HULK Interpreter documentation

Eduardo Brito Labrada

October 5, 2023

Abstract

In computer science, an **interpreter** is a computer program that directly executes intructions written in a programming or scripting language, without requiring them previously to have been compiled into a machine language program. An interpreter generally uses one of the following strategies for program execution:

1. Parse the source code and perform behavior directly;
2. Translate source code into some efficient intermediate representation or object code and inmediately execute that;
3. Explicitly execute stored precompiled bytecode made by a compiler and matched with the interpreter Virtual Machine.

In this project we will focus on the first strategies of those to create a interpreter for *Havana University Language for Kompilers (HULK)*. First we will define the basic syntax of the language and then we will show how the interpreter works in its entirety.

Contents

Introduction

HULK is a didactic, type-safe, object-oriented and incremental programming language. This is a simplified version of HULK where we will be implementing a subset of this programming language. In particular, this subset consists only of expressions that can be written on one line.

Expressions

HULK is a ultimately an expression-based language. Most of the syntactic constructions in HULK are expressions, including the body of all functions, loops and other block of code.

The body of a program in HULK always end with a single global expression (and, if necessary, a final semicolon[1]) that serves as the entrypoint of the program.

Arithmetic expressions

HULK defines three types of literal values: **numbers**, **strings** and **booleans**. Numbers are 32-bit floating-point and support all basic arithmetic operations with the usual semantic: `+` (addition), `-` (subtraction), `*` (multiplication), `/` (floating-point division), `^` (power), and parenthesized sub-expressions.

Strings

Strings literals in HULK are defined within enclosed double-quotes (`"`). A double-quote can be included literally by escaping it (`\"`), and other escaped characters are `\n` for line endings and `\t` for tabs. Also, strings can be concatenated with other strings (or the string representation of numbers) using the `@` operator.

Builtin math functions and constants

Besides `print`, HULK also provides some common mathematical operations encapsulated as builtin functions with their usual semantic. The list of builtin math functions is the following:

- `sqrt(<value>)` computes the square root if a value.
- `sin<angle>` computes the sine of an angle in radians.
- `cos(<angle>)` computes the cosine of an angle in radians.
- `exp(<value>)` computes the value of `e` raised to a value.
- `log(<base>, <value>)` computes the logarithm of a value in a given base.
- `rand()` returns a random uniform number between 0 and 1 (both inclusive).

[1] In this version of HULK all expressions end with a single semicolon

Besides these functions, HULK also provides two globals constants: `PI` and `E` wich represent the floating-point value of these mathematical constants.

As expected, functions can be nested in HULK (provided the use of types is consistent, but so far all we care about is functions from numbers to numbers, so we can forget about types until later on).

More formally, function invocation is also an expression in HULK, so everywhere you expected an expression you can also put a call to builtin function, and you can freely mix arithmetic expressions and mathematical functions, as you would expected in any programming language.

Functions

HULK also lets you define your own functions (of course!). A program in HULK can have an arbitrary number of functions defined before the final global expressions.

A function's body is always an expression, hence all functions have a return value (and type), that is, the return value (and type) of its body.

Inline functions

The easiest way of define a function is the inline form. An inline function is defined by an identifier followed by arguments between parenthesis, then the `=>` symbol, and then a simple expression (not an expression body) as body, ending with a final semicolon.

In HULK, all functions must be defined before the final global expression. All these functions live in a single global namespace, hence is not allowed to repeat function names. Similarly, there are no overloads in HULK (at least in "basic" HULK).

Finally, the body of any function can use other functions, regardless of whether they are defined before or after the corresponding function.

And of course, inline functions (and any other type of functions) can call themselves recursively.