Compilers [Fall 2016]

Programming Assignment IV

Objectives

1. To gain experience using bison, an LALR(1)-parser generator.
2. To understand the format of abstract syntax trees (ASTs) in DISM and DJ.
3. To implement an AST-building parser for DJ programs.
4. To practice writing semantic actions in bison CFGs.

Due Date: Sunday, October 23, 2016 (at 11:59pm).

Machine Details

Complete this assignment by yourself on the following CSEE network computers: c4lab01, c4lab02, ..., c4lab20. These machines are physically located in ENB 220. You can connect to the C4 machines from home using SSH. (Example: Host name: *c4lab01.csee.usf.edu* Login ID and Password: <your NetID username and password>) You are responsible for ensuring that your programs compile and execute properly on these machines.

Assignment Description

This assignment asks you to extend your dj2dism compiler so that it produces ASTs for DJ programs. You will again use bison (described in Section 5.5 of the textbook).

Begin by downloading the *ast.h* file at: http://www.cse.usf.edu/~ligatti/compilers-16/as4/. This file declares data structures and methods for DJ ASTs.

Then, create a new file called *ast.c*, and in it implement the methods declared in *ast.h*. Also modify your *dj.y* file from Assignment III by adding semantic actions to build appropriate ASTs for DJ programs. Your parser must print the AST for the input DJ program before exiting. The example executions below illustrate how to format the AST output.

If you have any errors in your *dj.y* file from Assignment III, you will need to correct them for this assignment. Part of your task for this assignment is to fix any bugs you uncover in your original DJ grammar.

Big Hint

The DISM simulator uses code to build and print ASTs that is very similar to what you need to write for this assignment. You will find it helpful to study the DISM simulator's full *dism.y*, *ast.h*, and *ast.c* files. Recall that all the DISM simulator code is posted at: http://www.cse.usf.edu/~ligatti/compilers-16/as1/dism/sim-dism/

Compilation of the Parser

You already have a *lex.yy.c* lexer for DJ. Run the following commands to create and compile your AST-building parser as a program called dj-ast.

```
> bison -v dj.y
> gcc dj.tab.c ast.c -o dj-ast
```

As with Assignment III, for full credit your parser must not have any conflicts or right recursion (except as allowed in Assignment III).

Example Executions
Your dj-ast should output ASTs for lexically and syntactically valid DJ programs. (As with Assignment III, your nascent compiler should report lexical and syntactic errors before exiting; in these cases, the compiler exits before producing an AST.) Format the AST output as follows.

```
> ./dj-ast good1.dj
0:PROGRAM   (ends on line 6)
1:  CLASS_DECL_LIST   (ends on line 1)
2:    CLASS_DECL   (ends on line 1)
3:      AST_ID(C)   (ends on line 1)
3:      OBJ_TYPE   (ends on line 1)
3:      VAR_DECL_LIST   (ends on line 1)
3:      METHOD_DECL_LIST   (ends on line 1)
1:  VAR_DECL_LIST   (ends on line 2)
1:  EXPR_LIST   (ends on line 3)
2:    NAT_LITERAL_EXPR(0)   (ends on line 3)
```

Notice that this output contains one line for each AST node. The nodes are output in a *preorder* AST traversal (i.e., we print a tree by printing its root and then recursively printing every tree that is a child of that root). Every line of output contains the node's depth *d* in the tree, then a colon, then *2*d* spaces, then the name of the AST node (with any node attributes as appropriate), and then the source-program line number on which this construct ends (recall that these line numbers can be obtained while building the AST by using the flex/bison yylineno variable).

The line numbers for AST nodes whose type ends with "LIST" (e.g., CLASS_DECL_LIST) may be off from the line numbers shown in this handout's example executions. If this happens, it just means your grammar produces these lists in a different way than mine does. E.g., in the example execution above, it'd be OK to state that the CLASS_DECL_LIST ends on line 2, the VAR_DECL_LIST ends on line 3, and the EXPR_LIST ends on line 4.

