Compiler Specification

CS212

Fall 2005

This document contains a list of all required specifications that your submission for CS212 must adhere to. It also includes details on assumptions made about the programs that will be compiled with your submission and information on how to handle errors.

1 Compiler Specifications

- Your compiler must use an AST as an intermediate stage between parsing and code generation.
  - The entire AST must be constructed before any code generation begins.
  - The different node types of the AST should correspond to separate classes. Do not confuse this requirement with the fact that all AST classes should be subclasses of a single AST interface or single AST abstract class.
- Your compiler must use symbol tables to keep track of variable locations and variable types.
- You must provide a class called `BaliCompiler` (in file `BaliCompiler.java`) that serves as the starting point for your compiler:
  - `BaliCompiler` must implement the `CS212Compiler` interface.
  - We provide a stubbed version of `BaliCompiler.java`, which you may choose to use.
  - Our `BaliCompiler.java` demonstrates setting up a `PrintWriter` to produce your code as a `String`, which the `CS212Compiler` interface requires.
- Only the starting point (`BaliCompiler.java`) has a specified name. All other classes that are part of your compiler can have any name.
- The `BaliCompiler` you submit must be in the default package. All other files you submit may be in a seperate package, but should not be in the `edu.cornell.cs`.
- Your compiler should not produce any output either to a file or to the standard output or error file. It is OK for your program to produce such output while you are debugging, but any such debugging output must be turned off in your submitted version.
- Your compiler must work with the latest version of SaM as posted on the website.
- Testing information
  - The code you submit should not include a `main` function. We will be testing your compiler by using the method provided by the `CS212Compiler` interface.
  - We supply a main class `BC.java` which you can use to test your compiler. Your compiler is required to work with the provided `BC.java` main class and must implement the provided `CS212Compiler` interface. You may not submit modified version of these files.
- You must generate Samcode that represents a valid translation of the supplied Bali program.

  - Your generated Samcode does not need to contain any comments.
  - Remember to test your Samcode in the SaM Simulator.

- You must adhere to some programming specifications:
  - Your code must comply with the Java 5 specification.
  - You must comment your code and maintain tenets of good programming style.

- Your compiler must be approved by the AMS submission application and submitted as an AMS-generated package.

2 Error Handling

There are two kinds of errors that can occur in Bali programs:

- *syntax errors*: code that violates the rules of a language's grammar.
- *semantic errors*: code that violates the rules of language's semantics.

For example, a missing semicolon or an unmatched parenthesis is a syntax error, whereas a type mismatch or an undeclared variable is a semantic error.

Rather than printing an error message to `System.out` (or `System.err`), errors in supplied Bali programs can be handled most clearly and cleanly by using *exceptions*. We have supplied exception classes corresponding to the two kinds of errors that can occur:

- `BaliSyntaxException` for syntax errors.
- `BaliSemanticException` for semantic errors.
- `IllegalBaliException`, an abstract root class for both the syntatic and semantic exceptions.

The `SamTokenizer` includes several ways to examine/consume tokens. Depending on how you write your code, you may also generate a `TokenizerException`, which is a runtime exception thrown by the tokenizer when various mismatches occur. If you have correctly caught all syntatical errors, your compiler should not generate `TokenizerException`s.

`BC.java`, the main program that runs your compiler, is designed so that it catches `IllegalBaliException`s and then prints the exception's message.

3 Assumptions

While ideally you would be writing a compiler for a "real" language using the actual tools used by compiler developers, there is not enough time to introduce all those tools and design concepts. There are several assumptions that we will be making when testing your compiler, regarding both the parsing and the code itself.

1. You do not need to check variable names. The definition of a name in the grammar is equivalent to the definition of a word token in SamTokenizer. However, you must check for duplicate variable names.
2. You do not need to process the single or double quote delimiters for characters and strings, respectively. Rather, these are handled by the tokenizer and returned as character or string tokens as appropriate. However, when your compiler outputs SaM-code, it must enclose strings/characters in the appropriate delimiters and escape all necessary characters (especially **"** for strings and **'** for characters).
3. Any text that follows **//** and is on the same line as the **//** is ignored. Comments are handled by the Tokenizer and thus do not need to be handled by your compiler.