Implementation description

Jiří Škrobánek[1]

December 26, 2019, Ostrava

Introduction

This document briefly describes the usage and implementation of parser and interpreter for scripting language specified in corresponding Language Specification document. Both parser and interpreter are implemented in C# 8.0 and .NET Core, the latest version uses .NET Core 3.1. For more information about particular features comments in the source code may be seen.
This project is maintained at `https://github.com/jiri-skrobanek/scri-lang.git`.

Parser

The purpose of the parser is to convert textual representation of a program into internal binary structure. The format of valid textual representation is described in language specification.
The process of parsing begins by tokenization of the code. There are various types of tokens which determine which kind of statement or expression should be matched.
Once tokenization is complete, the list of tokens is split into statements and a statement object is created for each statement. This is done recursively for function bodies, loops, and conditional branching.
Expressions are created recursively, operators are found by their priority in descending order ending with function calls. Once all operators of the highest priority are found, should their arguments still be composed expressions, operators of lower priority are found, this process is repeated for all priority levels.
The conversion from text to binary representation is linear in time complexity with the length of the text. This is possible thanks to the language having context-free grammar.

Interpreter

The interpreter receives binary representation of a program (either produced by the parser or generated programmatically) and executes it.
Classes exist for features of the language such as statements, expressions, values. Subtypes of these features have their own descendant classes. With this design, the language should be easily extensible, exempli gratia floating point arithmetic could be added if needed.
The program is executed inside an environment which provides special global built-in functions and variables. Built-ins can be defined inside the environment as .NET methods and fields, if annotated by Built-in attributes, these items are exported into the global scope of the interpreter.

[1]Faculty of Mathematics and Physics, Charles University, Prague, `jiri@skrobanek.cz`