Language Design and Functionalities Report

A. Introduction

Our language name is called “Hash Code”. The language is designed to support object-oriented programming. Since hashtags in social media are widely-used, thanks to Twitter, we decided to implement this style in our syntax.

B. Grammar Definition

1. Identifiers

- The identifiers can either be variables or constants
- A variable begins with a letter (either upper or lowercase) and are alphanumeric
- A constant may or may not start with a sign (+ or -)

<id> → <var> | <constant>
<var> → <var><alphanum> | <letter>
<alphanum> → <letter> | <digit>
<letter> → A | B | … | Z | a | b | … | z
<digit> → 0 | 1 | 2 | … | 9
<constant> → <sign><intlit>
<intlit> → <intlit><digit> | <digit>
<sign> → epsilon | + | -

2. Data types

- The language supports two data types, integer and float
- Integers are constants
- Floats are decimal constants (must have a decimal point)

<declare> → @int <var><init> | @float <var><init1>
<init> → epsilon | =<constant>
<init1> → epsilon | =<constant>.<intlit>

3. Expressions and Assignment Statements

- Assignment of expressions to a variable is allowed
- Expressions allow the use of +, -, *, /, and % operators
- The use of parentheses is also allowed

<assign> → <var> = <expr>
<expr> → <expr> <oper1> <term> | <term>
<oper1> → + | -
<term> → <term> <oper2> <factor> | <factor>
<oper2> → * | / | %
<factor> → (<expr>) | <id>

4. Statement Level Control Structures

- The language supports if-elif-else and for control structures
- Conditions for both control structures allow the following relational operators: ==, !=, >, <, >=, <=

- If-elif-else control structure:
  <if> → #if (<cond>)'\n' <block><elif><else> #end
  <elif> → #elif (<cond>) <block>'\n'<elif> | epsilon
  <else> → #else <block> | epsilon

- For control structure:
  <for> → #for (<var>=<id>; <cond>; <var><iter>)'\n'
  <block>
  #end
  <iter> → ++ | --

  Wherein:
  <cond> → <expr> <rel> <expr>
  <rel> → == | != | > | < | >= | <=

5. **Subprograms**
   - The language requires at least one function (which will be your “main” function)
   - Each function requires at least one variable declaration before any other blocks of code

   - Main function:
     <program> → #startprogram'\n'<block1>#endprogram'\n'<outsideprog>
     <outsideprog> → <func>'\n'<outsideprog> | epsilon

   - Subprogram:
     <func> → #function <var> (<param>)'\n' <block1> #end
     <param> → <param1> | epsilon
     <param1> → <id>,<param1>|<id>

   - Calling a function:
     <call> → #call <var>(<param>) #end

     Wherein:
     <block1> → <declare>'\n'<block>
     <block> → <function>'\n'<block> | <function>'\n'
     <function> → <print> | <read> | <for> | <if> | <assign> | <call> | ?<comment>?
     <print> → #print "<strexpr>" #end
     <strexpr> → #{<id>} | <comment>
     <read> → #read <var> #end
     <comment> → <var><space><comment> | epsilon

C. Lexical and Syntax Analysis

- LR parsing was used.

D. Names, Binding, and Scoping

- Names are case sensitive
- Reserved words are completely dependent on the syntax of Python (e.g. ‘def’ is not a valid variable name as it is used for function declaration).
- Keywords:
  1. #startprogram → Start of the main function
  2. #endprogram → End of the end function

3. #if → Start of an if block. Conditional.
4. #elif → Start of an elif block. Conditional.
5. #else → Start of an else block. Conditional.
6. #print → Used for displaying text.
7. #read → Used for reading input.
8. #for → Used for iteration/loops.
9. #call → Used for calling functions.
10. #function → Used for declaring functions.
11. #end → This signifies the end of a block. Almost all of the keywords are used to 'start' something. For the parser to recognize that a block is finished, it has to have a #end at the end. All of the keywords above need this keyword (in a new line xor inline) to be parsed correctly except for #startprogram, #endprogram, #elif, and #else.

- Name form: Variables are alphanumeric but should start with a letter. Both uppercase and lowercase letters can be used.
- Binding type: The language has an explicit, static type binding.
    1. Declaration of integers: @int varname12
    2. Declaration of floats: @float varname1415
    3. Declarations may or may not provide initialization of variables (e.g. @int varname12 = 5)
- Lifetime and scoping: The identifiers have static lifetime and scoping.
- Block representation: Functions are blocks that start and end with special keywords. Blocks after the initialization of control structures are detected when an '#end' keyword does not follow the initialization. Indentation is not required.

E. Data Types

- Only integers and floats are available as primitive data types. Strings are not supported.
- Our language does not support strings or any other user-defined data types. Coercion is supported while typecasting is not, since our base language is python.
- User-defined data types are not supported.

F. Expression and Assignment Statements

- PMDAS operations and grouping (through parentheses) are supported. Arithmetic expressions support infix notation.
- There is no operator overloading.
- Coercion is supported (e.g. int + float = float).
- The following relational operators are supported:
    a. Greater than (>)
    b. Greater than or equal to (>=)
    c. Less than (<)
    d. Less than or equal to (<=)
    e. Equal to (==)
    f. Not equal to (!=)
- Conditions: <expr> <relational operator> <expr>
- Assignment operator used is the equal sign (=). You can assign a variable to an expression: <assign> → <id> = <expr>

G. Statement-Level Control Structures

1. Selection statements: if-elif-else control structure
    - Supports nested if statements (multiple ifs)

- Elif and else blocks are optional
- Nested if sample syntax:

```
#if (cond)                                        → start of if block
        #if (cond)                                → start of nested if block
                #print "string" #end
        #end                                      → end of nested if block
        #print "more string" #end
#elif (cond)
        #print "striiinggg" #end
#else
        #print "yay" #end
#end                                              → end of if block
```

2. Iterative statements: for loop control structure
    - Similar to C for loop syntax
    - Allows counter and logically controlled loops
    - Allows nested iterations
    - For syntax:

```
#for (<var>=<id>; <cond>; <var><iter>)
        <block>
#end
```

H. Subprograms

- Subprogram definition:

#function functionname (parameter, parameter1)
block of code
#end

#function - indicates start of subprogram
functionname - variable; must start with a letter; alphanumeric; supports uppercase and lowercase letters
parameters - may be variables or integers
- does not include declaration of parameter data types and default initialization
- function can be defined with or without parameters
- parameters are separated with a comma
#end - indicates end of subprogram

- Calling subprograms:

#call functionname (parameter, parameter1) **#end**

#call - indicates start of calling a subprogram
#functionname - function to be called (variable)
#parameters - parameters to be passed
- must pass same number of parameters as defined in the function
#end - indicates end of calling a subprogram

- Language does not support subprogram returns.
- Global variables are not supported since the language is based on Python.

- Local variables within subprograms have local scope and are allocated within the subprogram.
- Subprograms can be defined before or after they are called. They are defined after the main function.

```
#startprogram
        Main function
#endprogram

#function functionname1 ()
        Subprogram1
#end

#function functionname2 (var)
        Subprogram2
#end
```

- Parameters are implicitly stated (no data types included in subprogram definition).
- Parameter format: (var1, var2, int1, var3, int2).
- Subprograms may or may not include parameters.
- Subprograms may include multiple parameters as well.
- The language does not support recursions.