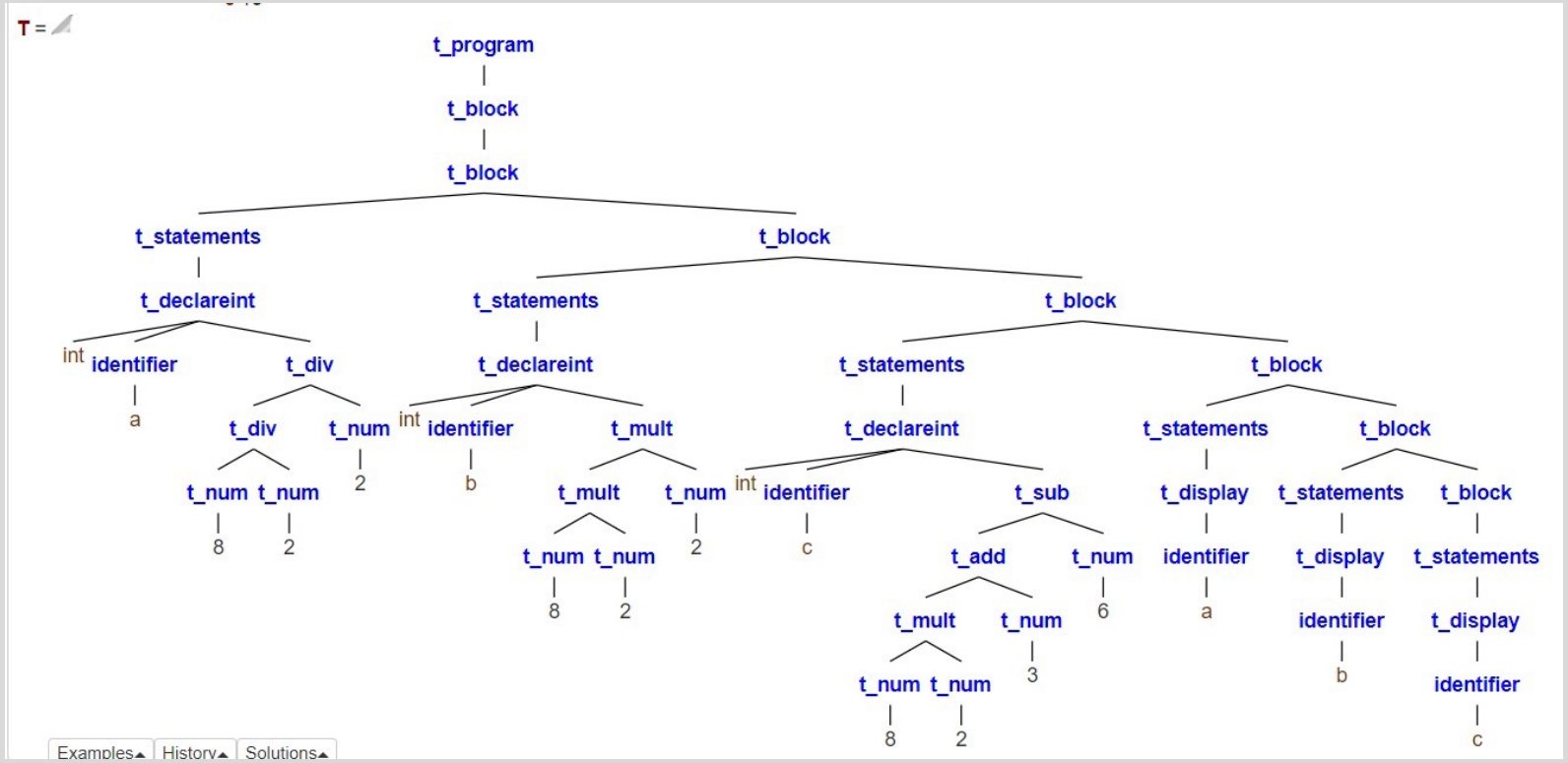
```
# Arithmetic operators
start
{
        int a = 8/2/2;
        int b = 8*2*2;
        int c = 8*2+3-6;
        print a;
        print b;
        print c;
}
terminate
```

EXECUTION OUTPUT

```
program(T,
[start,'{',int,a,=,8,/,2,/,2,;,int,b,=,8,*,2,*,2,;,int,c,=,8,*,2,+,3,-,6,;,print,a,;,print,b,;,print,c,;,'}',terminate],
[]),evaluate_program(T,F).
```

```
Singleton variables: [Output]
2
32
13
F = [(int,a,2), (int,b,32), (int,c,13)],
T =
t_program(
t_block(
t_block(t_statements(t_declareint(int,identifier(a),t_div(t_div(t_num(8),t_num(2)),t_num(2)))),
t_block(t_statements(t_declareint(int,identifier(b),t_mult(t_mult(t_num(8),t_num(2)),t_num(2)))),
t_block(t_statements(t_declareint(int,identifier(c),t_sub(t_add(t_mult(t_num(8),t_num(2)),t_num(3)),t_num(6)))),
t_block(t_statements(t_display(identifier(a))),
t_block(t_statements(t_display(identifier(b))),t_block(t_statements(t_display(identifier(c))))))
)
)
)
)
)
```

EXECUTION OUTPUT



THANK YOU

SUBMITTED BY

- VEDASREE BODAVULA (1225885273)
- SATVIK CHEMUDUPATI (1225665038)
- KAVYA ALLA (1225990508)
- SAI SUNIL NERALLA (1225718832)