SER502-Spring21-Team 10

IMPRO

Kamal Penmetcha (kpenmetc)

Karandeep Singh Grewal (kgrewal2)

Nikhil Hiremath (nhiremat)

Subramanian Arunachalam (saruna11)

Overview

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

About the language

About the language

Name: IMPRO

File extension: .imp

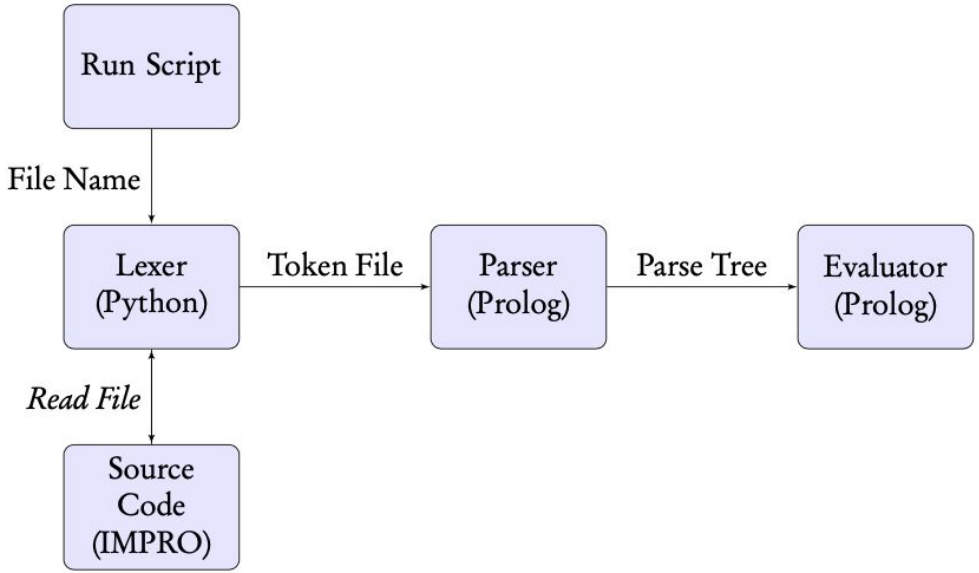
Programming Paradigm: Imperative

Programming languages used: Python(Lexer), Prolog(Parser and Evaluator)

Data Structures used: List

Design

Project pipeline



Project pipeline

XXX.IMP → LEXER.PY → XXX.IMPTOKENS → PARSER.PL → EVALUATOR.PL

Components of the design

1. **Run Script:** This will be a script that helps to run a IMPRO program file.

2. **Source Code:** This is the file that contains the code written in the developed language and needs to be executed.

3. **Lexer:** Lexer will accept the source code file (filename) as the input. It will break it into a list of tokens and will write to a new token file. The grammar of the developed language doesn't contain new line character. So we decided to use new line character as the separator for the tokens in the generated token file. The token file will contain each token put into a separate line (tokens are separated by a new line character).

4. **Parser:** This will read the file generated by the lexer and store the content of each line as separate element in a list. Then it will use the defined grammar rules to parse the list into a parse tree.

5. **Evaluator:** Evaluator will directly accept the parse tree generated by the parser and will start the evaluation process. It directly executes the instruction represented by the parse tree.

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

Non Terminals

```
% Start Symbol
program --> command_list.

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

command_list --> command, command_list.
command_list --> command_without_block, command_list.
command_list --> command.
command_list --> command_without_block.

% Single Line Commands
command_without_block --> print_command.
command_without_block --> assignment_command.
command_without_block --> variable_declaration_command.

% Multi Line Commands
command --> for_loop_command.
command --> while_loop_command.
command --> for_enhanced_command.
command --> if_command.
command --> if_elif_else_command.
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, [')'], block.
else_part --> ['else'], block.
elif_part --> ['elif'], ['('], condition, [')'], block.
elif_part --> ['elif'], ['('], condition, [')'], block, elif_command.

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

for_enhanced_command --> ['for'], variable_name, ['in'], ['range'], ['('], range_value, [','], range_value, [')'], block.

range_value --> variable_name | integer.

for_loop_command --> ['for'], ['('], assignment, [';'], condition, [';'], variable_change_part, [')'], block.

variable_change_part --> increment_expression.
variable_change_part --> decrement_expression.
variable_change_part --> variable_name, assignment_operator, expression.

condition --> expression, comparison_operators, expression.

decrement_expression --> variable_name, decrement_operator.
decrement_expression --> decrement_operator, variable_name.
increment_expression --> variable_name, increment_operator.
increment_expression --> increment_operator, variable_name.

print_command --> [print_string], ['('], string_value, [')'], end_of_command.
print_command --> [print_string], ['('], variable_name, [')'], end_of_command.
print_command --> [print_expression], ['('], expression, [')'], end_of_command.
```

