

LISP Lab 2
CS411 – Dr. Shaffer
Due: October 25, 2004

1 Cleaning up

1.1 LISP symbol or Smalltalk symbol?

As you inspect and print values produced by your parser it is sometimes hard to tell if the thing you are looking at is a Smalltalk ByteSymbol instance or a LISPSymbol instance. The reason this is so is because LISPSymbols hold onto ByteSymbol instances as the symbols “value” and when a LISPSymbol is told to “print itself” it prints its ByteSymbol value. Fix this by modifying LISPSymbol’s `printOn:` method to prefix the LISPSymbol with “L_”. Do the same with LISPInteger instances. Verify that it is working by inspecting the results of parsing “(1 2 a b c)”.

1.2 nil is a special kind of atom

After Lab1 you can build “cons-es” but they end in a non-LISP object (Smalltalk nil). That means that the `cdr` of the end of the list is not an instance of `LISPAtom` but, instead, an instance of `UndefinedObject` (nil’s class). That’s no good.

Create a new subclass of `LISPAtom` called `LISPNil`. Implement the singleton pattern (create one special instance of `LISPNil` and provide a method to get to it). Later on it will be useful to test if an object is nil so add the method `#isLISPNil` to return true and add it to `LISPCons` and `LISPAtom` to return false.

Modify `LISPCons` so that it initializes its `car` and `cdr` to your special instance of `LISPNil` instead of Smalltalk nil. Modify the `LISPCons`’s `printOn:` method to test using `isLISPNil` instead of `isNil`.

1.3 Quoted S-Expressions

In most LISP’s the single quote (or apostrophe) is used to generate a quoted expression. For example, `'a` is equivalent to `(quote a)` and `'(1 2 3)` is equivalent to `(quote (1 2 3))`. Modify your grammar to support this form directly. That is, translate `'a` to `(quote a)`.

1.4 Pretty printing lists

Look at the `#printOn:` method in `LISPCons` to see how lists are printed. Make a new method, `lispPrintOn:`, which prints lists following the normal LISP (and Scheme) “pretty printing” conventions. That is the list `(1 2 abc dog)` should print itself as `(1 2 abc dog)` instead of printing using the dotted pair notation. Also, it shouldn’t

print the `L_#` generated by the normal `printOn:` method. This is going to require some reorganization. First, all of your LISP classes need to have a common superclass (`LISPObjecT` makes sense as a name). Second, study the implementation of `printString` in `Object` to see how we print things in Smalltalk (you create a stream and then send `printOn:`). Do the same thing for pretty printing: make methods `LISPObjecT>>lispPrintString` and `LISPObjecT>>lispPrintOn:` and the override `lispPrintOn:` in the various subclasses.

2 Testing your code

2.1 Writing test cases

We will use the SUnit test framework to develop test cases for our system. You have run SUnit tests when completing your `ALGraph` assignment. To develop your own test case, create a subclass of `TestCase` (browse that class to see what it provides). Any method whose name begins with the word “test” is automatically part of your test suite. There are several sample tests included with SUnit (again, browse `TestCase` and look at its subclasses). Read through a few of those to get you started.

To run your tests either open a `TestRunner` (by evaluating “`TestRunner open`”) or load the “`RB SUnit Extensions`” parcel which adds the test runner buttons to your browser (like we did in the `ALGraph` project).

2.2 Testing the parser

Create a class to hold parser tests (how about `LISPParserTest?`). Here is an example test method to test parsing of literal ints.

```
testLiteralInteger
  self assert: ((LISPParser on: '45' readStream) parse isKindOf: LISPInteger).
  self assert: (LISPParser on: '45' readStream) parse value = 45.
  self assert: (LISPParser on: '1548' readStream) parse value = 1548.
  self assert: (LISPParser on: '-12' readStream) parse value = -12.
```

Write test cases for each language construct supported by your parser (literal integer, literal symbol, nil, list, a list containing a list, pretty printing etc).