Creating a Library

1. Create a subdirectory named discrete within the course directory you created in the “Setting Up DrRacket for R7RS Programming” lab.

2. Copy the file

/home/reseda/discrete-structures/code/discrete/utilities.ss

into your new discrete subdirectory, giving it the name my-utilities.ss.

3. Open a DrRacket window and use it to load in your copy of the utilities library — the file you created in the preceding exercise.

4. In the define-library-expression, the first subexpression specifies the library being defined. Change this specification so that the library will be called (discrete my-utilities).

5. Some of the procedures used in the CSC 151 course at Grinnell are not predefined in R7RS Scheme. I have created a library called (csc-151 collection) providing implementations of these otherwise missing procedures. In the Definitions subwindow, revise the import-expression so that it imports this library as well as (scheme base) and (scheme write). Click on the Run button again and confirm that the double procedure from the (csc-151 collection) library is now available in the Interactions subwindow.

6. In order to see what procedures are available in (csc-151 collection), you should look at the export-expression at the beginning of the source code for that library. In what MathLAN directory would you look for that source code?

7. One of the mathematical notations that we’ll encounter occasionally this semester is summation:

$$\sum_{i=m}^{n} f(i)$$

This is a shorthand way to refer to the sum of the values $f(m), f(m+1), \ldots, f(n-1), f(n)$ — the values of the function $f$ for arguments from $m$ up to and including $n$.

For example, to refer concisely to $7^2 + 8^2 + 9^2 + 10^2 + 11^2 + 12^2 + 13^2$, one could write

$$\sum_{i=7}^{13} i^2.$$  

A summation is like a loop in which the loop-control variable, $i$, takes on successive integer values from $m$ up to $n$. On each iteration of the loop, the result of applying the function to $i$ is computed and added to a running total. The value of the summation is the final value of the running total.

A summation in which $m$ and $n$ are equal has only one term, $f(m)$, and in this case that one term by itself is the value of the summation. (This is like a loop in which the body is executed only once.)

A summation in which $m$ is greater than $n$ has no terms at all, and in this case the value of the summation is 0. (This is like a loop in which the body is never executed at all, because the entry condition fails as soon as it is evaluated.)
Extend the \textit{(discrete utilities)} library so that it includes a procedure named \texttt{summation} that takes three arguments, corresponding to \(f\), \(m\), and \(n\) in the description above, and computes and returns \(\sum_{i=m}^{n} f(i)\). It is a precondition of this procedure that the arguments \(m\) and \(n\) are exact integers. A second precondition is that the argument \(f\) is a unary procedure that can receive any integer value as argument and yields a single numerical result.

8. Experiment with your \texttt{summation} procedure in the Interactions window. What are some easy-to-check cases? Are there any exceptional or boundary cases that need to be checked?

9. Explain why \(f\), \(m\), and \(n\) must be parameters of the \texttt{summation} procedure while \(i\) is not.

10. Arrange for the binding created by the definition of \texttt{summation} to be exported along with \texttt{assert}, \texttt{cube}, \texttt{natural-number?}, and \texttt{factorial}.

11. Save the extended version of the library back into the \texttt{my-utilities.ss} file.

\textbf{Importing}

12. Write a top-level program that imports the \texttt{(discrete my-utilities)} library and computes and reports, for each natural number \(k\) less than or equal to 25, the sum of the cubes of the natural numbers up to and including \(k\).

13. R7RS Scheme does not provide any framework analogous to RackUnit for unit testing of procedures. I have supplied a \texttt{(discrete testing)} library that provides somewhat similar functionality. It defines two new kinds of expressions, the \texttt{test-expression} and the \texttt{suite-expression}.

A \texttt{test-expression} is an S-expression that contains four or five subexpressions: (1) an identifier indicating the nature of the test to be conducted; (2) an expression to be evaluated — the “results of the test” are the values of this expression; (3) a numeral indicating how many results the test should produce; (4) an S-expression containing one subexpression for each expected result of the test; the value of each of these subexpression should be a predicate, which the corresponding result is required to satisfy; (optionally, 5): an expression of which the value is a predicate, to be applied to all of the results (as arguments in one call), which those results are required to satisfy.

For instance, here is a \texttt{test-expression} that checks whether the \texttt{square} procedure returns the correct value for the argument \(-5\):

\begin{verbatim}
(test negative-argument
  (square -5) ;; Evaluating this expression ...
  1 (number?) ;; ... should produce one result, a number ...
  (lambda (result)
    (= result 25))) ;; ... that is equal to 25.
\end{verbatim}

Evaluating a \texttt{test-expression} runs the test. When a test succeeds, the \texttt{test-expression} returns \#t. When the test fails, the \texttt{test-expression} prints out a diagnostic message giving the name of the failed test, the expression that gave the wrong answer, and the value(s) that actually resulted from the evaluation of that expression. It then returns \#f.

A \texttt{suite-expression} that comprises (1) an identifier that serves as the name of the test suite; (2) an S-expression containing zero or more binding specifications (as in a \texttt{let-expression}); and (3) zero or more \texttt{test-expressions}.

Evaluating a \texttt{suite-expression} runs all of the tests in an environment extended by the binding specifications and reports the results.

Study the unit tests for the \texttt{(csc-151 collection)} library in the program

\begin{verbatim}
/home/reseda/discrete-structures/code/tests/test-collection.ss
\end{verbatim}

Run that program and examine the output. Then write at least one additional test that might usefully be added to one of the suites there.
14. Write a top-level program that imports the \texttt{(discrete my-utilities)} library, tests the correctness of the \texttt{summation} procedure, and reports on any tests that fail.