CS 431 – Assignment 2 – Fall 2019

Due: Thursday, October 10th **BEFORE 8:00AM**

Total points: 100

You **must** work on this assignment with the same partner as you did for assignment 1. Both team members will receive the same grade unless I am made aware of significant collaboration issues within the team (e.g., one team member not completing a reasonable share of the work).

Learning outcomes

- Define the phrase structure of a programming language using an EBNF grammar and transform it into an equivalent grammar that is as close to LL(1) as possible.
- Implement a recursive-descent parser using a parser generator.

Each assignment will carry a weight between 1 and 5 when it gets factored into your overall project grade for this course. The weight of this assignment is 2. Later assignments will likely carry a larger weight, since they get more and more involved. Be aware that each and every assignment in this course requires that all previous assignments have been completed successfully.

Preparation

- Study the Bantam Java grammar on page 103 of the manual. You need NOT read the rest of Chapter 8 since this assignment is significantly different from the one described in that chapter. By the way, make sure to re-download the manual from the Canvas site before starting work on this assignment since I made a few changes to the grammar on page 103.
- Review your notes on top-down parsing, LL(1) grammars, and JavaCC.
- Read this handout carefully right away, and again just before submitting your work. **It would be a shame to lose points for failing to follow the directions given below.**
- Download the file `a2.zip` from the Canvas site. It will expand into a directory called `a2` containing the Bantam Java compiler shell.

Problem Statement

Your task is to complete a “yes/no parser” for the Bantam Java language using the parser generator JavaCC. In this assignment, your parser will take as input a sequence of tokens from your scanner and parse it according to the Bantam Java grammar. If parsing succeeds, a message is displayed. Otherwise, that is, if the input program is not syntactically correct, the compiler throws a `ParseException` and terminates. In the next assignment, you will complete the parser to make it output an abstract syntax tree instead of the “yes” answer.

The only file you need to modify for this assignment is `parser.jj` in the `src/parser` directory. After copying your token rules from the previous assignment into this file at the specified location, you must add your production rules at the end of the file. Finally, this file includes a specification for the precedence and associativity of the operators in the grammar, which you must enforce (it is slightly different from the one given in the manual). Here are some observations and “gotchas” to keep in mind as you develop your parser:

- Expressions can be tricky. Write your grammar to get the precedence and associativity of expressions correct now, or you will really suffer on the next assignment when we produce abstract syntax trees, which must reflect precedence and associativity.
- Method calls and assignment statements can be tricky for LL(1) parsers. You may need to left-factor. Use **as few LOOKAHEAD directives as possible**. Don't abuse them, using your best judgment here.
- LOOKAHEAD directives are useful, but can be dangerous, since you will not get warnings for the sections that use LOOKAHEAD. You MUST first try to left-factor, etc. The use of too many LOOKAHEAD directives or LOOKAHEAD directives with too large a lookahead value will both be penalized.
- Take advantage of EBNF extensions, since JavaCC supports them. But keep in mind the observations we made in class when we built parsers using JavaCC.

Testing your compiler

The following instructions assume that your current directory is the top-level `a2` directory. To build your parser type: `ant src`. To run your compiler on a Bantam Java program in the `tests` directory, type:

```
./bin/bantamc tests/file_name.btm
```

To run your compiler on all files in the `tests` directory, type: `ant tests`. Finally, I will test your submission by typing: `ant all` or simply `ant`, which will compile and run your compiler.

For full credit, JavaCC must build your parser with no errors and no warnings. Your compiler must pass all provided (and possibly other) tests.

Let me point out the fact that you may run the JavaCC-generated parser in debug mode. To do so, you simply need to rebuild your parser with a certain flag set to true. To achieve this effect in my ant script, make the following single-line change in the `src/build.xml` file, from:

```
<target name="build-parser">
  <javacc target="${PSRC}" outputdirectory="${PPKG}"
      javacchome="${JAVACC}"/>
</target>
```

to

```
<target name="build-parser">
  <javacc target="${PSRC}" outputdirectory="${PPKG}"
      javacchome="${JAVACC}" debugparser="true"/>
</target>
```

When the `debugparser` flag is on, the parser will output a pretty detailed trace of its execution, which is quite helpful while debugging.

Submission procedure

Before the deadline, you must copy to the A2 dropbox on Canvas a single file called `parser.jj` and turn in a hard copy of it at the beginning of class on the due date.

Make sure to format your `parser.jj` file nicely, with proper indentation and no more than 80 characters to a line to **prevent wraparounds** in your printout. Then use the Linux utility *enscript* for printing, two columns per page and in landscape. Here is the command that I want you to use:

```
> enscript -2rBh -c parser.jj
```

Do NOT print your file from within your IDE, as the color-coding used by the editor may result in a partially grayed-out, hard-to-read printout. Do NOT print in duplex (double-sided) mode. Note that the `-c` option will truncate every line that is too long. Since wraparound code is painful to scan, I avoid doing so at all cost. Bottom line: if your hard copy contains wraparound or truncated lines, you will lose points (proportionally to the number of such occurrences).

Points will be deducted if your parser does not work properly, and/or if you do not follow the naming/formatting/submisssion requirements listed in this handout.

The late-submission penalty stated in the syllabus applies to this and following assignments.

Good luck and have fun!