502 milestone 2 Presentation

Spectra

Team members

Akshat Nambiar (ASU ID: 1225484000)
Manideep Nalluri (ASU ID: 1225915641)
Ryan Collins (ASU ID: 1225687957)
Sai Prakash Ravichandran (ASU ID: 1225761147)
Sai Viswas Nirukonda (ASU ID: 1225421353)

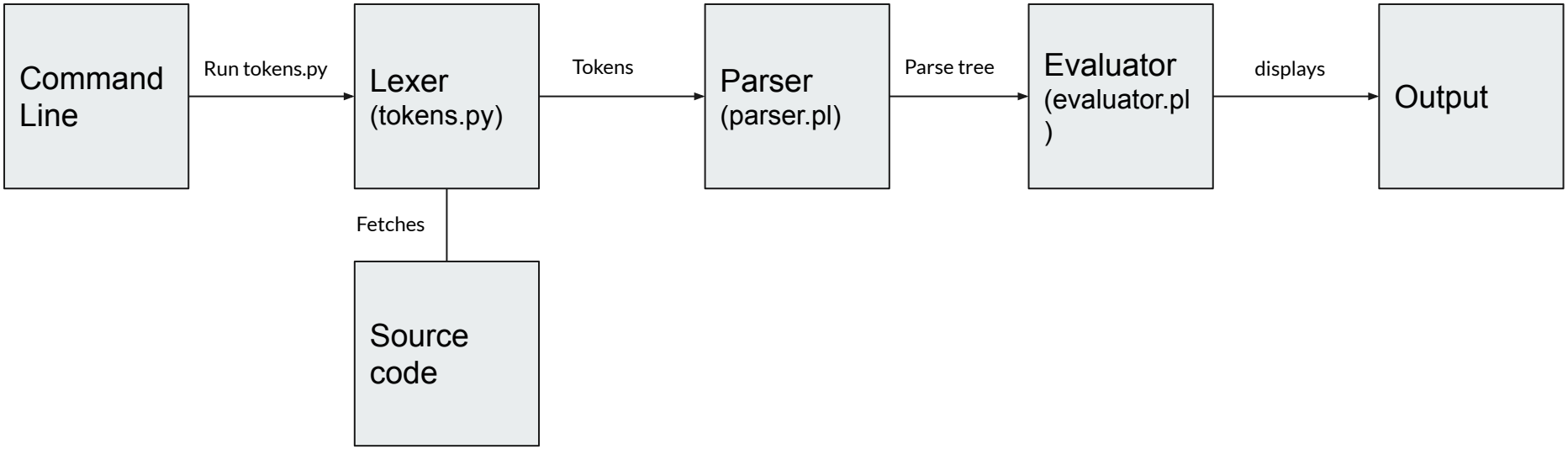
Overview

- Introduction
- Flow of execution - Design
- Lexer & Parser
- Grammar
- Semantics
- Execution

Introduction

- Name of the language: Spectra
- Extension: .spe
- Data structure used in tokenization: Lexer. Specifically, it uses the lexer provided by the SLY library, which is a Python implementation of lex and yacc parsing tools.
- Language/tools used to create tokenizer/lexer: Python
- Language/tools used to create parser and evaluator: Prolog

Flow of execution - Design



Design of the language (cont.)

Spectra code undergoes a series of steps before generating its output. The first step is the lexical analysis process, where the program's source code is analyzed to break it down into smaller units of meaning, called tokens. These tokens include keywords, identifiers, and operators, and are important in identifying syntax errors in the code.

The second step is the parsing process, where the code's syntax is analyzed to determine if it's grammatically correct. If it is, a parse tree data structure is created to represent the program's structure. This tree is used to ensure that the code's instructions are executed in the correct order, making it easier for the interpreter to translate the high-level code into machine code.

Finally, the interpreter reads the parse tree and executes the program by following the instructions specified by the code. It translates the code into machine code, which the computer can understand and execute. The interpreter evaluates expressions, performs operations, and makes decisions based on conditional statements, allowing the program to run and produce output. Overall, the process of lexical analysis, parsing, and interpreting is critical in ensuring that the Spectra code is correct, efficient, and produces the desired results.

Components of the Design

Lexer: Lexer takes in the source code and breaks it down into tokens.
Parser: The tokens generated from the lexer is passed into the parser and an abstract parse tree is generated.
Evaluator: The evaluator takes in the parse tree and generates the output by executing the instructions from the sparse tree.

