BILL: Designing a Language for Introductory Computer Science Courses

Andrew Glaude aig2440@rit.edu

github.com/aig24134/MS_Project_Bill

Introduction

Introduction to Computer Science courses are considered challenging by many students who are subjected to an extremely large number of new and novel concepts. Modern programming languages often used in teaching these courses are not ideal as they often lack specific features that could assist first-time programmers. It is the goal of this project to design and implement a new language that will reduce the amount of overhead when learning a new language and allow instructors to focus on core Computer Science principles.

Language Goals

• Compile Time Static Type System
• Easy Access to Common Libraries
• Familiar Syntax

Beginner’s Instructional Little Language

The Beginner’s Instructional Little Language (BILL) is a new language designed from the ground up as a language for use in instructing first time programmers. BILL features all of the aforementioned goals. BILL reduces the number of language features by removing things like: access modifiers, throwing and use in instructing first time programmers. BILL features all of the new language designed from the ground up as a language for Beginner’s Instructional Little Language focuses on core Computer Science principles.

Example Programs

Figure 1 shows a program that prints the first 20 Fibonacci numbers to the console. Much of the syntax and semantics of this example are identical to the C family of languages.

Figures 2 and 3 show BILL’s biggest departure from standard object syntax as seen in Java and C#. The class declaration in figure 1 has a formal parameter list which defines the available fields for a class as well as the constructor. This stems from rit_lib created by James Heliotis.

BILL Interpreter

The BILL Interpreter is implemented in C# and runs on Windows (.NET) as well as Unix (Mono). The grammar for BILL is implemented as an AALR grammar written by Devin Cook[c]. The parser itself uses generated code from the GOLD Parsing System[c] to provide a framework to build the abstract syntax tree. This project also makes use of NUnit, a popular .NET unit testing framework. There are test groups for each of the major parts of the project: the parser, the type checker, and the interpreter. As well as end to end tests that verify these major parts all work together correctly. All of the code is available on github under the MIT License.

The abstract syntax tree follows the Visitor pattern which allows for the creation of various “visitors” that are able to traverse the tree in any way they want. The two main visitors are the TypeValidatorVisitor, and the InterpreterVisitor. By splitting the code this way concerns are kept separate and future work on this project such as building a debugger or a different method of code generation is simple.

Future Work

• Methods in classes
• Better stability - more tests
• A switch to force all objects to be immutable
• Port to a non .NET language
• Create development tools (IDE, debugger)
• A Read-Eval-Print-Loop for Experimentation
• Assertions or Contracts

Conclusion

Teaching first-time programmers the principles of Computer Science is already a difficult task. BILL reduces language complexity allowing instructors to focus on problem solving ability, algorithms, and basic data structures. Basic labs and projects in BILL have reduced complexity compared to Java and C# while still providing compile-time type checking giving the programmer some peace of mind.

References
