

## Homework 4: Converting to Strings

- Preliminaries
- Introduction
- Problem
- Warning!
- Extra Credit
- Turning it In

### Preliminaries

**Assigned:** Friday, 2 March 2001

**Due:** Friday, 9 March 2001

*No extensions without prior permission!*

**Summary:** In this assignment, you will write a procedure that can convert an arbitrary value to a string.

**Purpose:** To enhance your skills with recursive procedures; to help you think more carefully about the different types of data; to help you think about the differences between pairs and lists.

**Collaboration:** Each student must write his or her own code for this assignment. However, you may certainly discuss possible structures to the solution with each other. You may also call upon others to help you correct or improve your code. If you receive help from others, make sure to document that help.

**Time:** I try to keep the core of my assignments under eight hours (extra credit may, of course, take extra time). When you find that you've already spent five hours on the assignment, please let me know.

### Introduction

Now that we make it a careful habit to verify preconditions, we must also provide useful and clear reports when those preconditions are not met. For example,

```
;;; Precondition:
;;; val is a number
...
(define foo
  (lambda (val ...)
    (if (not (number? val))
        (error "foo" "val must be a number")
        ...)))
```

However, when we read error messages, it is often more helpful to know what value we provided so that we can think about why that caused an error. Compare the following error messages.

```
> (foo 'a)
foo: val must be a number
> (bar 'a)
bar: expected a number, received 'a'
```

Most people find the second error message significantly more helpful. Now, the code we'd like to write to generate the second error should be rather straightforward:

```
(error "foo"
      (string-append "expected a number, received '"
                    val
                    "'"))
```

However, if we try that, we'll often get an error that says

```
string-append: expects argument of type <string>; given a
```

How can we get around that error? We can write a procedure that converts any value to a string. Suppose we called that procedure `value->string`. Here's what we might write for the desired error message.

```
(error "foo"
      (string-append "expected a number, received '"
                    (value->string val)
                    "'"))
```

## Problem

Your goal is to document, write, and test the `value->string` procedure used above. Your procedure should take any one value as a parameter and return that value as a string. Here's a start:

```
(define value->string
  (lambda (val)
    (cond ((symbol? val) (symbol->string val))
          ((number? val) (number->string val))
          (...))))
```

You'll find that it's fairly easy to deal with the basic types and much more complicated to deal with compound types, such as lists and pairs. One of the key intents of this assignment is for you to work out how to deal with lists and pairs.

Here are some sample applications of `value->string`.

```
> (value->string 'a)
"a"
> (value->string "hi")
"hi"
> (value->string 22/7)
"22/7"
> (value->string list)
"#<procedure>"
```

```
> (value->string (list 'a 'b))
"(a b)"
> (value->string (cons 'a 'b))
"(a . b)"
```

## Warning!

There are other ways to solve the problem discussed in the introduction. Your goal in this assignment is not so much as to solve the "How do I print nice error messages?" problem as to solve the "How do I convert an arbitrary value to the appropriate string?" problem.

## Extra Credit

Those of you with extra time, extra enthusiasm, or extra desire for a high grade may wish to do more on this assignment. While I generally think it's up to you to come up with extensions, here are a few simple possibilities:

- Print out the name of the procedure if it's a procedure.
- Learn about other data types by reading the Scheme report and deal with those other data types. (A vector is one common data type we have not yet covered.)

## Turning it In

Upload your modified .ss files to the Blackboard dropbox.