CS 6410: Compilers

Fall 2023

Part 5 – Additions

Acknowledgment: Project assignment modified from an assignment developed by Hal Perkins, faculty member of the University of Washington, Allen School of Computer Science and Engineering

Posted on Wednesday, October 4, 2023

Instructor: Tamara Bonaci
Khoury College of Computer Science
Northeastern University – Seattle

Please submit your project by 11:59pm on Wednesday, December 13, 2023. You should submit your project by pushing it to your Khoury GitHub repository, and providing a suitable tag. See the end of this writeup for details.

1 Assignment Overview

The first four parts of this project are expected to be tractable in a one-term course project, while at the same time including enough to cover the core ideas of compiling object-oriented (as well as procedural) languages. In this part of the project, however, you have an opportunity to add additional parts of `Java` to our `MiniJava` programming language.

Please note that this part of the project is worth 10 points in total (unlike the first four parts that were worth 20 points), but you are still expected to attempt it. You are free to explore and implement any of the concepts covered in class, as well as any other concept related to compilers, or to Java, as long as it is not already included in the project. Some ideas, intended to inspire you, or to help you if you are not sure where to start, are listed below.

1.1 Some Simple Ideas

- Add nested* ... */ comments. Add additional arithmetic and relational operators for integers that are not included in the basic `MiniJava` specification.
- Add null as a constant expression and support == and ! = for object reference types.
- Allow return statements anywhere in a method.
- Allow calls of local methods without the explicit use of this.
- Relax the ordering of declarations so that variables can be declared anywhere in a method body.
- Allow initialization of variables in declarations.
- Add void methods, a return statement with no expression, and appropriate type checking rules.
- Support public and private declarations on both methods and instance variables, and check to ensure that access restrictions are not violated.
- Add a top-level `Object` class that is implicitly the superclass of any class that does not explicitly extend another class.
- Add `String` literals, extend `System.out.println` to support `Strings` and overload the + operator for `String` values.
- Add `double` as a numeric type, overload `System.out.println` to print doubles, and implement arithmetic operations for `double`, possibly including mixed-mode integer and floating-point arithmetic.
- Implement some simple compiler optimization techniques (e.g., constant folding, constant propagation, dead code elimination, common subexpression elimination, inlining)

1.2 More Sophisticated, but Interesting Ideas

- Support `instanceof` and type casts (which can require a runtime `instanceof` check). (This actually is surprisingly easy to do.)
- Support `super` as a prefix in method calls.
- Add constructor declarations with optional parameters. To make this useful, you probably want to include the use of `super(...)` at the beginning of the body of a constructor. Support arrays of objects (no harder to implement than arrays of ints, except the type checking rules are more interesting).

Of the suggested extensions, adding `instanceof`, type casts, and `super` are particularly instructive.

2 What to Turn In

For this part of the project, you should include a brief `ADDITIONS-NOTES` file describing all additions and extensions that you have included in your compiler. You should also give a brief explanation of any changes you needed to make in previous parts of the project (scanner, parser, ASTs, semantics) as you implemented the additions.

As before, you will submit this part of the project by pushing code to your GitHub repository. Once you are satisfied that everything is working properly, create a `additions-final` tag, and push that to the repository.

As usual, your code should run on the linux-based machines when built with ant.