SimPL-FT - A simple programming language with Functions and Types

Carson Quigley

December 2024

Introduction

We proposed the Simplft programming language: A simple interpreted programming language that implements primitive data structures, functions, and types. The language is built on the Java programming language, making use of Java's type system to allow type casting objects for the visitor pattern. Java's type system also comes in handy when implementing types, as we can simply check the literal expression's value type within Java's type system to determine the primitive instead of parsing the string again. The result is a working, interpreted programming language that supports FLOAT, STRING, CHAR, and NULL types.

Methods

Foundation

The Simplft programming language aims to be an interpreted programming language that implements functions and types. To this end, we make use of the Java type system to allow higher-order classes and garbage collection to handle some of the more complex tasks. Additionally, we define the grammar, lexer, and parser, which we make use of Antlr for and will not be covered in this final report. What is left is for us to implement the interpreter and the type checker.

Interpreter

The interpreter is implemented using the visitor pattern. We receive our input from the parser as a token stream. The interpreter then calls the 'execute' function, which starts a process that eventually calls the appropriate visitor for the data. To implement functions, the visitor defines a function statement and a call statement that implement the function creation, storing in the environment, and retrieval. Functions are defined with their formal parameters, which we have extended to include types. They also contain a return type, which is used to verify the correct type is returned from the function.

Type Checker

The type checker behaves similarly to the interpreter. It is called immediately before the interpreter is run, and traverses the entire token stream before returning. Instead of computing values, however, it stores types in the

environment. The types are determined by the Java type of the data, though we could have implemented an automata to parse the string values of the data, as the interpreter does. When the automata for a given data type reaches a state where it has no more transitions and cannot transition out of it's current transition, we can infer that the data we are parsing is not of that type. When the string is finished being consumed, if an automata is in an accepting state, that type is taken for the type of the data. The returned type is then passed into the visitor pattern once more, where it is stored in the environment for retrieval later. As data is modified by the program the types are checked by the type checker to ensure the correct data type is being assigned to the variable. When functions are defined, they are stored in the environment wholistically, and the function body is type checked. Later, the function formal parameters are checked against the calling arguments to ensure the correct types are passed into the function call.

Conclusion

The simPL-FT programming language contains a simple type-checker that is capable of enforcing annotated data types on variables and functions. The type checker makes heavy use of the Java type system and the visitor pattern to allow for streamlined data processing while exploring the techniques that are used to implement modern languages. All type checking is done at runtime, though the language does not implement an interactive interpreter. The simPL-FT programming language successfully accomplishes these goals, as demonstrated by the provided test program in the project repository on github.