SimplCalc

Snir Levi, Rotem Marmari, Ido Hod

Scripting Language: SimplCalc

Syntax

- **Expressions:** Expressions are evaluated to produce a single integer value.
- **Numbers:** Integers are represented as literals, e.g., 10, -5.
- **Operators:**
  - Arithmetic operators: +, -, *, / (integer division)
  - Comparison operator: = (equal to)
- **End of command** is marked by ';'.

Data Types

- **Integer:** The only data type supported is the integer.

Control Flow (Limited)

This language has limited control flow capabilities. It only supports defining variables and simple expressions.

Interpreter Design

The interpreter can be implemented in various ways (Python, Java, etc.). Here's a high-level overview:

1. **Lexical Analysis:** Break the script into tokens (numbers, operators, parentheses, keywords).
2. **Syntax Analysis:** Verify the syntax of the script (correct function calls, parentheses matching).
3. **Evaluation:**
   - Numbers are pushed onto a stack.
   - Operators pop operands from the stack, perform the operation, and push the result back.
   - Function calls involve evaluating arguments first, then applying the function logic.
   - The define function stores the evaluated expression result in a symbol table with the variable name as the key.
   - The = operator compares two operands (pops from the stack) and pushes 1 for true or 0 for false.
4. **Error Handling:** Handle syntax errors and runtime errors (e.g., division by zero).

Example Evaluation

Let's evaluate the expression (+ (* 2 4) (- 4 6)):

1. Evaluate (* 2 4): Push 8 onto the stack.
2. Evaluate (- 4 6): Push -2 onto the stack.
3. Evaluate (+ 8 -2): Pop 8 and -2, add them, and push 6 onto the stack.
4. Final result: 6

This interpreter provides a basic framework for performing simple integer arithmetic with variables.

Miscellaneous

Added a print function.
Syntax: (print (<something to print>));

BNF grammar for SimplCalc:

<program> ::=	<statement> | <program> <statement>	Marked text was added on Section 5 Section 6 Section 7
<statement> ::=	<expression> | <assignment> | <conditional> | <loop> | <print> | <comment>	

<expression> ::=	<number> | <variable>
	| (<operator> <expression>+);
	| (<comparison> <expression> <expression>);	

<number> ::=	[+-]? <digit>+	
<digit> ::=	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9	
<variable> ::=	<identifier>	
<identifier> ::=	[a-zA-Z_][a-zA-Z0-9_]*	
<operator> ::=	+ | - | * | /	
<comparison> ::=	= | < | >	
<assignment> ::=	(define <identifier> <expression>);	
<conditional> ::=	(if <expression> : <statement>);	
<loop> ::=	(while <expression> : <statement>);	
<print> ::=	(print (<print_list>));	
<print_list> ::=	"<text>" | <expression> | <print_list> , <print_list>	
<comment> ::=	# <text>	
<text> ::=	any sequence of characters	

- <program>: A program is one statement or a sequence of statements.
- <statement>: A statement can be an expression to be evaluated, a variable assignment, a conditional or a loop.
- <expression>: An expression can be a number, a variable, an operator applied to one or more expressions within parentheses, or a comparison between two expressions.
- <number>: A number is an optional sign followed by one or more digits.
- <digit>: A digit is any character from 0 to 9.
- <variable>: A variable is an identifier starting with a letter or underscore, followed by letters, numbers, or underscores.
- <identifier>: Defines the valid format for variable names.

- <operator>: Defines the supported arithmetic operators.
- <comparison>: Defines the supported comparison operators (=, <, >).
- <assignment>: Defines the syntax for assigning an expression result to a variable using the define keyword.
- <conditional>: Defines the syntax of an if statement.
- <loop>: Defines the syntax of a while loop.
- <print>: Defines the syntax of a print statement.
- <print_list>: Defines the syntax of a print statement's content. The syntax allows the user to print multiple texts and expressions within the same print statement.
- <comment>: Defines the syntax of a comment.

Memory Management

These are the limitations we chose to enforce:

```
self.max_result_length = 50      # calculation result length
self.max_program_length = 1000   # script length
self.max_exp_length = 50         # expression length
self.max_variable_length = 10
self.max_variables_allowed = 10
```

Statically Typed or Dynamically Typed?

Even though we assume that all the variables are integers, dynamic typing refers to how variable types are determined and enforced, not necessarily to the specific types of variables used in a language. The fact that variable types are determined at runtime based on the assigned values still classifies it as dynamically typed.
In SimplCalc, you don't explicitly declare the type of a variable, and the interpreter dynamically determines the type of a variable based on the value assigned to it. This characteristic aligns with the definition of dynamic typing, regardless of the specific type (in this case, integers) assumed for variables in the language.