Simple, but ugly

Here is a simple way to add built-in functions to your language. It’s ugly, but it works. For example, let’s add a `println` built-in. One does so by adding some tests to the `evalCall` function. Originally, this function looks like:

```javascript
function evalCall(t, env)
{
    // this code assumes a function call of the form f(x,y)
    var name = getCallName(t);
    var args = getCallArgs(t);
    var eargs = evalArgs(args, env);
    var closure = eval(name, env);
    var params = getClosureParams(closure);
    var body = getClosureBody(closure);
    var senv = getClosureEnvironment(closure);
    var xenv = EnvExtend(senv, params, eargs);

    return eval(body, xenv);
}
```

We modify the function to check for a call to the built-in function:

```javascript
function evalCall(t, env)
{
    // this code assumes a function call of the form f(x,y)
    var name = getCallName(t);
    var args = getCallArgs(t);
    var eargs = evalArgs(args, env);
    // check for built-in functions here
    if (stringEquals(name, "println"))
        return evalPrintln(eargs);
    else
    {
        var closure = eval(name, env);
        var params = getClosureParams(closure);
        var body = getClosureBody(closure);
        var senv = getClosureEnvironment(closure);
        var xenv = EnvExtend(senv, params, eargs);

        return eval(body, xenv);
    }
}
```

and dispatch to the appropriate handler for the built-in. This code assumes that functions are called by name only. Finally, add the `evalPrintln` function:

```javascript
function evalPrintln(evaluatedArgList)
{
    while (evaluatedArgList != null)
    {
        displayLexeme(car(evaluatedArgList));
        evaluatedArgList = cdr(evaluatedArgList);
    }
}
```

You can perform similar actions for each of your built-ins.
A better way

When we create our initial global environment, we can seed it with all our built-in functions:

```c
env = createEnv();
insertEnv(env,new Lexeme(ID,"println"),new Lexeme(BUILTIN,evalPrintln));
...
```

```c
eval(parseTree,env);
```

Note that the BUILTIN lexeme is passed the evaluation function associated with the name of the built-in function. In C, we would augment our Lexeme structure with an `fval` field:

```c
struct Lexeme
{
    char *type;
    int ival;
    char *sval;
    Lexeme **aval;
    Lexeme *(*fval)(Lexeme *); //fval is a pointer to a function
    Lexeme *left;
    Lexeme *right;
}
```

The `fval` field accepts the address of a function that accepts a lexeme pointer as an argument and returns a lexeme pointer as a result, exactly the signature of `evalPrintln`.

Now in `evalCall`, we can dispatch to the built-in evaluation function directly:

```c
function evalCall(t,env)
{
    //this code assumes a function call of the form f(x,y)
    var name = getCallName(t);
    var args = getCallArgs(t);
    var eargs = evalArgs(args,env);
    var closure = lookup(name,env);
    //check for built-in functions here
    if (closure.type == BUILTIN)
        return closure.fval(eargs);
    else
    {
        var params = getClosureParams(closure);
        var body = getClosureBody(closure);
        var senv = getClosureEnvironment(closure);
        var xenv = EnvExtend(senv,params,eargs);

        return eval(body,xenv);
    }
}
```

Since most modern languages allow the passing of function pointers, this method is the preferred approach for handling built-in functions.