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”.

It is followed by a block. A block starts and ends with curly braces and comprises declarations, 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: It is a sequence of characters enclosed within double quotes. Default value: “” (Empty)

Arithmetic Operators:

1. Modulus: %
2. Multiplication: *
3. Division: /
4. Addition: +
5. Subtraction: -
6. Increment: ++
7. Decrement: --

Arithmetic Operator Precedence:

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: “=”

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:

The print statement is used to write to the screen. The print statements start with a ‘print’ keyword and then followed by the string to be printed inside ‘()’.

Example: print(“Hello World!”)

Loop statement:

Simple for pattern

A loop statement starts with “for” followed by parenthesis (). Inside the parentheses we have three parts which are namely “declaration”, “conditional expression” and “increment statement”. In declaration we would declare the iterable variable, later the bool_expr will evaluate the iteration limit for the loop and lastly the update statement will update the iterable 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 iterable values which the iterable variable can take in 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 = “Hello World!”;
3. X = “true”;
4.

Boolean expression:

The boolean expression can simply evaluate to be a ‘true’ or ‘false’ value. ‘not’ boolean operator: This operator is used to negate the value of a boolean expression. Syntax : ‘not’ keyword followed by a boolean expression.

‘and’ boolean operator: This operator is used to check whether both value of the two boolean expression evaluates to ‘true’ Syntax : boolean expression1 followed by ‘and’ keyword followed by a boolean expression2.

‘Or’ boolean operator: This operator is used to check whether either value of the two boolean expression evaluates to ‘true’ Syntax : boolean expression1 followed by ‘or’ keyword followed by a boolean expression2.

‘equality’ boolean operator: This operator is used to check whether the values of two expressions(numeric/boolean/string) are equal. Syntax : expression1 followed by ‘ ==’ keyword followed by expression2.

‘!=’(not equal) boolean operator: This operator is used to check whether the values of two expressions(numeric/boolean/string) are not equal. Syntax : expression1 followed by ‘ !=’ keyword followed by expression2.

‘>’(greater than) boolean operator: This operator is used to check whether the value of an expression1(numeric/boolean/string) is greater than expression2. Syntax : expression1 followed by ‘ >’ keyword followed by expression2.

‘<’(lesser than) boolean operator: This operator is used to check whether the value of an expression1(numeric/boolean/string) is lesser than expression2. Syntax : expression1 followed by ‘<’ keyword followed by expression2.

‘>=’(greater than or equal to) boolean operator: This operator is used to check whether the value of an expression1(numeric/boolean/string) is greater than or equal to expression2. Syntax : expression1 followed by ‘ >=’ keyword followed by expression2.

‘<=’(lesser than or equal to) boolean operator: This operator is used to check whether the value of an expression1(numeric/boolean/string) is lesser than or equal to expression2. Syntax : expression1 followed by ‘<=’ keyword followed by expression2.

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( number i = 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 }

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

statementList -> statement ; 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 --

statement -> loopStatement

statement -> conditionalStatement

print_statement -> print(string_expr)

loopStatement -> for (declaration ; bool_expr ; statement) block

loopStatement -> for var_name in range (number number) block

loopStatement -> while(bool_expr) block

conditionalStatement -> if (bool_expr) then block

conditionalStatement -> if (bool_expr) then block else then block

conditionalStatement -> if (bool_expr) then block else conditionalStatement

number_expr -> level_1

number_expr -> number_expr + level_1

number_expr -> number_expr - level_1

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

bool_expr -> true

bool_expr -> false

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 -> bool_expr = = bool_expr

bool_expr -> string_expr = = string_expr

bool_expr -> number_expr ! = number_expr

bool_expr -> bool_expr ! = bool_expr

bool_expr -> string_expr ! = string_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

string_expr -> string

var_name -> X {atom(X)}

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.