Non Terminals

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

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

value --> float | integer | boolean_value | string_value | variable_name.

boolean_operators --> and_operator | or_operator | not_operator.

operators --> ['+'] | ['-'] | ['*'] | ['/'] | boolean_operators.

assignment_command --> variable_name, assignment_operator, expression,
    end_of_command.

variable_declaration_command --> variable_type, variable_name, end_of_command.
variable_declaration_command --> variable_type, variable_name, assignment_operator,
    expression, end_of_command.
```

```
variable_name --> lower_case, variable_name.
variable_name --> variable_name, upper_case.
variable_name --> variable_name, upper_case, variable_name.
variable_name --> variable_name, ['_'], variable_name.
variable_name --> lower_case.

string_value --> single_quote, character_phrase, single_quote.
string_value --> double_quote, character_phrase, double_quote.

character_phrase --> character, character_phrase.
character_phrase --> character.

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

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

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

Terminals

```
variable_type --> ['int'] | ['float'] | ['bool'] | ['string'].

decrement_operator --> ['--'].
increment_operator --> ['++'].

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

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_value --> ['True'].
boolean_value --> ['False'].

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

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

Snapshots

Running

```
Press ENTER or type command to continue
~/Git/SER502-Spring2021-Team10/src $ python Lexer.py test_for_enhanced.imp --evaluate
Starting Lexer
Reading Source Code: SUCCESSFUL
Writing Tokens in test_for_enhanced.imptokens: SUCCESSFUL

