SER 502-Spring 2023-Team 15

PHOENIX

Team Members
Heet Punjawat
Shivanjay Wagh
Janki Padiya
Manan Kohli
Omkar Pisal

COURSE
SER 502

TEAM15

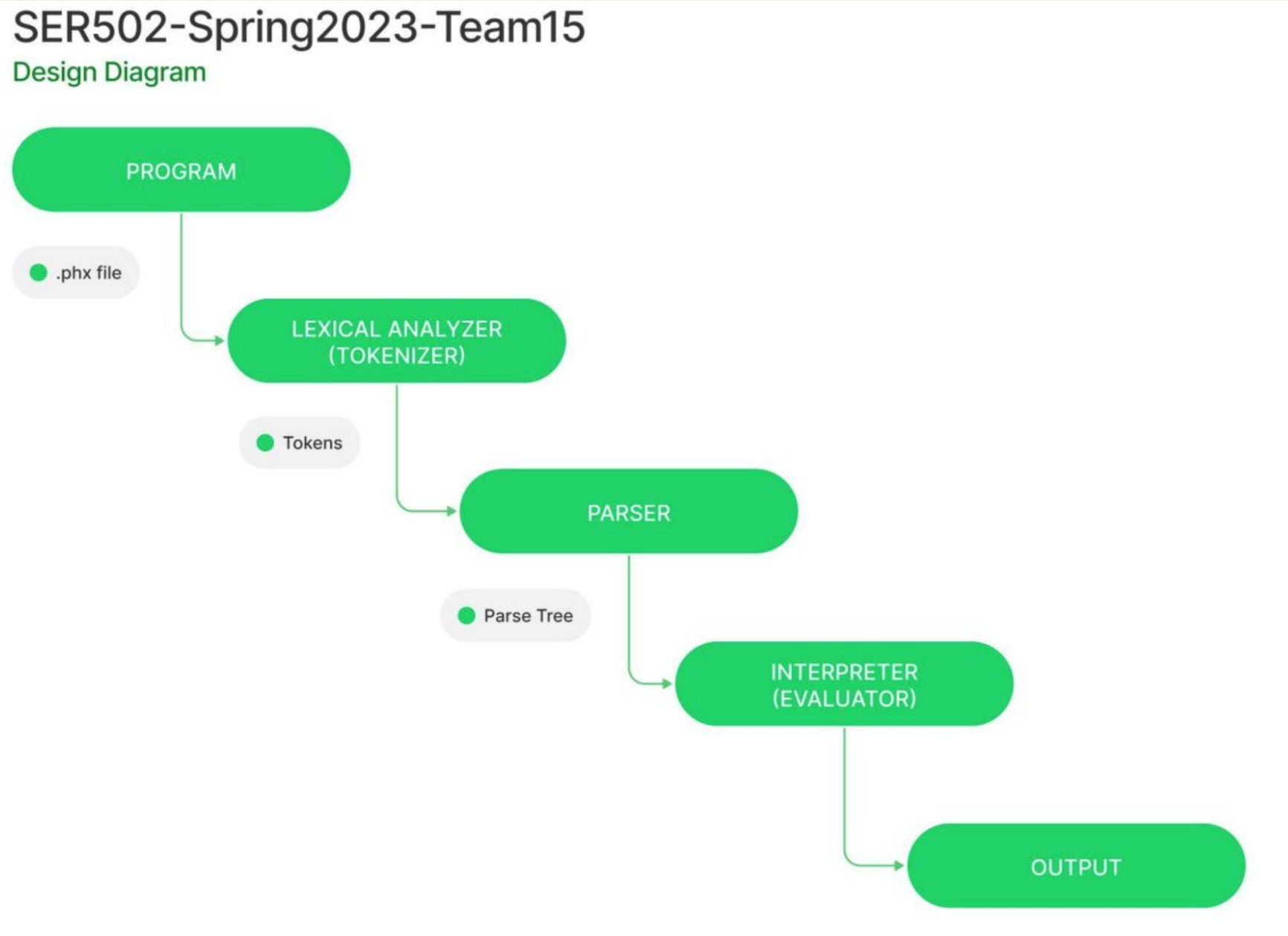
Overview

- About the Language
- Design
- Grammar
- Snapshot
- Future Scope

About the Language

Name - Phoenix
File Extension - .phx
Programming Paradigm - Imperative
Programming Languages Used
- Python (Lexer)
- Prolog (Parser and Evaluator)
Data Structures Used - List

Design


SER502-Spring2023-Team15
Design Diagram
PROGRAM
.phx file
LEXICAL ANALYZER (TOKENIZER)
Tokens
PARSER
Parse Tree
INTERPRETER (EVALUATOR)
OUTPUT

Project Pipeline

XXX.PHX → LEXER.PY → XXX.PHXTOKENS → PARSER.PL → EVALUATOR.PL

Components of The Design

Program - A program written in Phoenix (.phx) language will be processed by computer to produce the desired output. The processing of a program involves several stages, including lexical analysis, parsing, interpretation or evaluation, and output generation.

