

Static Scoping Assignment (SSPS)

CptS 355

Fall 2009

November 11, 2009

Assigned: 11 Nov, 2009

Due: Friday, 4 Dec 2009, 11:59:59PM. Develop all your code in a directory named “ssps”. When you are finished make a gzipped tar file or zip file of the directory and turn it in using the turn-in page as for previous assignments. Include a README file describing the environment for running your interpreter. The name of the main executable file should be `ssps.py` or just `ssps`.

Credit: This assignment will count approximately 11% of your final grade.

Policy: This assignment is to be your own, personal, individual work.

Project 4 is intended to enhance understanding of the behavior of, and mechanisms used to implement, static and dynamic scope rules.

You will be modifying your SPS interpreter to handle a slightly different language. Let's call it Scoped Simple PostScript, SSPS. SSPS has no `dict`, `begin` or `end` operations. Instead, each time a procedure is called a new dictionary is automatically pushed on the dictionary stack. The dictionary must be able to hold an arbitrary number of names.

The SPSS interpreter will take a command line switch, either `-s` or `-d`, to indicate whether it should behave using static scope rules or dynamic scope rules. The default, if no switch is supplied, is *dynamic scope rules*. You can check for the command line switch by importing the `sys` module and then inspecting the variable `sys.argv` which is a python list of all the command line arguments.

Using dynamic scope rules, SPSS will behave very much like SPS except that there is no need to use `dict` operations in programs. My interpreter for dynamic SSPS was actually a little smaller than my original SPS interpreter because I no longer had to detect the `dict`, `begin`, and `end` operations. Instead, each time I was about to call a procedure I unconditionally pushed a new dictionary and when a procedure was about to return I popped the dictionary. I suggest that you first adapt your SPS interpreter along these lines.

To implement static scope rules you need a *static chain* which is the set of dictionaries visited by following the *static links* (also called the *access links*) we discussed in class. You already have a stack of dictionaries, probably implemented as a list, so you don't have explicit *control links* (also called *dynamic links*). The dynamic chain is implicit in the order of the dictionaries in the list – you search from one end (the warm end) and push and pop at that end as well. How can you implement the static chain? I suggest making the stack be a stack of tuples instead of just a stack of dictionaries. Each tuple contains an integer and a dictionary. The integer is the *static link* that tells you the position in the list of the (dictionary, static-link) tuple for the parent scope.

Where do static-link values come from? As we saw in class, at the point when a function is called the static link in the new stack entry needs to be set to point to the stack entry where the function definition was found. (Note that with the stack being a list, this “pointer” is just an index in the list.) So when calling a procedure you create a new dictionary automatically (as in the dynamic case) but what you push on the dictionary stack is now the pair (dictionary, index-of-definition's stack entry). *Hint 1:* in my implementation this all took only a handful of lines of new code but it was tricky to get all the pieces right. My advice is think more, write less for this part of the project. *Hint 2:* This is much easier if you put the warm end of your dictionary stack at the end of the list rather than the beginning. *Note:* If you want to explore python's object facilities and create a class for the objects on the dictionary stack feel free. The code will be a little cleaner, but using objects is not required as we have not discussed python objects yet.

As discussed in class, variable lookups now proceed by looking in the current dictionary at the top of the dictionary stack and then following the `static-link` fields to other dictionaries (instead of just looking at all the dictionaries on the stack in turn, which gives dynamic scope rules).

Output of the interpreter

The output of the SPSS interpreter consists of the contents of the operand **and dictionary** stacks whenever the `stack` operation is executed. Print the operand stack one value per line **and the contents of the dictionary stack one name and value per line** with a line containing “----” between different dictionaries on the dictionary stack. Remember please the difference between a *dictionary* and a *dictionary entry*.

What if my SPS interpreter didn't work correctly?

You may start from the skeleton found at <http://www.eecs.wsu.edu/~hauser/cs355/handouts/sps.py>; but if your code was almost correct it will probably be easier to work with your own code. Note that the skeleton does not meet all the requirements of the sps assignment and you need to first complete it to do so. It is missing at least *false*, *true*, and the boolean operations *and*, *or*, and *not*.

How can I tell if my static scoping code is working correctly?

You will have to create some test cases for which you can predict the correct answers. We covered one simple example in class. Translated to SSPS and simplified a bit that example looks like:

```
/x 4 def
/g { x } def
/f { /x 7 def g } def
f
```

which will leave 7 on the stack using dynamic scoping and 4 using static scoping. Remember that testing to the specification is your responsibility. We read your code looking for bugs in addition to running it on some tests. Tests can only show the presence of bugs, never their absence!