

CS126: Introduction to Compilers

Assignment 2: LEX

Out: 9/12/03

Due: 9/19/03

1.0 Introduction

The purpose of lexical analysis in the compiler is to break the input (from one or more files) into a stream of tokens suitable for parsing. In this assignment you are going to write a lexical analyzer for DECAF using lex (or flex if you desire).

Your job is to write the lexical portion of the JavaCC syntax specification for DECAF along with a small amount of support code that will serve to provide the connection between the body of the compiler and JavaCC.

2.0 Interfaces

Your lexical specification should generate all tokens that are valid as part of the Decaf language. In addition, it should recognize any other character or character sequence that might be in a file. (That is, it should produce a token for all input, even if that input is not part of Decaf.) You can assume that the input is all ASCII (not Unicode).

Several steps are required to integrate with the rest of the compiler. First, you should create a subdirectory and check out the compiler code from cvs. The instructions for doing this will be on the website/newsgroup. Second, you should create your own subdirectory in the compiler source directory. This directory will correspond to a subsidiary package (`spr.decaf.<yourid>`), where you will be writing your compiler code. Third, you should create a class in that directory that extends `DecafFactory`. For now, this class should define one method,

```
DecafParser newParser(DecafErrorHandler)
```

that returns an instance of your parser class. Next you should create a JavaCC source file that defines your parser class. Note that your parser class should implement `spr.decaf.DecafParser`. This means that you will have to provide implementations of the basic methods of `DecafParser`. A sample implementation is shown in Figure 1. Note that this code goes in the `PARSER_CODE` section of the JavaCC file. Next you can add the appropriate options and lexical specifications to the JavaCC file. Finally, you should build the compiler (see instructions on the web/newsgroup).

```

private DecafErrorHandler error_handler;
private String cur_file;

public <PARSER>(DecafErrorHandler err)
{
    error_handler = err;
    cur_file = null;
    disable_tracing();
}

public boolean setFile(String file)
{
    try {
        FileInputStream fis = new FileInputStream(file);
        ReInit(fis);
        cur_file = file;
    }
    catch (IOException e) {
        error_handler.addError(ERROR_ERROR,null,"can't open file");
        return false;
    }
    return true;
}

void parserFile() { }

DecafAst getAstRoot() { return null; }

Object nextToken()
{
    try {
        Token t = token_source.getNextToken();
        if (t.kind == 0) return null;
        return t;
    }
    catch (TokenMgrError e) {
        System.err.println("Token error: " + e.getMessage());
        error_handler.addError(ERROR_FATAL,null,"lexing problem");
    }
    return null;
}

```

FIGURE 1. Sample parser implementation

To run the compiler, use the `-lexonly` option (`-l` is sufficient). This should cause the program to print the stream of tokens for the specified source file(s). A directory of test files (based on assignment 1) will be made available.

3.0 Mechanics

Handins should be done electronically. (Directions will be posted in the `cs126` newsgroup/website). You are again free to work in small teams on this assignment. You should hand in a copy of your JavaCC file along with a listing of the token stream for enough of the sample input files to demonstrate that your program works. Alternatively, you can create your own test files.