Lexical Analyzer (Tokenizer) - Lexical Analyzer will accept the source code (.phx) file as an input and produce a list of tokens. We will use Python for lexical analysis. Each token represents a specific type of lexical element such as keywords, identifiers, operators, constants, and punctuation marks. A stream of tokens will be produced by lexical analyzer that can be used as input for the next stage of the compilation process, which is parsing.

Parser - The parser checks if syntax and format of the written program is correct. It makes sure that the program follows the rules of the programming language. It generates the parse tree using the tokens provided by the lexical analyzer. A parser builds a parse tree using a pre-defined set of rules called a grammar. A parser uses grammar to check whether a program conforms to the syntax of the language.

Evaluator(Interpreter) - The evaluation process will start as soon as the evaluator approves and uses the parse tree that the parser produced. The evaluator uses the parse tree essentially as a blueprint to carry out the relevant set of instructions.

Features

Commands

- ❑ For loop
- ❑ While loop
- ❑ For loop (Enhanced)
- ❑ If
- ❑ If Elif Else
- ❑ If Else
- ❑ Print
- ❑ Variable declaration
- ❑ Variable assignment

Variable Naming

- ❑ Variable name can only start with lower case letter.
- ❑ Variable name can contain lower case or upper case letter.
- ❑ Variable name can contain underscores.

Features

Data Types

- ❑ Integer
- ❑ Floating point numbers
- ❑ String
- ❑ Boolean

Operations

- ❑ Addition
- ❑ Subtraction
- ❑ Multiplication
- ❑ Division
- ❑ Braces
- ❑ Ternary operators

Features

Reserved Keywords

- ❑ False - Boolean Value
- ❑ True - Boolean Value
- ❑ and - Logical Operator
- ❑ bool - Variable Type
- ❑ elif - Conditional Command
- ❑ else - Conditional Command
- ❑ float - Variable Type
- ❑ for - Loop Command
- ❑ if - Conditional Command
- ❑ in - Checks if value is present in a list
- ❑ int - Variable
- ❑ not - Logical Operator
- ❑ or - Logical Operator
- ❑ string - Variable Type
- ❑ while - Loop Command

Grammar

```
---------------------
% TERMINALS %
---------------------

bool_val --> ['True'].
bool_val --> ['False'].

var_type --> ['int'] | ['float'] | ['bool'] | ['string'].

and_operator --> ['and'].
or_operator --> ['or'].
not_operator --> ['not'].

assignment_operator --> ['='].
end_of_statement --> [';'].

inc_operator --> ['++'].
dec_operator --> ['--'].

comp_operators --> ['<'], ['>'], ['<='], ['>='], ['=='], ['!='].

ternary_operator --> ['?']

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'].

single_quote --> ['\''].
double_quote --> ['\'"'].
```

```
--------------------
/* NON-TERMINALS */
--------------------

program --> statement_list.

block --> ['{'], statement_list, ['}'].

statement_list --> statement.
statement_list --> statement, statement_list.
statement_list --> statement_without_block.
statement_list --> statement_without_block, statement_list.


% statements without block (simple statements)
statement_without_block --> print_statement.
statement_without_block --> assign_statement.
statement_without_block --> var_decl_statement.

% Multi Line statements (complex statements)
statement --> while_loop_statement.
statement --> for_loop_statement.
statement --> for_enhanced_statement.
statement --> if_statement.
statement --> if_elif_else_statement.
statement --> if_else_statement.
```

```
elif_clause --> ['elif'], ['('], condition, [')'], block.
elif_clause --> ['elif'], ['('], condition, [')'], block, elif_clause.

if_statement --> if_clause.
if_elif_else_statement --> if_clause, elif_clause, else_clause.
if_else_statement --> if_clause, else_clause.

while_loop_statement --> ['while'], ['('], condition, [')'], block.

for_enhanced_statement --> ['for'], var_name, ['in'], ['range'], ['('], range_val, [';'],
range_val, [')'], block.

range_val --> var_name | integer.

for_loop_statement --> ['for'], ['('], assign_statement, [';'], condition, [';'],
var_change_part, [')'], block.

var_change_part --> inc_expression.
var_change_part --> dec_expression.
var_change_part --> var_name, assignment_operator, expression.

condition --> expression, comp_operators, expression.
```

```
elif_clause --> ['elif'], ['('], condition, [')'], block.
elif_clause --> ['elif'], ['('], condition, [')'], block, elif_clause.

if_statement --> if_clause.
if_elif_else_statement --> if_clause, elif_clause, else_clause.
if_else_statement --> if_clause, else_clause.

while_loop_statement --> ['while'], ['('], condition, [')'], block.

for_enhanced_statement --> ['for'], var_name, ['in'], ['range'], ['('], range_val, [';'],
range_val, [')'], block.

range_val --> var_name | integer.

for_loop_statement --> ['for'], ['('], assign_statement, [';'], condition, [';'],
var_change_part, [')'], block.

var_change_part --> inc_expression.
var_change_part --> dec_expression.
var_change_part --> var_name, assignment_operator, expression.

condition --> expression, comp_operators, expression.
```

```
inc_expression --> var_name, inc_operator.
inc_expression --> inc_operator, var_name.
dec_expression --> var_name, dec_operator.
dec_expression --> dec_operator, var_name.

print_statement --> [print_str], ['('], string_val, [')'], end_of_statement.
print_statement --> [print_str], ['('], var_name, [')'], end_of_statement.
print_statement --> [print_expr], ['('], expression, [')'], end_of_statement.

expression --> value.
expression --> value, operator, expression.
expression --> ternary_expression.
expression --> ['('], expression, [')'], operator, expression.

ternary_expression --> ['('], condition, [')'], ['?'], expression, [':'], expression.

value --> float | integer | bool_val | string_val | var_name.
```

```
assign_statement --> var_name, assignment_operator, expression, end_of_statement.

var_decl_statement --> var_type, var_name, end_of_statement.
var_decl_statement --> var_type, var_name, assignment_operator, expression,
end_of_statement.

var_name --> lower_case, var_name.
var_name --> var_name, upper_case.
var_name --> var_name, upper_case, var_name.
var_name --> var_name, ['_'], var_name.
var_name --> lower_case.

string_val --> single_quote, char_phrase, single_quote.
string_val --> double_quote, char_phrase, double_quote.

char_phrase --> char, char_phrase.
char_phrase --> char.

char --> lower_case | upper_case | digit | symbol.

float --> integer, ['.'], integer.
float --> integer.

integer --> digit, integer.
integer --> digit.
```

