

COMP 520: Compilers

Compiler Project – Final submission

Due: Fri Apr 25, 2014 (accepted no later than Mon Apr 28)

The final submission consists of the following parts.

1. Guide to your compiler

This is a short document that you place in the submission directory in pdf format (ascii text or Word is OK too). The document should contain the following:

- **Scope of your project.** Please make clear which basic and optional parts of the project you have implemented. List known limitations of your implementation (it is better for you to identify these than for us to find them).
- **Summary of changes to distributed components.** This part of the document should summarize the changes you made to the `AbstractSyntaxTree` classes that were distributed. Describe any changes you made to the AST class structure as well as additions made to the classes to support contextual analysis and code generation. You may include changes in mJAM for extensions if necessary, but please provide a justification in addition to explaining what you changed.
- **A description of your tests.** This should describe any tests in the Testcases directory. 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 a comprehensive testcase for each extension.

2. Compiler sources

Place a complete copy of your `miniJava` (and `mJAM` if needed) 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` and a disassembled listing file `test35.asm`. Both of these files can be generated using facilities provided in the `mJAM` distribution.

If the source program *is not* a valid `miniJava` program, your compiler should write a diagnostic message and `exit(4)`.

The operation of your compiler should be as specified in PA1-PA4.

3. Testcases

This directory should contain the test programs you have described in your guide.

4. Grading

The base functionality of the final project, as specified in PA4, will be assessed by functional testing, inspection of the generated code, and evaluation of generated diagnostics. The overall score of the project is obtained by combining contributions from all checkpoints to obtain a total project score between 0-100.

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.

Point value	Feature
2	Static field initialization.
3	Add keyword null and support its use. Be sure uses are type correct.
3	Support class constructors with parameters.
4	for loops. Be sure to consider the possible forms of the initialization (including declaration of the iterator variable), loop test, and increment portions of the for statement.
6	Add a 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 variables of type String, and it must be possible to print String values by overloading <code>System.out.println()</code> .
6	Code generation for conditional operations <code>&&</code> and <code>and</code> and <code> </code> to select efficient code sequences when used in expressions or when used as predicates in conditional or repetitive statements. Efficient code sequences means: (1) there is no alternation between jumps and construction of truth values on the stack and (2) there are no chains of consecutive jumps without intervening tests in the evaluation of a conditional expression.
8	Add overloaded methods that differ in the types of their arguments, and perform type checking to determine their validity and to resolve overloading.
20-25	Inheritance of fields and methods, and dynamic method invocation. Be sure type checking is extended appropriately. Optionally support <code>instanceof</code> and/or <code>super()</code> .