

Project 4

CptS 355

Spring 2006

22nd March 2006

Assigned: 27 March, 2006

Due: Monday, Apr. 10, 2006, 11:59:59PM. Develop all your code in a directory named “four”. 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 the first assignment. 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. Do not work with other students on it: if you have a question or are stumped see the instructor or the TA.

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.

To implement static scope rules you need a *static chain*. You already have a stack of dictionaries, probably implemented as a list. 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? At the point where a procedure is defined (*i.e.*, when doing a `def` for a block of code) you need to remember the position in the list of the current top-of-stack along with the definition of the procedure. At the point where a procedure is defined the current top-of-stack corresponds to the enclosing scope. Why? 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-parent's stack entry). *Hint*: 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. *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 `staticLink` fields to other dictionaries (instead of just looking at all the dictionaries on the stack in turn, which gives dynamic scope rules).

Note on interpreter input

Please arrange things so that your interpreter reads its SPSS input from the *standard input*.

That is, it must be possible to run your interpreter on file `input.ps` by executing

```
python ssps <input.ps
```

If all the interpreters handle input the same way it makes things easier for us when we evaluate your work.

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*.

- (Note the new requirement for a `stack` command and the requirement to print the contents of the dictionary stack.)

What if my SPS interpreter didn't work correctly?

You will need to complete it for this project.

There is not much point in doing this project if procedure calling doesn't work in your sps implementation, so *ask for help*. As usual, this project is to be **your personal, individual work**.

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!