Notes on Implementing Anonymous Functions

Lambdas

In the function language world, anonymous functions are often known as *lambdas*. Anonymous functions are on-the-fly function definitions and can appear anywhere an expression can appear. Typically lambdas are used to define small functions without polluting the namesapce (since anonymous functions have no names). Lambdas are most often passed to other functions or returned from functions.

An easy way to implement lambdas starts at parsing. Your grammar rule for lambdas should appear as a unary:

```
unary : ... |
    | LAMBDA OPAREN optParamList CPAREN block |
    | ... |
```

Let’s also suppose your parse tree for a function definition looks like this:

```
FUNCDEF
    / \
    ID GLUE
    / \
    optParamList block
```

You should produce a similar parse tree for the lambda:

```
LAMBDA
    / \
    NULL GLUE
    / \
    optParamList block
```

The only difference is the tag at the root of the tree and the null pointer where the function name used to appear.

Since a lambda definition now looks like a function definition, we can make a closure out of it in exactly the same was as with a function definition. The only difference is that the lambda evaluator just returns the closure; it does not store it in the current environment:

```
function evalLambda(tree, env)
{
    return cons(CLOSURE, env, tree);
}
```