SER502 Project

ALBITOR

Team 27

Srilakshmi Sravani Andaluri(1225685461)
Vamsi Krishna Somepalli(1226463994)
Sathwik Reddy Dontham(1225590249)
Saish Vemulapalli (1226212496)
Rohith Reddy Byreddy(1226053974)

Overview of Demonstration

- Language Introduction
- Implementation Overview
- Language Design
- Tools used
- Key Features
- Installation Guide
- Language Grammar
- Interpreter (Evaluator)
- Test run

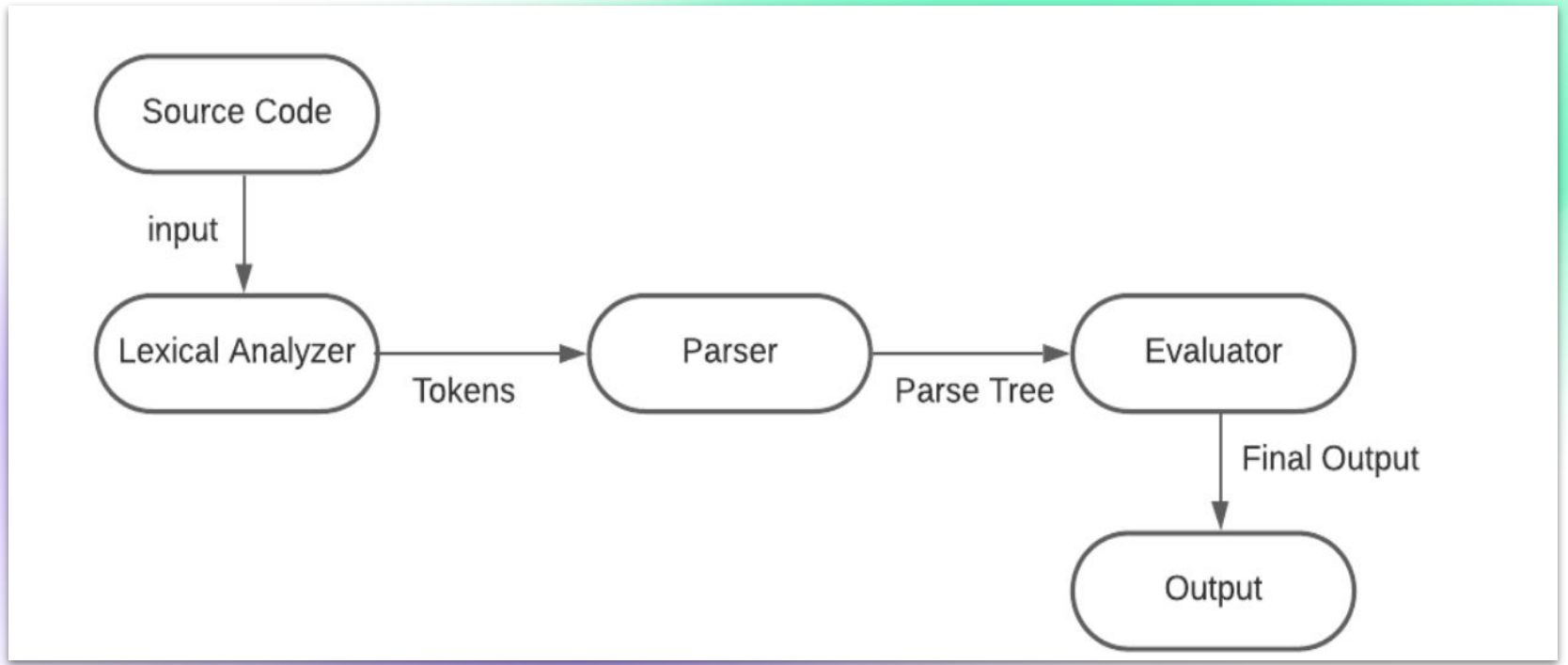
Language Introduction

- Our language “Albitor” is inspired from existing languages such as Python, C and C++.
- The name ALBITOR has been chosen considering its derivation from word “arbiter” which means “someone who makes a judgment, solves an argument, or decides what will be done”.
- Grammar is written in Python whereas Evaluator is written in SWI-Prolog.

