Programming Language Concepts

CSCI-344 **Programming 2**
Term 20241

Impcore with Locals

1 Introduction

You will modify the Impcore interpreter to support functions with local variables in order to gain experience with a real interpreter.

2 Description

Complete Exercise 30 of Chapter 1 from *Programming Languages: Build, Prove, and Compare* (p. 86). The exercise asks you to modify the Impcore interpreter to support local variables. The new concrete syntax for function definitions is as follows:

`(define` *function-name* `(`*formals*`)` [`(locals` *locals*`)`] *expression*`)`

where *locals*, having the same syntax as *formals*, names the function's local variables. And, like *formals*, there may be no duplicate names in *locals*; however, the same name may appear in both *formals* and *locals*.

Local variables can alleviate the need for global variables or extra formal parameters to store temporary results; for example, an iterative factorial function can be now be written as follows:

```
(define fact (n) (locals ans)
  (if (< n 0)
    -1
    (begin
      (set ans 1)
      (while
        (> n 1)
        (begin
          (set ans (* ans n))
          (set n (- n 1))))
      ans)))
```

2.1 Interpreter Source Code

Source code for the interpreter is available on the CS Department file system at:

`/usr/local/pub/mtf/plc/programming/prog02-impcore-locals/impcore-with-locals`

and packaged as an archive at:

`/usr/local/pub/mtf/plc/programming/prog02-impcore-locals/impcore-with-locals.tar`

Note that this source code contains just the C code from the textbook and supplement, with simple comments identifying page numbers. There is a `Makefile` for building the interpreter.

Relative the original Impcore interpreter, there are new versions of `all.h`, `definition-code.c`, `parse.c`, `printfuns.c`, and `tableparsing.c`. Executing

```
$ diff -u /usr/local/pub/mtf/plc/src/bare/impcore /usr/local/pub/mtf/plc/src/bare/impcore-with-locals
```

will display the differences.

Copy the interpreter source code to a local directory and make modifications to your local copy; for example, executing

```
$ tar xvf /usr/local/pub/mtf/plc/programming/prog02-impcore-locals/impcore-with-locals.tar
```

will copy the interpreter source code to a new local directory named `impcore-with-locals`.

3 Requirements and Submission

Modify the interpreter as necessary to support local variables.

Write a `README.txt` file. Your `README.txt` file should be formatted as follows:

```
Name: ... name ...
Time spent on assignment: ... time spent ...
Collaborators: ... list of collaborators ...

... description of your solution to the problem ...
```

Submit all modified files and `README.txt` to the `Programming 02` Assignment on myCourses by the due date.

4 Hints

- Think about how to update the operational semantics to handle local variables. In particular, think about which judgements would need to change and which rules would need to change.
- You should examine `all.h` and the definition of `struct Userfun` to understand the revised abstract syntax of function definitions. (Since the provided source code has made the necessary changes to the abstract syntax, the first sentence of the penultimate paragraph of Exercise 30 does not apply to this assignment; i.e., the definitions of `struct Userfun` and function `mkUserfun` **do not** have to change.)
- You will need to modify `eval.c` to give semantics to local variables.
- You will need to modify `parse.c` to check that names of local variables are not duplicated.
- You may need to modify `all.h` to change the prototypes of existing functions and/or to add prototypes of new functions.
- You may need to modify `imptests.c` to change the evaluation of expressions for `check-expect` and `check-error`.
- You will **not** need to modify `parse.c` or `tableparsing.c` to parse the new concrete syntax into the revised abstract syntax; the provided source code has made the necessary changes to the parser to recognize local variables and store them in the abstract syntax tree. (Since the provided source code has made the necessary changes the parser, the first sentence of the penultimate paragraph of Exercise 30 does not apply to this assignment; i.e., the definition function `reduce_to_xdef` **does not** have to change.)
- There are at least two different strategies for solving this problem.
- You will not need to write very much code, but you will need to read (and understand) a significant portion of the interpreter in order to know the right code to write.