Vanessa Chambers
CS51 Final Project
8 May 2019

For my miniml Ocaml interpreter, I added two extensions, one that handles expression evaluation in lexical environments, and one that handles float evaluations (vs integer operations).

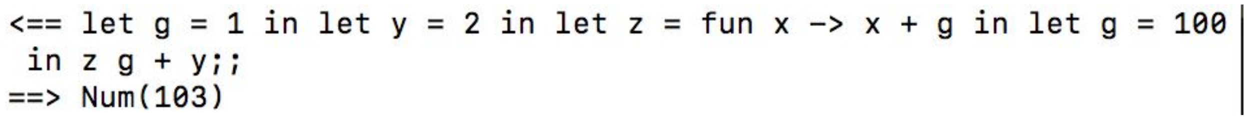
Lexical Evaluation

Evaluating an expression in a lexical environment has a very similar result to regular substitution evaluation, however, lexical programming uses environments which have closures. Indicated by the word “close,” lexical environments store values and their keys inside of a closure as soon as they are called. For example, with this function :

let g = 1 in let y = 2 in let z = fun x -> x + g in let g = 100 in z g + y

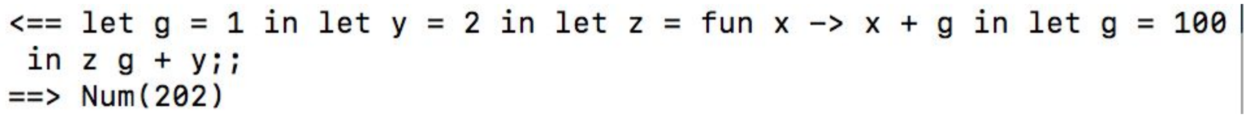
g, y, and z are stored in and environment with respective closures (g = 1), (y = 2), and (z, fun x -> x +g) until redefined -- g, y, and z can be thought of like keys, and 1, 2, and “fun x -> x + g” can be thought of as values). These closures sort of lock expressions into their definitions as soon as they are defined. Therefore, if I update g=1 to g=100, the function “z = fun x -> x + g” is still “locked” into its former definition, when g was 1. So, unless I redefine the function z after updating g, it will still refer to this g inside of the old environment it is stored in. Calling this function in a lexical vs dynamic environment will yield two different results:

Lexical Environment:



```
<== let g = 1 in let y = 2 in let z = fun x -> x + g in let g = 100
 in z g + y;;
==> Num(103)
```

Dynamic Environment:

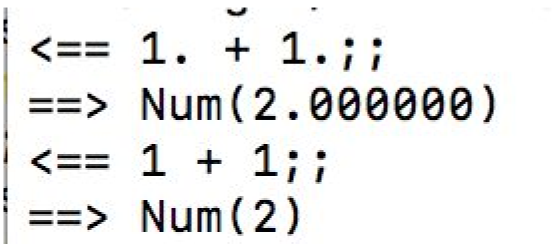


```
<== let g = 1 in let y = 2 in let z = fun x -> x + g in let g = 100
 in z g + y;;
==> Num(202)
```

The results of these two equivalent expressions are different because the function was defined in an an environment with g equal to 1, so when the function is called, it refers to this g. In a dynamic environment, functions are not evaluated until they are called, so I could theoretically update g as many times as I’d like before the function call, and the function would use the freshly updated value for g.

Float Evaluation

I additionally allowed my miniml to evaluate floats. Originally, the miniml could only evaluate integers and could only perform binary operations on integers. I extended this to allow for float operations.



```
<== 1. + 1.;;
==> Num(2.000000)
<== 1 + 1;;
==> Num(2)
```