COMP 520: Compilers

Compiler Project – Final submission

Due: Fri Apr 28, 11:59 PM

The final submission of the project has no new functionality. It’s an opportunity to correct errors in your compiler and make any improvements you wish. It’s also possible to add extensions to your compiler for extra credit. However, no extra credit is needed - all grades can be obtained with the basic project (an exception is a team project which earns score at a lower rate). The following details the parts of your submission that you should place in your final submission directory.

1. Guide to your compiler
This is a short document titled guide.pdf or guide.txt that you place in your final submission directory. The document should contain the following:

- **Scope of your project.** Please make clear any optional parts of the project you have implemented. List known limitations of your implementation (it is better for you to identify these than for me to find them).

- **Summary of changes to AST classes.** Summarize the changes you made to the AbstractSyntaxTree classes that were distributed. Describe changes, if any, made to the AST class structure as well as additions made to the classes to support contextual analysis and code generation.

- **A description of tests (if tests are supplied).** This should describe any tests in the Tests directory (see below). If you are not able to run PA4Test.java you should include some test(s) that exercise the portion of the compiler that is working. If you have completed optional extensions of the project, you should include some comprehensive tests for each extension. You can follow the passxxx, failxxx convention we have been using for tests that should demonstrate correct behavior from your compiler and those that should be rejected by your compiler.

2. miniJava Compiler
Place a copy of your miniJava directory in the submission directory. Your compiler should compile miniJava programs supplied as files with extension "java". For example, if test35.java is a valid miniJava program, your compiler should terminate with exit (0) and generate object code file test35.mJAM.

If the source program is not a valid miniJava program, your compiler should write a diagnostic message and exit t (4). Error messages beyond syntactic analysis need to be prefixed with “***”.

The operation of your compiler should be as specified in PA1-PA4 except where it conflicts with any extensions added.

3. Tests
This directory (if present) contains any special test programs described in your guide.
Grading
The base functionality of the final project, as specified in PA4, will be assessed by functional
testing, inspection of the generated code, and inspection of generated diagnostics. The overall
score of the project is obtained by combining contributions from all five checkpoints to obtain a
normalized project score out of 100.

If you wish, you may earn additional points to add to the overall score by incorporating further
features of Java into miniJava, as shown below. Unless indicated otherwise, the intended
semantics of each feature correspond to Java semantics, and, in general, may involve extensions
or changes in all phases of your compiler.

You can get a fine grade without attempting any extension, so it is perfectly reasonable (and the
typical choice) to concentrate on finalizing the base project. If you do want to investigate an
extension, make sure to understand what would be involved by checking Java semantics. Some
extensions may be more work than their point value suggests.

<table>
<thead>
<tr>
<th>Point value</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Static field initialization.</td>
</tr>
<tr>
<td>2</td>
<td>Parameterized class constructors. There should only be one constructor per class, but it may have parameters. If none is defined, the default constructor should be available.</td>
</tr>
<tr>
<td>4</td>
<td>for loops. Be sure to consider the possible forms of the initialization (including declaration of the iterator variable), loop test, and increment parts.</td>
</tr>
<tr>
<td>4</td>
<td>Add the String type and string literals. No operations need to be supported on strings, but you must be able to assign a string literal or a String reference to a variable of type String, and it must be possible to print String values by overloading System.out.println().</td>
</tr>
<tr>
<td>5</td>
<td>Add overloaded methods that differ in the types of their arguments, and perform type checking to determine their validity and to resolve overloading.</td>
</tr>
<tr>
<td>10–20</td>
<td>Inheritance of fields and methods, and dynamic method invocation. Be sure type checking is extended appropriately. Optionally support instanceof and/or super().</td>
</tr>
</tbody>
</table>