Implementation Overview

- We have used Python-Lark for parse tree generation.
- Our language was developed on the Debian Linux subsystem, which provided a reliable and stable environment for our team to work in.
- In addition to our language itself, we also wrote shell scripts to facilitate the execution of sample programs.

Language Design



Tools Used

- **Compiler:** Python3 and Lark-Parser
- **Parse Tree Generator:** Lark-Parser
- **Interpreter:** SWI Prolog

Key Features

Data Types:

- String (str)
- Integer (int)
- Boolean (bool)

Variable/Identifier:

- lowercase letter (a-z)
- Uppercase letter (A-Z)

Conditional Statements:

- 1. Ternary Operator : (Boolean?true:false)
- 2. if-then-else (eg: if Boolean true else false)

Loops:

- For loop (for assignment ; boolean ; increment)
- While loop (while condition true)
- For in range loop (for identifier in range(expression,expression)

Arithmetic Operations :

- Addition - ‘+’
- Subtraction - ’-’
- Multiplication - ’*’
- Division - ’/’
- Brackets - (‘()’)
- Mod - (%)

Comparison Operations:

- Equals (==)
- Not equals (!=)
- Greater than (>)
- Less than (<)
- Greater than or equal to (>=)
- Less than or equal to (<=)

Logic Operations :

- And - (and)
- Or - (or)
- Not - (not)

Boolean:

- true
- false

Print:

- Print the given identifiers which includes boolean, integers and strings.

Installation Guide

- Python3 is required. If not already present install from https://www.python.org/downloads/
- Setup pip and install lark-parser. (pip install lark-parser)
- Install SWI - Prolog using the below link https://www.swi-prolog.org/build/unix.html
- Run the compiler and interpreter using command line and run the following command

  Compilation command:
  python3 file_extract.py <program file with extension>

  Interpreter command:
  swipl -s evalFile.pl -g "run_program(<parse_tree>), halt."

Language Grammar

```
import lark

def parser_hy(program):
    p = lark.Lark(

    r'''

    program : BEG block END

    block : CBO command CBC

    command :   statement command
                | statement

    statement : print_statement SC
                | assignment_statement SC
                | declaration_statement SC
                | for_loop
                | while_loop
                | if_condition
                | ternary_operation

    print_statement :   PRINT PO var PC

    assignment_statement :  declaration_statement EQ expression
                            | var EQ expression

    declaration_statement : data_type var

    for_loop :  FOR PO assignment_statement SC compare_operations SC assignment_statement PC block
                | FOR PO var IN RANGE PO arithmetic_operation COM arithmetic_operation PC PC block

    while_loop :    WHILE PO compare_operations PC block

    if_condition :  IF PO compare_operations PC block
                    | IF PO compare_operations PC block ELSE block
```

Language Grammar(continued)

```
expression :    boolean
                | string_literal
                | arithmetic_operation

arithmetic_operation :  arithmetic_operation1 ADDITION arithmetic_operation
                        | arithmetic_operation1 SUBTRACTION arithmetic_operation
                        | arithmetic_operation1

arithmetic_operation1 : arithmetic_operation2 MULTIPLICATION arithmetic_operation1
                        | arithmetic_operation2 DIVISION arithmetic_operation1
                        | arithmetic_operation2 PERCENTAGE arithmetic_operation1
                        | arithmetic_operation2

arithmetic_operation2 : PO arithmetic_operation PC
                        | NUMBER
                        | var

compare_operations :    compare_statement LOGICAL_OPERATOR compare_operations
                        | compare_statement

compare_statement :     arithmetic_operation COMPARISON_OPERATOR arithmetic_operation
                        | NOT compare_statement


ternary_operation : compare_operations QUE block COL block

COMPARISON_OPERATOR:    "<"
                        | "<="
                        | ">"
                        | ">="
                        | "=="
                        | "!="

LOGICAL_OPERATOR :      "&&"
                        | "||"

boolean :   TRUE
            | FALSE

NUMBER :    /[+-]?\d+/

data_type : INTEGER
            | BOOL
            | STRING
```