Commands:

While Loop
For loop
Enhanced for loop
If
If Else
If Else Ladder
Variable declaration and Assignment

Commands:

While Loop
For loop
Enhanced for loop
If
If Else
If Else Ladder
Variable declaration and Assignment

Features:

Data types:
- Integer
- Float
- String
- Boolean

Operations:
- Addition
- Subtraction
- Multiplication
- Division
- Ternary Operator

Grammar

```
/*
Mapping
P --> Program
K --> Block
D --> Declaration
CL --> Command List
C --> Multi Line Command
E --> Expression
I --> Identifier
*/


P ::= CL.

K ::= ['{'], CL, ['}'].

CL ::= C, CL.
      |single_line_commands, CL.
      |C.
      |single_line_commands.


single_line_commands ::= print.
                        |assignment_command.
                        |D.
```

```
C ::= for_loop.
      |while_loop.
      |for_range.
      |if_command.
      |if_elif_else_command.
      |if_else_command.


if_command ::= if_part.
if_elif_else_command ::= if_part, elif_part, else_part.
if_else_command ::= if_part, else_part.

if_part ::= ['if'], ['('], condition, [')'], K.
else_part ::= ['else'], K.
            |['elif'], ['('], condition, [')'], K.
            |['elif'], ['('], condition, [')'], K, elif_command.

while_loop ::= ['while'], ['('], condition, [')'], K.

for_range ::= ['for'], I, ['in'], ['range'], ['('], inRange, [';'], inRange, [')'], K.

inRange ::= I | integer.

for_loop ::= ['for'], ['('], assignment, [';'], condition, [';'], variableChange, [')'], K.
```

```
variableChange ::= increment.
                  |decrement.
                  |I, assignmentConstruct, E.

condition ::= E, comparisonConstructs, E.

decrement ::= I, decrementConstruct.
             |decrementConstruct, I.
increment ::= I, incrementConstruct.
             |incrementConstruct, I.

print ::= [print_string], ['('], string, [')'], end_of_command.
         |[print_string], ['('], I, [')'], end_of_command.
         |[print_expression], ['('], E, [')'], end_of_command.

ternary_expression ::= ['('], condition, [')'], ['?'], E, [':'], E.

value ::= float | integer | boolean | string | I.

boolean_operators ::= andConstruct | orConstruct | notConstruct.

operators ::= ['+'] | ['-'] | ['*'] | ['/'] | boolean_operators.

assignment_command ::= I, assignmentConstruct, E, end_of_command.
```

```
D ::= variable_type, I, end_of_command.
     |variable_type, I, assignmentConstruct, E, end_of_command.

I ::= lower_case, I.
     |I, upper_case.
     |I, upper_case, I.
     |I, ['_'], I.
     |lower_case.

string ::= single_quote, character_phrase, single_quote.
          |double_quote, character_phrase, double_quote.

character_phrase ::= character, character_phrase.
                    |character.

character ::= lower_case | upper_case | digit | symbol.

float ::= integer, ['.'], integer.
         |integer.

integer ::= digit, integer.
           |digit.
```

```
variable_type --> int | float | bool | string.

decrementConstruct --> --.
incrementConstruct --> ++.

comparisonConstructs --> < | > | <= | >= | == | !=.

single_quote --> '.
double_quote --> ".

lower_case --> 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.

upper_case --> 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.

symbol --> ' '| ! | " | # | $ | & | ' | ( | ) | * | + | , | - | . | / | : | ; | < | = | > | ? | @ | [ | \ | ] | ^ | _ | ` | { | | | } | ~.

digit --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9.

boolean --> ['True'] | ['False'].

assignmentConstruct --> =.
end_of_command --> ;.

andConstruct --> and.
orConstruct --> or.
notConstruct --> not.
```