Code Snapshots

1. Open the terminal and set the directory to source folder inside the project folder

```
Last login: Fri Apr 28 23:13:51 on ttys001
(base) gray@Heets-Air ~ % cd /Users/gray/Desktop/Sem\ 2\ Notes/SER\ 502/Project\ /SER502-Spring2023-Team15
(base) gray@Heets-Air SER502-Spring2023-Team15 % cd src
(base) gray@Heets-Air src %
```

2. Run the following command for the xxx.phx file.

```
Last login: Fri Apr 28 23:13:51 on ttys001
(base) gray@Heets-Air ~ % cd /Users/gray/Desktop/Sem\ 2\ Notes/SER\ 502/Project\ /SER502-Spring2023-Team15
(base) gray@Heets-Air SER502-Spring2023-Team15 % cd src
(base) gray@Heets-Air src % python lexer.py --evaluate testFinal.phx
```

xxx.phx source code (we're using the testFinal.phx file here)

EXPLORER

p.phx | testWhile.phx | testNesting.phx | README.md

SER502-SPRING2023-...

- doc
  - contribution.txt
  - SER502-Spring2023-Team15-Do...
- src
  - evaluator.pl
  - grammar
  - lexer.py 9+
  - main.pl
  - parser.pl
  - testBasicForLoop.phx
  - testEnhancedNesting.phx
  - testFinal.phx
  - testNesting.phx
  - testPrint.phx
  - testWhile.phx
  - tokenReader.pl
- README.md

src > testFinal.phx

```
int x;
x = 5;

int y = (15 + 25)/2;

int z = x + y;

bool isBoolean = true;
float number = 0.314;


# PRINT STATEMENTS
print_str("Testing Print statements");
print_expr(10-5);
print_expr(z);
print_expr(number);


# REGULAR LOOP INSIDE ENHACED LOOP
print_str("Regular loop inside Enhanced loop");
int i;
int j;
for i in range(1;3) {
    for(j=1; j<6; j++) {
        print_expr(j);
    }
}


# WHILE
```

5-Do... 9+

```
# WHILE
print_str("Testing While Loop");
int a = 3;
while(a<10){
    print_expr(a);
    a = a + 1;
}


# Boolean Expressions
print_str("Testing Boolean expressions");
bool isTrue = true;
bool isFalse = false;
print_str("isTrue");
print_expr(isTrue);
print_str("isFalse");
print_expr(isFalse);


# TERNARY EXPRESSION
print_str("Testing Ternary expressions");
int T = (15>20 ? 15+20:20-15);
print_expr(T);
```

