SER 502

Emerging Programming Languages and Paradigms (Spring 2020) Project Phase 1 – Language

Name: Trace

Akhilesh Krishnan (akrish84)

Rahul Suresh(rahul0717)

Swarnalatha Srenigarajan (Swar-jain)

Idhant Haldankar(idhant96)

Description:

Program begins with the keyword “execute”.

Itisfollowedbyablock.Ablockstartsandendswithcurlybracesandcomprisesdeclarations, statements.

The user can declare variables of type: Number, Boolean or string.

Data types supported in our language:

Number: Comprises integer or floating point number. Default Value: 0

Boolean: true or false. Default value: false

String: Itisasequenceofcharactersenclosedwithindoublequotes.Defaultvalue:“”(Empty)

Arithmetic Operators:

1. Modulus: %
2. Multiplication: *
3. Division: /
4. Addition: +

5. Subtraction: -
6. Increment: ++
7. Decrement: --

Arithmetic OperatorPrecedence:

1. Parenthesis: ()
2. Modulus, Multiplication, Division: %, *, /
3. Addition, Subtraction: +,-

Relational Operators:

1. Lesser : “<”
2. Greater : “>”
3. Less than or equal to : “<=”
4. Greater than or equal to : “>=”
5. Not equal to : “! = “
6. Equality check :“==”

Logical Operators:

1. and
2. or
3. Not

Assignment Operator:

1. Equals: “=”

Ternaryoperator:

Thisoperatorrequires**3operands**andcanbeusedasareplacementforifthenelsethen statements.

Example for ternary operator :

a = false;

a ? print(“true”) : print(“false”);

Declarations:

A block can have zero or more number of declaration statements. Our program is strongly typed and data type has to be explicitly mentioned. Have a signal declaration in a line. Every declaration should end with semicolon “;”

Example of declaration.

1. number a =10;
2. number a;
3. bool isValid =true;
4. string name;

Conditional Statement

The “if (condition) then” statement executes a block if the condition evaluates to true. If the condition evaluates to false, the block following the “else then” gets executed. This is used to
execute different blocks based on different conditions. The keyword “else if (condition) then” is used to check for another condition when the “if (condition) then” condition evaluates to false.

Examples of conditionals: if (x ==

2) then { x = x + 2 } else then { x = x +

1}

if (x == 2) then {x = x + 2}

else if (x == 3) then { x = x + 1 } else then { x

= x + 3}

```
if (x == 2) then { x = x + 2 }
```

Statement Lists:

Statement Lists is the part which follows Block. Statement List part can contain one or more of many of the types of statements.

Different kinds of statements: Print statement, Assignment statements, Loop Statement List, Conditional Statement List.

Print statement:

Theprintstatementisusedtowritetothescreen.Theprintstatementsstartwitha'print' keywordandthenfollowed byastring,booleanexpressionornumberexpressiontobeprinted inside '()'.

Example: print("Hello World!")

Loop statement:

Simple for pattern

Aloopstatementstartswith"for"followedbyparenthesis().Insidetheparentheseswehave threepartswhichare namely"declaration","conditionalexpression"and"increment statement".Indeclarationwewoulddeclaretheiterablevariable,laterthebool_exprwill evaluatetheiteration limitfortheloopandlastlytheupdatestatementwillupdatetheiterable variable.

Examples of Simple for pattern :

```
for(i=0;i<10;i=i+1;){i = i+1;

}
```

Range pattern

A loop statement starts with "for" followed by iterable variable initialization.In declaration we would declare the iterable variable, and range of iterable values the iterable variable can take. The range function generates a list of iterablevalueswhichtheiterablevariablecantakein each iteration.

Examples of Simple for pattern :

```
For i in range(1,10) { i = i+

1;

}
```

Simple While Loop

The while loop will begin with the "while" keyword followed by parenthesis (). Inside the parentheses the bool_expr, which gets executed each time the while loop is called. The loop continues until bool_expr return true and ends when it returns false.

