

# Bantam Java Compiler Usage Manual



Marc L. Corliss

Hobart and William Smith Colleges

*corliss@hws.edu*

*<http://math.hws.edu/mcorliss>*

E Christopher Lewis

VMware

*lewis@vmware.com*

*<http://www.eclewis.net/>*

©2007, Marc L. Corliss and E Christopher Lewis

This manual is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 2.5 License (<http://creativecommons.org/licenses/by-nc-nd/2.5/>). This license allows you to duplicate and distribute this manual in unmodified form for non-commercial purposes. See the license for more details.

Note: this document was extracted from a larger document covering all aspects of the Bantam Java compiler project. It is available at <http://www.bantamjava.com>.

This manual describes the command-line usage of the Bantam Java compiler. Although parts of the student's compiler are unimplemented (and will eventually be implemented by the student), the main class of the compiler is implemented, which includes command-line, flag handling. So the unimplemented compiler will accept all options, although some may have no effect (*e.g.*, enabling debugging for the code generator). If using the reference Bantam Java compiler, then all flags will work except most of the debugging flags (the parser debugging flag works but the others do not).

The Bantam Java compiler accepts the following options:

```
bantamc [-o <output_file>] [-t <architecture>] [-gc] [-dl] [-dp] [-ds] [-dc]
        [-sl] [-sp] [-ss] <input_files>
```

**Options.** Each of the compiler options are enumerated and discussed below.

**-o output\_file**

Specify the output file to use; the default is out.s

**-t architecture**

Specify the target architecture to generate code for; x86 and mips (must be all lowercase) are the currently supported target architectures. mips is the default.

**-gc**

Enable the garbage collector. The Bantam compiler uses a simple, mark-and-sweep garbage collector. By default the garbage collector is disabled.

**-dl**

Enable lexical analysis debugging. Note: students must add the debugging code that uses this flag. By default this flag is off.

**-dp**

Enable syntactic analysis debugging. Note: students must add the debugging code that uses this flag (unless using the Java Cup implementation, in which case, debugging code is automatically inserted by the parser generator). By default this flag is off.

**-ds**

Enable semantic analysis debugging. Note: students must add the debugging code that uses this flag. By default this flag is off.

**-dc**

Enable code generation debugging. Note: students must add the debugging code that uses this flag. By default this flag is off.

**-sl**

Stop the compiler after lexing. If this flag is set, the compiler halts after performing lexical analysis. If lexical errors are discovered, these are printed to standard error, otherwise the scanned tokens are printed to standard output. By default this flag is off.

**-sp**

Stop the compiler after parsing. If this flag is set, the compiler halts after performing syntactic analysis. If lexical or syntactic errors are discovered these are printed to standard error, otherwise the parsed program is printed to standard output (as a Bantam source program). If `-sl` is specified, then this flag is ignored. By default this flag is off.

**-ss**

Stop the compiler after semantic analysis. If this flag is set, the compiler halts after performing semantic analysis. If errors are discovered these are printed to standard error, otherwise the annotated program (annotations indicate type information) is printed to standard output (as a Bantam source program with annotations in comments). If `-sl` or `-sp` is specified, then this flag is ignored. By default this flag is off.

**Input files.** The input files to the compiler must be Bantam files. These files should contain Bantam code and end with the suffix “.btm”. If an input file is specified that does not end with “.btm” then the compiler will immediately print an error and exit. If the input files contain errors, then the compiler will not produce an output file, but instead print error messages and exit (unless the student has implemented a buggy compiler).

**Output file.** The output file is either a MIPS or x86 assembly file depending on the `-t` flag (if `-t` is not specified then MIPS is the target). Although not required, by convention this file should end with “.s”. By default, the compiler uses the file “out.s”.

**Examples.** Here are some example uses of the Bantam compiler. These examples assume we are in the `tests/` directory.

```
bash$ bantamc -o foo.s Foo.btm
bash$ spim foo.s
```

The first command above will compile the Bantam program **Foo.btm** and create a MIPS assembly file (to be used with the SPIM emulator) called **foo.s**. The second command above runs the program using the SPIM emulator for MIPS.

```
bash$ bantamc -t x86 -o foo.s Foo.btm
bash$ gcc -o foo foo.s lib/runtime.s
bash$ ./foo
```

The commands above compile and run **Foo.btm** on an x86, 32-bit machine. The first command compiles **Foo.btm** to **foo.s**, specifying the target as x86. The second command uses the gcc assembler to translate the assembly file to an executable called **foo**. Finally, the third command runs the program. The x86 target uses the AT&T x86 assembly format, and must be run with an AT&T assembler, such as gas, the GNU assembler that is bundled with gcc. Currently, the x86 target can only be used on 32-bit x86 machines. In the future, we will add a target for 64-bit x86 machines.

```
bash$ bantamc -gc -t x86 -o foo.s Foo.btm
bash$ gcc -o foo foo.s lib/runtime.s
bash$ ./foo
```

The commands above are identical to the previous set of commands (they compile and run **Foo.btm** on an x86 machine) except in this case the garbage collector is used.

```
bash$ bantamc -sl Foo.btm
```

The command above performs only lexical analysis on **Foo.btm** and then prints out the scanned tokens to standard output, unless errors are discovered, in which case the errors are printed to standard error.

```
bash$ bantamc -dp -sp Foo.btm
```

The command above performs lexical and syntactical analysis on **Foo.btm** and then prints out the parsed program to standard output, unless errors are discovered, in which case the errors are printed to standard error. In addition, debugging is enabled during parsing.