Starting Parser
```

Generating Parse Tree

```
Generating Parse Tree: SUCCESSFUL
t_program(t_command_list(t_variable_declaration_command(t_variable_type(int),t_variable_name(x)),t_command_list(t_assignment_expression(t_variable_name(x),t_e
xpression(t_integer(3))),t_command_list(t_variable_declaration_command(t_variable_type(int),t_variable_name(y),t_expression(t_divide(t_expression(t_add(t_inte
ger(4),t_integer(5))),t_integer(3)))),t_command_list(t_variable_declaration_command(t_variable_type(int),t_variable_name(z),t_expression(t_ad
(x),t_variable_name(y)))),t_command_list(t_variable_declaration_command(t_variable_type(int),t_variable_name(i),t_expression(t_integer(5))),t_command_list(t_v
ariable_declaration_command(t_variable_type(int),t_variable_name(j)),t_command_list(t_variable_declaration_command(t_variable_type(bool),t_variable_name(isBoo
lean),t_expression(t_boolean(true))),t_command_list(t_variable_declaration_command(t_variable_type(float),t_variable_name(number),t_expression(t_float(0.314))
),t_command_list(t_print_string("PRINT STATEMENT EXECUTED"),t_command_list(t_print_expression(t_expression(t_variable_name(x))),t_command_list(t_print_express
ion(t_expression(t_sub(t_integer(10),t_integer(5)))),t_command_list(t_print_string("FOR REGULAR inside FOR ENHANCED"),t_command_list(t_for_enhanced_command(t_
variable_name(i),t_expression(t_integer(1)),t_expression(t_integer(3)),t_block(t_command(t_for_loop_command(t_assignment_expression(t_variable_name(j),t_expre
ssion(t_integer(1))),t_condition(t_expression(t_variable_name(j)),t_comparison_operator(<),t_expression(t_integer(4))),t_post_increment(t_variable_name(j)),t_
block(t_command(t_print_expression(t_expression(t_variable_name(j))))))),t_command_list(t_variable_declaration_command(t_variable_type(int),t_variable_name(
a),t_expression(t_integer(7))),t_command_list(t_if_command(t_if(t_condition(t_expression(t_variable_name(a)),t_comparison_operator(==),t_expression(t_integer(
3))),t_block(t_command(t_print_string("Equal to 3")))),t_elif(t_condition(t_expression(t_variable_name(a)),t_comparison_operator(==),t_expression(t_integer(5)
)),t_block(t_command(t_print_string("Equal to 5"))),t_elif(t_condition(t_expression(t_variable_name(a)),t_comparison_operator(==),t_expression(t_integer(7))),
t_block(t_command(t_print_string("Equal to 7")))),t_else(t_block(t_command(t_print_string("I do not know"))))),t_command_list(t_print_string("WHILE LOOP"),t_
command_list(t_while_command(t_condition(t_expression(t_variable_name(a)),t_comparison_operator(<),t_expression(t_integer(10))),t_block(t_command_list(t_print
_expression(t_expression(t_variable_name(a))),t_command(t_assignment_expression(t_variable_name(a),t_expression(t_add(t_variable_name(a),t_integer(1)))))))),t
_command_list(t_print_string("BOOLEAN EXPRESSIONS"),t_command_list(t_variable_declaration_command(t_variable_type(bool),t_variable_name(isTrue),t_expression(t
_boolean_expression(t_expression(t_boolean(true)),t_boolean_operator(or),t_expression(t_boolean(false))))),t_command_list(t_variable_declaration_command(t_var
iable_type(bool),t_variable_name(isFalse),t_expression(t_boolean_expression(t_expression(t_boolean(true)),t_boolean_operator(and),t_expression(t_boolean(false
))))),t_command_list(t_print_string("isTrue"),t_command_list(t_print_expression(t_expression(t_variable_name(isTrue))),t_command_list(t_print_string("isFalse"
),t_command_list(t_print_expression(t_expression(t_variable_name(isFalse))),t_command_list(t_print_string("TERNARY EXPRESSION"),t_command_list(t_variable_decl
aration_command(t_variable_type(int),t_variable_name(pi),t_expression(t_ternary_expression(t_condition(t_expression(t_integer(1)),t_comparison_operator(>),t_e
xpression(t_integer(2))),t_expression(t_add(t_integer(3),t_integer(4))),t_expression(t_add(t_integer(5),t_integer(6)))))),t_command(t_print_expression(t_expre
ssion(t_variable_name(pi))))))))))))))))))))))))))))))))
Starting Evaluation
```

IMPRO SOURCE CODE

```
# ASSIGNMENTS AND DECLARATIONS
int x;
x = 3;
int y = (4 + 5)/3;
int z = x + y;
int i = 5;
int j;
bool isBoolean = true;
float number = 0.314;

# PRINT COMMAND
print_string("PRINT STATEMENT EXECUTED");
print_expression(x);
print_expression(10-5);

# FOR REGULAR inside FOR ENHANCED
print_string("FOR REGULAR inside FOR ENHANCED");
for i in range(1;3) {
    for(j=1; j<4; j++) {
        print_expression(j);
    }
}

# IF ELSE
int a = 7;
if( a == 3){
    print_string("Equal to 3");
}
elif(a == 5){
    print_string("Equal to 5");
}
elif(a == 7){
test_program.imp" 58L, 1042B
```

EVALUATION

```
aration_command(t_variable_type(int),t_variable_name(pi),t_ex
xpression(t_integer(2))),t_expression(t_add(t_integer(3),t_in
ssion(t_variable_name(pi))))))))))))))))))))))))))))))))))))
Starting Evaluation
"PRINT STATEMENT EXECUTED"
3
5
"FOR REGULAR inside FOR ENHANCED"
1
2
3
1
2
3
1
2
3
"Equal to 7"
"WHILE LOOP"
7
8
9
"BOOLEAN EXPRESSIONS"
"isTrue"
true
"isFalse"
false
"TERNARY EXPRESSION"
11

Environment after evaluation
[(bool,isTrue,true),(bool,isFalse,false),(int,pi,11)]
?-
```

Future Scope

Other potential features

- Multiple variable declaration in a single line.
  int a, b, c;

- Variable assignment in the same command as variable declaration.
  ```
  int a=10, b=20;
  int a=b=c=10;
  ```

- String version for enhanced for loop.
  ```
  for character in string("Hello World"){
      print(character);
  }
  ```

- List Data Structure
  ```
  x = [1, 2, "hello", 2.32];
  ```