HW 4 — Variables and Parameters

CS 421 Spring 2016
Revision 1.0

Assigned April 21, 2016
Due April 28, 2016 11:59:59pm

1 Change Log

1.0 Initial Release

2 Objectives

Your objective for this assignment is to understand the details behind different variable scoping techniques and parameter passing styles.

3 Handing In

You should commit and push a PDF of your solution to your student repository in the `hw4-variables` folder and name your solution file `hw4-submission.pdf`. We have also given you a starter file which you can find in your student repository.

To commit and push your work:

```
git add hw4-submission.pdf; git commit -m "submitting hw4"; git push
```

4 Problems

1. Static vs Dynamic Scoping The following piece of code would implement `eval` for expressions of the form `e1 e2` (i.e., application expressions). However, a section of the code is missing and has been replaced with a comment.

```
import Data.HashMap.Strict

eval (AppExp e1 e2) env =
  let CloVal (param body cenv) = eval e1 env
    in eval body $ {- YOUR TASK -}
```

Here, the type of `env` is `HashMap String Val`. You may use `insert` from the `HashMap` module.

Fill in the missing portion of the code to implement:

(a) **Static scoping**

(b) **Dynamic scoping**

2. Call By Value, Reference, and Result Consider the following code which is written in H++, a hybrid of Haskell and C. Its syntax is very Haskellesque, except it allows for variables to be re-assigned, and for `printf` to be used freely. `:=` is an assignment operator. Note that both `do` and `let` blocks execute code sequentially.

```
i = 10

foo a b c = do
   a := a * 2
   b := b * a + i
   c := a + b
   return b

main = let j = 10
           k = 10
           r = foo i j k
       in printf "%d %d %d %d" r i j k
```

What does the above code sample print out if our parameter passing style is:

Call By Value

Call By Reference

Call By Result

3. Call by Value, Name, and Need Consider the following code sample. Assume both `foo` and `bar` are functions defined elsewhere, and that we don't care what they do, *except* that they don't call themselves or each other.

```
baz x y =
   x + x + y + y

main = printf "%d " (baz (foo 5) (bar 10))
```

How many times do each of `foo` and `bar` run for the following parameter passing styles? (Write down two numbers for each style, one for `foo` and one for `bar`. Label each number appropriately.)

Call By Value

Call By Name

Call By Need

4. Pick the correct parameter passing style We want to write a function called `doubleOrNothing` that takes a `guard`, a `body`, and a `default`, representing a boolean guard and two integer values, respectively. If `guard` is True, return the result of adding `body` to itself. Otherwise, return `default`.
(Yeah, we know it's terrible code. We're just trying to make a point about parameters.)

```
doubleOrNothing guard body default =
   if guard then body + body else default
```

We then run the following:

```
fact n = n * (doubleOrNothing (n>0) (fact (n-1)) 1)

main = printf "%d" (fact 3)
```

Consider what happens when we run that code over the following parameter passing styles. How many times does `fact` get called for each style? (Recall ∞ is a proper value in our world.)

Call By Value

Call By Name

Call By Need