CS 164, Fall 2020 **CS 164: Homework #1** **P. N. Hilfinger**

Due: Friday, 11 September 2020, 2400

General instructions about homework. Please submit this homework electronically. See Submitting Your Work in *Using Git in CS164* on the class web page. You will need GIT and you will need to have electronically registered a CS instructional login for CS164, using Webacct. In any case, please notify us of problems.

You'll find templates for some solutions in the `shared` repository. Set up a local GIT repository in a working directory (say `cs164-repo`) as described in *Using Git in CS164*, and then start your assignment with

```
$ cd cs164-repo
$ git fetch shared
$ git merge shared/hw1
$ git push
```

Your local repository will now be on branch `master` and contain a directory named `hw1` with some files in it. Place answers to any questions that don't require programs in the file called `hw1/hw1.txt`.

If you are working at home using your own installation (as opposed to using the instructional machines remotely), you'll need a Python installation. See the Python link from the class home page if you need one.

1. A *subsequence* of a string S (or any kind of sequence, really) is simply a sequence consisting of zero or more characters from S in the same order as in S; for example, `"ack"`, `""`, `"bk"`, and `"back"` are all subsequences of `"back"`, but `"kb"` and `"bb"` are not. A *substring* of S is a *contiguous* subsequence of zero or more characters of S; for example `"ack"`, `"ba"`, `""` and `"back"` are substrings of `"back"`, but `"bk"` is not. Given a string S of length N, how many subsequences does it have? How many substrings? To simplify your life, you *may* assume that letters in S are not repeated (the problem becomes rather "interesting" if you count only distinct subsequences when S may contain repetitions).

More problems on the next page.

2. [From Aho, Sethi, Ullman] Give as simple a description as possible of the languages denoted by the following regular expressions. For example, it is better to describe the language denoted by `((a*b*)*(b*a*)*)*` as "The set of all strings from the alphabet $\{a, b\}$" rather than "A sequence of 0 or more a's followed by 0 or more b's, all repeated 0 or more times and then followed by *etc.*" The latter might be correct, but shows no thought.

a. `1(0|1)*1`

b. `((ϵ|1)0*)*`

c. `(0|1)*1(0|1)(0|1)(0|1)`

d. `0*10*10*`

e. `(00|11)*((01|10)(00|11)*(01|10)(00|11)*)*`

3. [Adapted from Aho, Sethi, Ullman] Write the simplest regular expression you can for each of the following languages, using basic regular expressions from Python. Turn in files called **P3a.py**, **P3b.py**, etc., containing solutions to these problems, using the templates for these files provided in the skeleton for this homework set. Python "regular expressions" are considerably more powerful than the "classical" basic expressions we want here. For this problem, restrict yourself to using just ordinary characters (that stand for themselves), plus the special characters

```
[ ]  ( )  *  +  |  .  $  ^ ?
```

In the following, "letters" mean lower-case letters. Each pattern that you write must match the *entire* input string (an entire line with the frameworks provided) for a match to succeed, not just an initial segment.

a. Strings of letters that contain all five vowels in reverse order (but not necessarily continguously), each one exactly once.

b. Strings composed of letters a–f in which the letters are in alphabetical order (not all letters have to appear in a string, but those that do must be in order).

c. Comments consisting of a string starting with `/*` and ending with `*/` without any `*/` in between unless it occurs inside balancing quotes `"` and `"`. Quotes must be balanced, so that `/*"*/` is illegal, and only balancing pairs of quotes "deactivate" the `*/` symbol, so that `/*""*/""*/` is illegal (the `*/` occurs between quotes, but they don't balance each other).

d. All strings of 0's and 1's with an even number of 0's and an odd number of 1's.

e. All strings of 0's and 1's that do not contain the substring '101'.

f. All strings of 0's and 1's that do not contain the subsequence '101'.

4. In lecture you saw one definition of a *regular grammar*. Let's be specific about (meta)syntax for this problem and say that rules in this grammar have the following form:

$$\alpha_0 \ \texttt{:} \ \alpha_1\alpha_2\cdots\alpha_n \ \texttt{;}$$

where $n \geq 0$, α_0 is an identifier (as in Java), and each α_i for $i > 0$ is either an identifier or one of the two literals `'0'` or `'1'` (in single quotes). These rules define the identifiers (nonterminals) to be sets of strings. The two literals represent {`"0"`} and {`"1"`} as for regular expressions. There can be any number of these rules, subject to the restrictions from lecture:

- Each α_i for $0 < i < n$ must be a literal or an identifier whose rules all appear before any of the rules for α_0; and
- α_n, for $n > 0$, may also be α_0.

Write regular grammars equivalent to each of the regular expressions in problem 2. Turn in files called **P4a.y**, **P4b.y**, etc., containing solutions to these problems, using the templates for these files provided in the skeleton for this homework set. You can compile these into test programs with the command `make` on the instructional machines (at home, you'll need the BISON program, the `make` program, and `gcc`).