Output

Lexer has started, source code has been scanned and tokenized, ans sent to the parser.

```
Last login: Fri Apr 28 23:13:51 on ttys001
(base) gray@Heets-Air ~ % cd /Users/gray/Desktop/Sem\ 2\ Notes/SER\ 502/Project\ /SER502-Spring2023-Team15
(base) gray@Heets-Air SER502-Spring2023-Team15 % cd src
(base) gray@Heets-Air src % python lexer.py --evaluate testFinal.phx
Lexer Running
Source Code Scanning: SUCCESS
Tokens write operation testFinal.phxtokens: SUCCESS

Starting Parser
```

Parse Tree is generated

```
Starting Parser
Generating Parse Tree: SUCCESSFUL
t_program(t_statement_list(t_var_decl_statement(t_var_type(int),t_var_name(x)),t_statement_list(t_assignment_expression(t
_var_name(x),t_expression(t_integer(5))),t_statement_list(t_var_decl_statement(t_var_type(int),t_var_name(y),t_expression
(t_divide(t_expression(t_add(t_integer(15),t_integer(25))),t_integer(2)))),t_statement_list(t_var_decl_statement(t_var_ty
pe(int),t_var_name(z),t_expression(t_add(t_var_name(x),t_var_name(y)))),t_statement_list(t_var_decl_statement(t_var_type(
bool),t_var_name(isBoolean),t_expression(t_boolean(true))),t_statement_list(t_var_decl_statement(t_var_type(float),t_var_
name(number),t_expression(t_float(0.314))),t_statement_list(t_print_str("Testing Print statements"),t_statement_list(t_pr
int_expr(t_expression(t_sub(t_integer(10),t_integer(5)))),t_statement_list(t_print_expr(t_expression(t_var_name(z))),t_st
atement_list(t_print_expr(t_expression(t_var_name(number))),t_statement_list(t_print_str("Regular loop inside Enhanced lo
op"),t_statement_list(t_var_decl_statement(t_var_type(int),t_var_name(i)),t_statement_list(t_var_decl_statement(t_var_typ
e(int),t_var_name(j)),t_statement_list(t_for_enhanced_statement(t_var_name(i),t_expression(t_integer(1)),t_expression(t_i
nteger(3)),t_block(t_statement(t_for_loop_statement(t_assignment_expression(t_var_name(j),t_expression(t_integer(1))),t_c
ondition(t_expression(t_var_name(j)),t_comp_operator(<),t_expression(t_integer(6))),t_after_increment(t_var_name(j)),t_bl
ock(t_statement(t_print_expr(t_expression(t_var_name(j))))))))),t_statement_list(t_print_str("Testing While Loop"),t_stat
ement_list(t_var_decl_statement(t_var_type(int),t_var_name(a),t_expression(t_integer(3))),t_statement_list(t_while_statem
ent(t_condition(t_expression(t_var_name(a)),t_comp_operator(<),t_expression(t_integer(10))),t_block(t_statement_list(t_pr
int_expr(t_expression(t_var_name(a))),t_statement(t_assignment_expression(t_var_name(a),t_expression(t_add(t_var_name(a),
t_integer(1)))))))),t_statement_list(t_print_str("Testing Boolean expressions"),t_statement_list(t_var_decl_statement(t_v
ar_type(bool),t_var_name(isTrue),t_expression(t_boolean(true))),t_statement_list(t_var_decl_statement(t_var_type(bool),t_
var_name(isFalse),t_expression(t_boolean(false))),t_statement_list(t_print_str("isTrue"),t_statement_list(t_print_expr(t_
expression(t_var_name(isTrue))),t_statement_list(t_print_str("isFalse"),t_statement_list(t_print_expr(t_expression(t_var_
name(isFalse))),t_statement_list(t_print_str("Testing Ternary expressions"),t_statement_list(t_var_decl_statement(t_var_t
ype(int),t_var_name(T),t_expression(t_ternary_expression(t_condition(t_expression(t_integer(15)),t_comp_operator(>),t_exp
ression(t_integer(20))),t_expression(t_add(t_integer(15),t_integer(20))),t_expression(t_sub(t_integer(20),t_integer(15)))
))),t_statement(t_print_expr(t_expression(t_var_name(T)))))))))))))))))))))))))))))))))
Starting Evaluation
"Testing Print statements"
```

Final Output

```
Starting Evaluation
"Testing Print statements"
5
25
0.314
"Regular loop inside Enhanced loop"
1
2
3
4
5
1
2
3
4
5
1
2
3
4
5
"Testing While Loop"
3
4
5
6
7
8
9
"Testing Boolean expressions"
"isTrue"
true
"isFalse"
false
"Testing Ternary expressions"
5

Environment after evaluation
[(bool,isTrue,true),(bool,isFalse,false),(int,T,5)]
(base) gray@Heets-Air src %
```