Language Grammar(continued)

```
INTEGER :   "int"
BOOL    :   "bool"
STRING  :   "str"
BEG : "begin"
END : "end"
NOT : "not"
ADDITION : "+"
SUBTRACTION : "-"
DIVISION : "/"
MULTIPLICATION : "*"
PERCENTAGE : "%"
EQ : "="
SC : ";"
CBO : "{"
CBC : "}"
PO : "("
PC : ")"
QUE : "?"
COL : ":"
DQO : "'"
DQC : "'"
IF : "if"
ELSE : "else"
IN : "in"
FOR : "for"
PRINT : "print"
TRUE : "true"
FALSE : "false"
COM : ","
RANGE : "range"
WHILE : "while"
```

Language Grammar (continued)

```
    var :   VARIABLE_NAME

    string_literal :    /"(\\.|[^\\"])*"/

    %import common.WORD -> VARIABLE_NAME


    WHITE : /\s+(?=([^\"]*[\"][^\"]*[\"])*[^\"]*$)/
    %ignore WHITE

    ''',
    parser = 'lalr',
    start = 'program'
    )

    return str(p.parse(program))

def progaram_parser(program):

    tree = parser_hy(program)
```

Interpreter (Evaluator)

```
run_program(P):- eval_program(P, []).

% Predicate for evaluating the program.
eval_program(tree(program,[_,Block,_]), InitialEnv):- eval_block(Block,InitialEnv, _EnvRes).

% Predicate for evaluating the block.
eval_block(tree(block,[_,Cmd,_]), Env, EnvRes):- eval_command(Cmd, Env, EnvRes).

% Predicate for evaluating the command.
eval_command(tree(command,[Stmt,Cmd]),Env, EnvRes):- eval_statement(Stmt,Env, EnvTemp), eval_command(Cmd,EnvTemp, EnvRes).
eval_command(tree(command,[Stmt]), Env, EnvRes):- eval_statement(Stmt, Env, EnvRes).

% Predicate for evaluating statement.
eval_statement(tree(statement,[Print_stmt,_]), Env, EnvRes):- eval_print_statement(Print_stmt, Env, EnvRes).
eval_statement(tree(statement,[Assignment_stmt,_]), Env, EnvRes):- eval_assignment_statement(Assignment_stmt, Env, EnvRes).
eval_statement(tree(statement,[Declaration_stmt]), Env, EnvRes):- eval_declaration_statement(Declaration_stmt, Env, EnvRes).
eval_statement(tree(statement,[For_loop]), Env, EnvRes):- eval_for_loop(For_loop, Env, EnvRes).
eval_statement(tree(statement,[While_loop]), Env, EnvRes):- eval_while_loop(While_loop, Env, EnvRes).
eval_statement(tree(statement,[If_condition]), Env, EnvRes):- eval_if_condition(If_condition, Env, EnvRes).
eval_statement(tree(statement,[Ternary_operation]), Env, EnvRes):- eval_ternary_operation(Ternary_operation, Env, EnvRes).

% Predicate for evaluating Print statement.
eval_print_statement(tree(print_statement,[_,_,Var,_]),Env,Env) :-
    eval_var(Var, Env, Var1),
    search(Var1, Env, X),
    write(X),nl.

% Predicate for evaluating Assignment statement.
eval_assignment_statement(tree(assignment_statement,[Declaration_stmt,_,Expr]),Env, EnvRes):-
    eval_declaration_statement(Declaration_stmt,Env, Env1),
    eval_expression(Expr,Env, Env2, Value),
    assign(Env1, Value, Env2, EnvRes).

eval_assignment_statement(tree(assignment_statement,[Var,_,Expr]),Env, EnvRes):-
    eval_var(Var,Env, Env1),
    eval_expression(Expr,Env, Env, Value),
    assign(Env1, Value, Env, EnvRes).

