Interpreter, quote

This portion of the project is a relatively small enhancement of the previous portion, where you add functionality for the special form `quote`. You should build this assignment as an extension of the work that you did for the previous one.

This assignment is to be done individually. You can talk to other people in the class, me (Dave), and any of the course staff (graders, lab assistants, teaching assistants, prefects) for ideas and to gain assistance. You can help each other debug programs, if you wish. The code that you must be your own, however, and you are not permitted to share your code with others, not even your partner from other portions of the course. See the course syllabus for more details or just ask me if I can clarify.

1. Get started

Download this 06-quote.zip file, and extract the zip file to your ProgrammingLanguages folder that you've been working in.

2. Functionality

For this phase of the project, you must additionally support the correct evaluation of the following Scheme expression:

```
(quote expr)
```

Likewise, you'll need to support evaluation of the abbreviated form

```
'expr
```

That said, the parser should already be converting the second form into the first.

3. Sample output

```
$ cat test-in-01.scm
(quote a)
(quote (a b c))
(quote (a b (quote (c d e))))
(let ((x (quote a)) (y (quote (a b c))))
  y)

$ ./interpreter < test1-in-01.scm
a
(a b c)
(a b (quote (c d e)))
(a b c)
```

4. Don't annotate the parse tree

Do *not* implement this by trying to somehow wrap up the quoted expression in some sort of enclosure indicating that it is quoted.

Some of you will feel tempted to do otherwise! Some of you may find that you are convinced that somehow storing `quote` or a single quote somehow explicitly after evaluation will help you in some way. Some of you might even find that by doing so, you can succeed at this task. Don't do it. It's a trap. It might help you get this part of the assignment to work, but you'll get into hot water later. This reflects a common and interesting point of confusion: `quote` does not tell the interpreter to *do* something to its arguments; rather, it tells the interpreter to *not do* something.

If you are feeling a strong urge to somehow specially annotate something indicating that something is quoted, don't do it. If you are experiencing such compulsions because your tree is not printing properly, past experience has sometimes been that the problem is the code for *printing* the parse tree, not structuring it in the first place.

5. Your repository and your files

Your repository will follow the same structure that it has for the previous assignments. There should be no changes at all in the files that I provide; you'll then need to add in the files from your previous version, and/or switch over to using my binaries. But note that the binaries I provide only go through part 4 of the project; I am not providing a working if/let interpreter. (The project gets too intermingled at this point for me to be able to supply complete partial work.)

Building and testing your code will work precisely the same as on the previous assignment.

6. Grading

It is important that you follow the instructions above about not wrapping up the quoted expression in some sort of enclosure. The code for `quote` is notable in what it doesn't do, rather than in what it does do. TRUST ME from past experience; if you get this done incorrectly, you may pass all the tests, but you will get bitten later. To help highlight this at this stage, the graders will be looking through your source code to estimate as best as they can if you have done this correctly.

7. What to submit

Follow the instructions for the last assignment: submitting should be the same.

Have fun interpreting!

8. Preparing for celebrations of knowledge

The celebration of knowledge may ask you to write on paper C code to solve problems similar to the above. It might have you write the code from scratch; it might have you fill in some blanks in partially written code; or it

might use some other method for having you demonstrate that you can write code. To prepare, you should practice writing C code on paper that can solve any of the above functions, or others like it.

This is assignment is quite brief, so a celebration of knowledge might also ask conceptual questions. Why is the quote form implemented as we did? Is it a proper Scheme function or not, and why? And so on.

This assignment was originally created by David Liben-Nowell and has since been updated by Dave Musicant, Jed Yang, and Laura Effinger-Dean.