CS-153: Introduction to Compiler Design
C−− :
A Minimal C Implementation
Final Report

Skyler Hawthorne, Bruno Lopes, Jeremy Wong

December 12, 2014

1 Introduction

C−− is an implementation of a subset of C. The front-end was generated by a jjTree/JavaCC grammar file. The symbol table and parse-tree procedures were implemented in Java. The backend code-generator was also implemented in Java, and it outputs `*.j` files containing Jasmin assembly code.

1. Most standard C types, including: `int`, `float`, and `char`. However, for the sake of simplicity, we opted not to implement type modifiers such as `long` or `unsigned` types.
2. Basic arithmetic operators, including:
   - Multiplicative (∗, /, %)
   - Additive (+, −)
   - Relational (<, >, <=, >=, ==)
3. Single variable declarations and initialization, as well assignment statements. Multiple variable-declarations turned out to be easy to parse, but hard to generate, so we removed them from our grammar.
4. If statements.
5. While statements.
6. Function calls (parameters are passed by value).

All code uses local variables, including variables in the main function. All of the above generates Jasmin code that can be assembled into `.class` files. We implemented basic error recovery—on a syntax error, it skips to the end of the line and prints the error message.

2 Design

Our design was more or less mirrored from the Pcl example Prof. Mak gave in class. The front-end is generated from jjTree / JavaCC. The intermediate code uses most of the textbooks code. We chose to use the Visitor design pattern to process the parse tree generated by jjTree.

- SimpleNode extends `ICodeNode`
- We use all the type checking, symbol table, and cross-referencing code from the textbook, but modified for Cs syntax.
- `CParserVisitorAdapter` serves as the base class for our `CodeGeneratorVisitor`, which emits all the Jasmin.

- `CodeGenerator` extends the Backend class from the textbook, and uses `CodeGeneratorVisitor` to parse the root node.

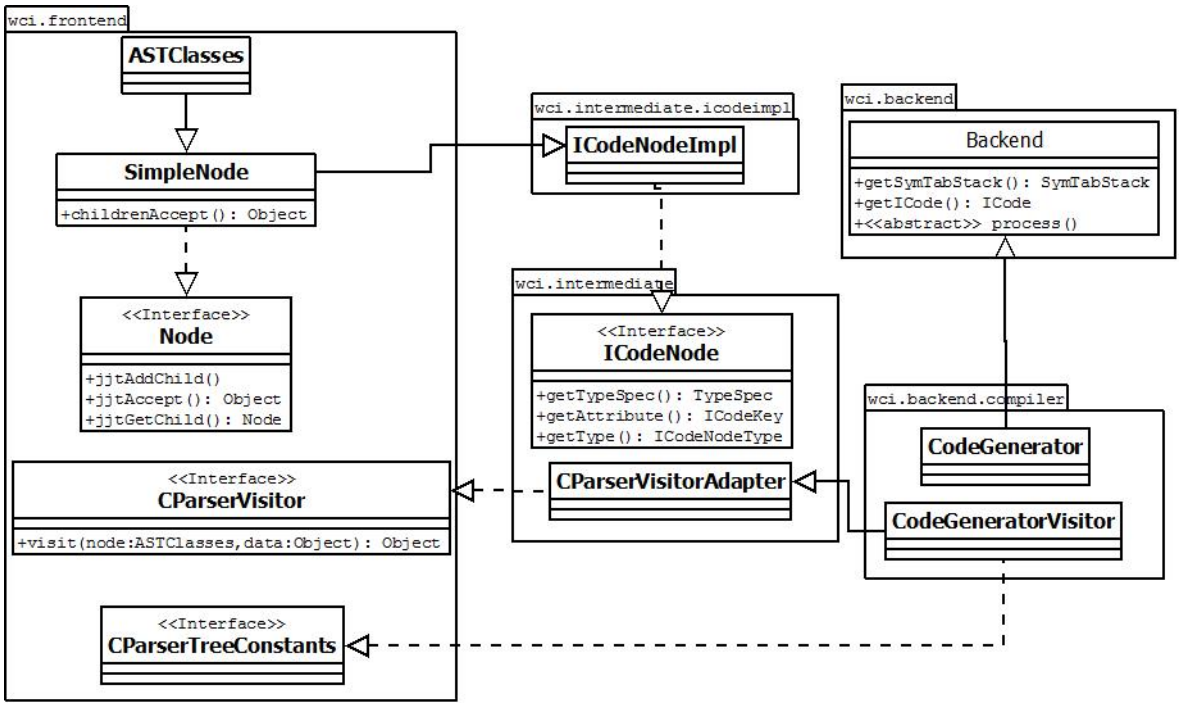

Figure 1: A top-level view of C−−'s design.

3 Grammar

Please see the attached `c.html` file.

4 Code Generation

These are code templates for the if, while, and assignment statements.

1. If statement

   Code to evaluate boolean expression
   `comparison` *false-label*
   Code for IF statement
   `goto` *next-statement*
   false-label:
   code for THEN statement
   next-statement:

2. While statement

   while-label:
   Code for comparison
   `comparison` *false-label*
   Code for WHILE statement
   false-label:

3. Assignment statement

Code for expression
Code for possible type conversions
`putstatic` *programName/fieldName typecode*

5 Build & Run Instructions

1. Unzip the `segfault-final.tar.gz` file.
2. `cd segfault-final`
3. `mkdir bin`
4. `jjtree src/c.jjt`
5. `javacc src/wci/frontend/c.jj`
6. `javac -sourcepath src -d bin src/**/*.java`
   (a) If your shell doesnt support ** globbing, simply passing `CParser.java` should compile everything needed:
   `javac -sourcepath src -d bin src/wci/frontend/CParser.java`

Now you should be able to build any of the sample programs in the test directory. For example, to build `fib.c`:

1. `java -cp bin wci.frontend.CParser ./test/fib.c`
   (a) This outputs `fib.j` in within test, and prints out the parse tree and cross-reference table.
2. `jasmin test/fib.j`
3. `java Fib`
   (a) Make sure `.` is in your `CLASSPATH`.