Execution steps and screenshots

```
PS C:\Users\Welcome\Desktop\github test\SER502-Spring2023-Team35\src> python tokens.py --evaluate sample.spe
Tokenization in progress
Tokens are stored in sample.spetokens

Parser in progress
Parse Tree generation in progress:
t_program(t_comm_list(t_decl(t_variable_type(int),t_iden(x)),t_comm_list(t_assign(t_iden(x),t_expr(t_integer(3))),t_comm_list(t_decl(t_variable_type(int),t
en(y),t_expr(t_divide(t_expr(t_add(t_integer(2),t_integer(3))),t_integer(5)))),t_comm_list(t_decl(t_variable_type(int),t_iden(z),t_expr(t_add(t_iden(x),t_
(y)))),t_comm_list(t_decl(t_variable_type(int),t_iden(i),t_expr(t_integer(5))),t_comm_list(t_decl(t_variable_type(int),t_iden(j)),t_comm_list(t_decl(t_vari
e_type(bool),t_iden(chk),t_expr(t_boolean(true))),t_comm_list(t_decl(t_variable_type(float),t_iden(num),t_expr(t_float(0.314))),t_comm_list(t_print_string
* PRINT TEST ***"),t_comm_list(t_print_expr(t_expr(t_iden(z))),t_comm_list(t_print_expr(t_expr(t_sub(t_integer(120),t_integer(100)))),t_comm_list(t_print_
ng("*** FOR LOOP TEST ***"),t_comm_list(t_for_range_comm(t_iden(i),t_expr(t_integer(1)),t_expr(t_integer(3)),t_block(t_comm(t_for_loop_comm(t_assign(t_iden
,t_expr(t_integer(1))),t_bool(t_expr(t_iden(j)),t_comp_opr(<),t_expr(t_integer(3))),t_post_inc(t_iden(j)),t_block(t_comm(t_print_expr(t_expr(t_iden(j)))))
,t_comm_list(t_print_string("*** IF ELSE TEST ***"),t_comm_list(t_decl(t_variable_type(int),t_iden(a),t_expr(t_integer(10))),t_comm_list(t_if_comm(t_if(t_
(t_expr(t_iden(a)),t_comp_opr(==),t_expr(t_integer(5))),t_block(t_comm(t_print_string("Equal to 5")))),t_elif(t_bool(t_expr(t_iden(a)),t_comp_opr(==),t_exp
_integer(10))),t_block(t_comm(t_print_string("Equal to 10"))),t_elif(t_bool(t_expr(t_iden(a)),t_comp_opr(==),t_expr(t_integer(15))),t_block(t_comm(t_print
ing("Equal to 15"))))),t_else(t_block(t_comm(t_print_string("Def"))))),t_comm_list(t_print_string("*** WHILE TEST ***"),t_comm_list(t_while_comm(t_bool(t_
(t_iden(a)),t_comp_opr(<),t_expr(t_integer(15))),t_block(t_comm_list(t_print_expr(t_expr(t_iden(a))),t_comm(t_assign(t_iden(a),t_expr(t_add(t_iden(a),t_in
r(1)))))))),t_comm_list(t_print_string("*** BOOLEAN EXP TEST ***"),t_comm_list(t_decl(t_variable_type(bool),t_iden(isTrue),t_expr(t_boolean_expression(t_ex
t_boolean(true)),t_boolean_operator(or),t_expr(t_boolean(false))))),t_comm_list(t_decl(t_variable_type(bool),t_iden(isFalse),t_expr(t_boolean_expression(t_
r(t_boolean(true)),t_boolean_operator(and),t_expr(t_boolean(false))))),t_comm_list(t_print_string("True"),t_comm_list(t_print_expr(t_expr(t_iden(isTrue))),
omm_list(t_print_string("False"),t_comm_list(t_print_expr(t_expr(t_iden(isFalse))),t_comm_list(t_print_string("*** TERNARY EXP TEST ***"),t_comm_list(t_de
_variable_type(int),t_iden(h),t_expr(t_ternary_expression(t_bool(t_expr(t_integer(1)),t_comp_opr(>),t_expr(t_integer(2))),t_expr(t_add(t_integer(1),t_inte
1))),t_expr(t_add(t_integer(2),t_integer(2)))))),t_comm(t_print_expr(t_expr(t_iden(h)))))))))))))))))))))))))))))))))
Evaluation in progress
"*** PRINT TEST ***"
4
20
```

```
"*** FOR LOOP TEST ***"
1
2
1
2
1
2
"*** IF ELSE TEST ***"
"Equal to 10"
"*** WHILE TEST ***"
10
11
12
13
14
"*** BOOLEAN EXP TEST ***"
"True"
true
"False"
false
"*** TERNARY EXP TEST ***"
4

Environment after evaluation
[(bool,isTrue,true),(bool,isFalse,false),(int,h,4)]
PS C:\Users\Welcome\Desktop\github test\SER502-Spring2023-Team35\src>
```