Additional examples:

```
> ./dj-ast good2.dj
0:PROGRAM   (ends on line 2)
1:  CLASS_DECL_LIST   (ends on line 1)
1:  VAR_DECL_LIST   (ends on line 1)
1:  EXPR_LIST   (ends on line 1)
2:    NAT_LITERAL_EXPR(0)   (ends on line 1)
```

```
> ./dj-ast good12.dj
0:PROGRAM   (ends on line 9)
1:  CLASS_DECL_LIST   (ends on line 1)
2:    CLASS_DECL   (ends on line 3)
3:      AST_ID(C)   (ends on line 1)
3:      OBJ_TYPE   (ends on line 1)
3:      VAR_DECL_LIST   (ends on line 1)
3:      METHOD_DECL_LIST   (ends on line 2)
4:        METHOD_DECL   (ends on line 2)
5:          NAT_TYPE   (ends on line 2)
5:          AST_ID(f)   (ends on line 2)
5:          NAT_TYPE   (ends on line 2)
5:          AST_ID(n)   (ends on line 2)
5:          VAR_DECL_LIST   (ends on line 2)
5:          EXPR_LIST   (ends on line 2)
6:            NAT_LITERAL_EXPR(5)   (ends on line 2)
1:  VAR_DECL_LIST   (ends on line 4)
2:    VAR_DECL   (ends on line 5)
3:      AST_ID(C)   (ends on line 5)
3:      AST_ID(c)   (ends on line 5)
1:  EXPR_LIST   (ends on line 6)
2:    ASSIGN_EXPR   (ends on line 6)
3:      AST_ID(c)   (ends on line 6)
3:      NEW_EXPR   (ends on line 6)
4:        AST_ID(C)   (ends on line 6)
2:    PRINT_EXPR   (ends on line 7)
3:      DOT_METHOD_CALL_EXPR   (ends on line 7)
4:        ID_EXPR   (ends on line 7)
5:          AST_ID(c)   (ends on line 7)
4:        AST_ID(f)   (ends on line 7)
4:        DOT_METHOD_CALL_EXPR   (ends on line 7)
5:          ID_EXPR   (ends on line 7)
6:            AST_ID(c)   (ends on line 7)
5:          AST_ID(f)   (ends on line 7)
5:          DOT_METHOD_CALL_EXPR   (ends on line 7)
6:            ID_EXPR   (ends on line 7)
7:              AST_ID(c)   (ends on line 7)
6:            AST_ID(f)   (ends on line 7)
6:            NAT_LITERAL_EXPR(0)   (ends on line 7)
> ./dj-ast good13.dj
0:PROGRAM   (ends on line 5)
1:  CLASS_DECL_LIST   (ends on line 1)
1:  VAR_DECL_LIST   (ends on line 1)
1:  EXPR_LIST   (ends on line 3)
2:    IF_THEN_ELSE_EXPR   (ends on line 3)
3:      NOT_EXPR   (ends on line 2)
4:        NOT_EXPR   (ends on line 2)
5:          NOT_EXPR   (ends on line 2)
6:            NOT_EXPR   (ends on line 2)
7:              NAT_LITERAL_EXPR(0)   (ends on line 2)
3:      EXPR_LIST   (ends on line 2)
4:        PRINT_EXPR   (ends on line 2)
5:          NAT_LITERAL_EXPR(0)   (ends on line 2)
3:      EXPR_LIST   (ends on line 3)
4:        PRINT_EXPR   (ends on line 3)
5:          NAT_LITERAL_EXPR(1)   (ends on line 3)
```

Extra Credit

For up to +20% extra credit, build and print ASTs in a recursive-descent parser that satisfies the extra-credit constraints for Assignment III.

Submission Notes

- Type the following pledge as an initial comment in your *dj.y* and *ast.c* files: "I pledge my Honor that I have not cheated, and will not cheat, on this assignment." Type your name after the pledge. Not including this pledge will lower your grade 50%.
- Upload and submit your *dj.y* and *ast.c* files in Canvas. Please upload and submit each file by itself; do not zip them into one.
- You may submit your assignment in Canvas as many times as you like; we will grade your latest submission.
- For every day that your assignment is late (up to 3 days), your grade reduces 10%.
- To make it easier for our teaching assistant to read and evaluate your code, please use spaces rather than tabs in your code and avoid long lines of code (I try to limit lines to 80 characters in width).
- Your programs will be graded on both correctness and style, so include good comments, well-chosen variable names, etc. in your programs.