

CS 654: Homework # 1

due 2012/2/7

1 Measuring a Scanner

On “weise” (or any machine using gcc 3.1 or up) run the C++ preprocessor (`g++ -E`) on the following program:

```
#include <iostream>

int main()
{
    std::cout << "Hello, world!" << std::endl;
}
```

- List the files that the preprocessor must look at in its *input*. (Hint pipe the results of `g++ -E hello.cc` through `grep '\# 1'` to find the lines where an included file starts in the result. Ignore the first three lines of output.) Use `wc` to estimate how many characters the preprocessor must look at, broken down by file. (Some files are read twice; don't forget to add up *all* the characters read.)
- How many characters are in the *output* of the preprocessor?
- The output of the preprocessor is the input of the scanner proper. Regular comments have already been removed, but the preprocessor adds lines starting with `'#'` that are treated as comments by the scanner. How many tokens (approximately) represent the *output* of the scanner? (Please indicate how you made the estimate.)
- Write a paragraph analyzing your results. What do they indicate about scanning (even short) programs in C++?

2 Regular Expressions

Solve Exercise 2.1(e) on page 103 of the textbook (3rd edition), inexact constants in Scheme. Give a regular expression AND a deterministic finite state automaton. Explain how they work.

3 Testing a Scanner

In order to test a Cool scanner, one needs to test all situations: every digit, every letter, every keyword, every operator. But it is also necessary to test different kinds of errors: what can go wrong inside a string or a long string. A file could end at a particularly tricky point inside a long string. Since each file can test “end of file” in only one way, we need multiple test files. Write a series of test files (and place in your `homework1` directory) to test the Cool scanner exhaustively. Use the script `test2` in `$CLASSHOME/cmd` to see how many situations your files are covering:

```
test2 [-v] file1.cool ...
```

Use the `-v` flag to get help on what tests are still missing.

CS 754: Homework # 1

due 2012/2/7

1 A Family of Scanners

Read the original lex and flex papers

Lesk75 M. E. Lesk and E. Schmidt, *Lex — A Lexical Analyzer Generator*, 1975.

Paxson95 V. Paxson, *Flex, A Fast Scanner Generator*, 1995.

Then read documentation for JLex and JFlex (the engine behind `coollex`). (To find lex, I had to download the ancient “Unix Programmer’s Manual” volume 2b.)

Explain the motivation behind each tool. Are there new features? Are some features no longer supported? Why does the “lex” idea persist? Write about 500 words, explaining the progression.

2 Alternatives

The “lex” method has several robust alternatives. First, most scanners are hand-written. Why? Also there is the technique of “scannerless parsing.” What “lex” style ideas are used in scannerless parsing? Write about 1000 words total on the reasons (good or bad) for hand writing scanners and about scannerless parsing. Provide a bibliography and compare published work in this area.