```

Interpreter (Evaluator)(continued)

```
229
230 eval_compare_statement(tree(compare_statement,[token(_NOT,'not'), Compare_stmt]), Env, EnvRes, true) :-
231     not(eval_compare_statement(Compare_stmt, Env, EnvRes, true)).
232
233 eval_compare_statement(tree(compare_statement,[token(_NOT,'not'), Compare_stmt]), Env, EnvRes, false) :-
234     eval_compare_statement(Compare_stmt, Env, EnvRes, true).
235
236 % Predicate used for evaluating Ternary Operation.
237 eval_ternary_operation(tree(ternary_operation,[Compare_operations,_,Block,_,_]),Env,EnvRes):-
238     eval_compare_operations(Compare_operations, Env, EnvTemp, true),
239     eval_block(Block, Env, EnvRes).
240
241 eval_ternary_operation(tree(ternary_operation,[Compare_operations,_,_,_,Block]),Env,EnvRes):-
242     eval_compare_operations(Compare_operations, Env, EnvTemp, false),
243     eval_block(Block, Env, EnvRes).
244
245 %Predicate to evaluate boolean.
246 eval_boolean(tree(boolean,[token(TRUE,'true')]),Env,Env,true).
247 eval_boolean(tree(boolean,[token(FALSE,'false')]),Env,Env,false).
248
249 %Predicate to evaluate number and var
250 eval_number(tree(number,[Number]),_, Number).
251 eval_var(tree(var,[token(VARIABLE_NAME, Var)]),_, Var).
252
253 % assign identifiers with values
254 assign(A, B, [], [(A, B)]).
255 assign(A, B, [(A, _)|C], [(A, B)|C]).
256 assign(A, B, [H|T], [H|U]) :-
257     H \= (A, _),
258     assign(A, B, T, U).
259
260 % lookup for a value in variable set
261 search(A, [(A,B)|_], B).
262 search(A, [_|B], C) :- search(A, B, C).
```

Test run

```
begin
{
int a=12;
str n="a is even number";
str m="a is odd number";
int c=a-2*(a/2);
if ( c == 0) {
print(n);
}
else {
print(m);
}
}
end
```

```
begin
{
int n=15;
int m=3;
int s=n+m;
int p=n-m;
int q=n*m;
int r=n/m;
int aa=n%m;
str a= "Addition:";
print (a);
print (s);
str b="Substraction:";
print (b);
print (p);
str c="Multiplication:";
print (c);
print (q);
str d="Division:";
print (d);
print (r);
str bb="reminder";
print(bb);
print(aa);
}
end
```

Test run(continued)

```
dellsathwik@DESKTOP-0N9736D:~/home/project/SER502-Spring2023-Team27-Dev/SER502-Spring2023-Team27-Dev/src$ sh testRuns.sh
Compiling programs...

Compilation done successfully

interpreting the programs

testRuns.sh: 15: Sleep: not found
-------------------------------------------------------
code 1 program
-------------------------------------------------------
"a is even number"

-------------------------------------------------------
code 2 Program
-------------------------------------------------------
"Addition:"
18
"Substraction:"
12
"Multiplication:"
45
"Division:"
5
"reminder"
0
```

Test run(continued)

```
begin
{
int a=5;
int b=6;
str s="greater number:";
int c=0;
a<b?{c=b;}:{c=a;}
print(s);
print(c);
}
end
```

```
begin
{
int a=1;
int b=6;
while(a<=b)
{
print(a);
a=a+1;
}
}
end
```

Test run(continued)

```
------------------------------------------------------------
code 3 Program
------------------------------------------------------------
"greater number:"
6


------------------------------------------------------------
code 4 Program
------------------------------------------------------------
1
2
3
4
5
6
```

Test run(continued)

```
begin
{
int n=5;
int sum=0;
for(int i=0;i<=n;i=i+1)
{
sum=sum+i;
}
str a="sum of first 5 numbers is";
print(a);
print(sum);
}
end
```

```
begin
{
int x=10;
int y=20;
int n=0;
for(i in range(x,y))
{
n=n+1;
}
str a="number of elements";
print(a);
print(n);
}
end
```

```
begin
{
int x=5;
int y=10;
int z=0;
if(x > 0 and y > 0)
{
str a="And operation";
print(a);
}
if(x > 0 or y < 0)
{
str b="Or operation";
print(b);
}
if(not z != 0)
{
str c="Not operation";
print(c);
}
bool x=true;
print(x);
}
end
```

Test run(continued)

```
------------------------------------------------------------
code 5 Program
------------------------------------------------------------
"sum of first 5 numbers is"
15

------------------------------------------------------------
code 6 Program
------------------------------------------------------------
"number of elements"
10

------------------------------------------------------------
code 7 Program
------------------------------------------------------------
"And operation"
"Or operation"
"Not operation"
true
------------------------------------------------------------
Done!
```

Thank you