Examples of Simple for pattern :

```
while (i < 10){ i =

i+ 1;

}
```

Assignment Statement:

This statement is used to set a value given to a variable. Examples of

assignment statement :

1. X = 5;
2. X = "HelloWorld!";
3. X = "true";

Example Program:

```
execute     {

number a;

bool isValid = false;
string name = “team22”;

if(name   ==   “team22”   )   then   {

print(“team 22”);

} else then {

print(“not eam 22”);

}

for(numberi=0;i<22;i++){ a =i;

print(“hi”);

}

a = (20 % 5) + 2;

isValid       =       true;

while(isValid)  {  int  b

= 10;

print(“isValid”);

isValid = false;

}

print(“This is a sample program”);

}
```

Grammar

```
program -> [execute], block.

block -> ['{'], declarationList, statementList, ['}'].

/* Declaration part */
 %------
declarationList -> declaration, [';'], declarationList.
declarationList -> [].

 %------
declaration -> numberDeclaration.
declaration -> booleanDeclaration.
declaration -> stringDeclaration.

 %------
numberDeclaration -> [number], var_name, ['='], number_expr.
numberDeclaration -> [number], var_name.

 %------
booleanDeclaration -> [bool], var_name, ['='], bool_expr.
booleanDeclaration -> [bool], var_name.

%------
stringDeclaration -> [string], var_name, ['='], string_expr.
stringDeclaration -> [string], var_name.

%-----
/* Statements part */
statementList -> statement, [';'], statementList.
statementList -> loopStatement, statementList.
statementList -> conditionalStatement, statementList.
statementList -> [].

%-----
statement -> print_statement.
statement -> var_name, ['='], number_expr.
statement -> var_name, ['='], bool_expr.
statement -> var_name, ['='], string_expr.
statement -> var_name, ['+'], ['+'].
statement -> var_name, ['-'], ['-'].

%---
print_statement -> [print],['('], string_expr, [')'].
print_statement -> [println],['('], string_expr, [')'].

%---
/* Looping Statements -> (for loop, for in range() loop and while loop ) */
loopStatement -> [for], ['('], declaration, [';'], bool_expr , [';'], statement, [')'], block.
loopStatement -> [for], var_name, [in], [range], ['('], number, [','], number ,[')'], block.
loopStatement -> [while],['('], bool_expr ,[')'], block.

/* Conditional Statements -> (empty, if then, if then else, if then else if... ) */
conditionalStatement -> [if], ['('], bool_expr , [')'], [then], block.
conditionalStatement -> [if], ['('], bool_expr , [')'], [then], block , [else], [then], block.
conditionalStatement -> [if], ['('], bool_expr , [')'], [then], block , [else], conditionalStatement.
conditionalStatement -> bool_expr, ['?'], statement, [':'], statement.

number_expr-> [len],['('],string_expr,[')'].
number_expr -> number_expr, ['+'], level_1.
number_expr -> number_expr, ['-'], level_1.
number_expr -> level_1.

level_1 -> level_1, ['*'], level_2.
level_1 -> level_1, ['/'], level_2.
level_1 -> level_1, ['%'], level_2.
level_1 -> level_2.

level_2 -> ['('], number_expr, [')'].
level_2 -> number.
level_2  -> var_name.


/* Boolean Expression -> (true, false, not, and ,or, ==, !=, >, <, >=, <=) */

bool_expr -> [not], bool_expr.
bool_expr -> bool_expr, [and] , bool_expr.
bool_expr-> bool_expr, [or] , bool_expr.
bool_expr -> number_expr, ['='], ['='], number_expr.
bool_expr -> number_expr, ['!'], ['='], number_expr.
bool_expr -> number_expr, ['>'], number_expr.
bool_expr -> number_expr, ['<'], number_expr.
bool_expr -> number_expr, ['>'], ['='], number_expr.
bool_expr -> number_expr, ['<'], ['='], number_expr.
bool_expr -> string_expr, ['='], ['='], string_expr.
bool_expr -> string_expr, ['!'], ['='], string_expr.
bool_expr -> bool_expr, ['='], ['='], bool_expr.
bool_expr -> bool_expr, ['!'], ['='], bool_expr.
bool_expr  -> var_name.
bool_expr -> ['true'].
bool_expr -> ['false'].
```

```
/* String Expression -> checks for string type.*/
string_expr -> string.
string_expr -> var_name.
string_expr -> [str],['('], number_expr, [')'].
string_expr -> [str],['('], bool_expr, [')'].
string_expr -> string_expr, ['+'], string_expr.


var_name -> X, {atom(X)}.

/* primitive types */
number -> X, {number(X)}.
string -> X, {string(X)}.
```

Parsing Technique:

We will be using python to parse our program. Using ply library we will tokenize the input. Our grammar will be written in DCG. We will use pylog library to run prolog code. We will use DCG to generate our abstract parse tree and evaluate it using prolog. The final output will be displayed by python. Since we are using python our program will run on python's runtime environment. The tokenized result will be stored in a list and passed to DCG.

Future Work:

1. Adding support for functions
2. Including switch conditionalstatement.