Milestone 1 - SER502

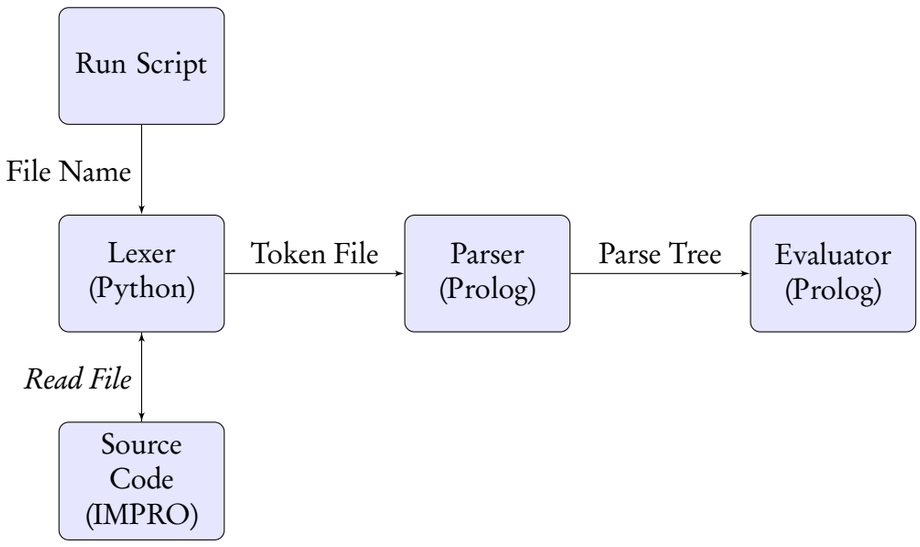
Team 10
Kamal Penmetcha
Karandeep Singh Grewal
Nikhil Hiremath
Subramanian Arunachalam

March 31, 2021

1 About Developed Language

Name	IMPRO
File Extension	.imp
Programming Paradigm	Imperative
Github Repository	github.com/kgrewal2/SER502-Spring2021-Team10

2 Project Pipeline



2.1 Run Script

This will be a script that helps to run a IMPRO program file.

2.2 Source Code

This is the file that contains the code written in the developed language and needs to be executed.

2.3 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).

2.4 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.

2.5 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.

3 Programming Languages Used

Component	Programming Language
Lexer	Python - SLY (Sly Lex Yacc) (Subject to change)
Parser	Prolog
Evaluator	Prolog

4 Data Structures

Data structure that will be used in the python lexer will be **LIST** and save it as a token file. This file is read by prolog parser and converted into a **LIST**. The parser then generates a parsetree **LIST** as the output. This output is accepted by the evaluator component and directly run on the machine. All the variables of the program executions are also stored as a prolog **LIST**.

5 Parsing Technique

Recursive Descent Parsing is a top-down parsing technique that turns all the non-terminals into a group of mutually recursive procedures based on the right hand side of the grammar rules. The rules of the parse are written in DCG.

6 Design

We aim to keep the syntax of our developed programming language to be a combination of C and Python. Example: We have taken the concept of braces from C whereas concept of a simpler print statement from Python. Different levels of grammar details are discussed in the following subsections.

6.1 Program

This is the top level unit of the code that will be written in our developed programming language. Any file that will be executed in our developed language will have the `Program` as the main structure. `Program` consists of command list.

6.2 Commands and Command List

Commands are the building blocks for the `Program`. Commands are of two different types: Commands without block & Commands with block. The term `block` will be discussed later. A set of one or more commands is called a command list. Each command without a block should end with a semi-colon symbol.

```
print("Hello World");
int i = a;
```

6.3 Block

A set of one or more commands enclosed inside the curly brackets is called a block.

```
{
    // commands
}
```

6.4 Commands with block

6.4.1 For Loop

```
for(int i=0; i<10; i++){
    // commands
}
```

6.4.2 While Loop

```
while(i<10){
    // commands
}
```

6.4.3 For Loop (Enhanced)

```
for i in range(1, 10){
    // commands
}
```

6.4.4 If

```
if(i==10){
    print("Ten");
}
```

6.4.5 If Elif Else

```
if(i==10){
    print("Ten");
}
elif(i==1){
    print("One")
}
else{
    print("Not ten or zero")
}
```

6.4.6 If Else

```
if(i==10){
    print("Ten");
}
else{
    print("Not Ten");
}
```

6.5 Commands without block

6.5.1 Print

```
print("Your age is " + 10 + " years old");
```

6.5.2 Variable Declaration

```
int x;
```

6.5.3 Variable Assignment

Uses equal to symbol for assignment operation

```
x = 10;
```

6.6 Variable Naming

- Variable name can only start with lower case letter.

```
variable
```

- Variable name can contain lower case or upper case letter.

```
calculationAPI
calculationModule
```

- Variable name can contain underscores.

```
calculation_API
calculation_result
```

6.7 Data Types

6.7.1 Integer

```
int x;
x = 23;
```

6.7.2 Floating Point Numbers

```
float x;
x = 100.23;
```

6.7.3 String

```
string x;
string y;
x = ‘hello’;
y = "there";
```

6.7.4 Boolean

```
bool x;
x = True;
x = False;
```

6.8 Operations

6.8.1 Addition

Uses plus symbol

```
x = 1 + 2 + a;
```

6.8.2 Subtraction

Uses minus symbol

```
x = 1 - 2 - a;
```

6.8.3 Multiplication

Uses star symbol

```
x = 1 * 2 * a;
```

6.8.4 Division

Uses slash symbol

```
x = 1 / 2;
```

6.8.5 Braces

Uses braces' symbols

```
x = 1 + (2 - 3 * (3 / 6));
```

6.8.6 Ternary Operator

Uses question mark symbol and colon symbol

```
int x = (i == 10) ? i + 10 : i - 10;
```

6.9 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
```

6.10 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];
```

7 Contribution

Link to `contribution.txt`
https://github.com/kgrewal2/SER502-Spring2021-Team10/blob/main/contribution.txt

Milestone 1 - Design Document - Done by the whole team through various online meetings.

	Project Level Future Contribution
Nikhil Hiremath	Lexer
Kamal Penmetcha	Parser
Karandeep Singh	Evaluator
Subramanian	Testing Scripts

8 Grammar

Notation Used: Prolog DCG
The implementation for the non-terminals like `number/2` will be done using prolog built-in predicates.

```
%%%%%%%%%%%%%%%%
% 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'], ['('], condition, [')'], 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'], ['('], expression, [')'], end_of_command.

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'].
```