



University of Arizona, Department of Computer Science
CSc 372 — Assignment 2 — Due noon, Wed Sep 15 — 3%
Christian Collberg
September 8, 2004

1 Introduction

The purpose of this assignment is for you to practice writing recursive functions over lists.

Before starting this assignment, set your DrScheme language level to **Standard (R5RS)**.

All your function definitions should be *pure*, i.e. they should not use any of Scheme's imperative features such as `set!`. Also, never use iteration, always recursion.

Every function should be commented. At the very least, the comments should state what the function does, which arguments it takes, and what result it produces.

You may make use of the standard list-manipulation functions `car`, `cdr`, `cons`, `eq?`, and `equal?`. However, you may **not make use of the more complex built-in functions such as `append`, `reverse`, etc.** If you find you need these functions you should define them yourself.

You should make your functions as simple and elegant as possible by introducing appropriate helper functions.

You will be graded primarily on **correctness**, **elegance**, **clarity**, and **documentation**.

Functions which you have defined for one problem can be used to solve subsequent problems.

This assignment is graded out of 100. The Scheme assignments are worth a total of 10% of your final grade. This assignment is worth 3% of your final grade.

2 Simple Functions

1. Write a function (`prod A`) which computes the product of a list of numbers. For this problem you can safely assume that the list only contains numbers and has at least one element. The function should have the following behavior: [10 points]

```
> (prod '(1))  
1  
> (prod '(1 2))  
2  
> (prod '(1 2 3 4))  
24
```

2. Write a function (`double A`) which takes a list

$$a = (a_1, a_2, \dots, a_n)$$

of length n as argument and returns a list of length $2n$:

$$b = (a_1, a_1, a_2, a_2, \dots, a_n, a_n)$$

The function should have the following behavior:

[10 points]

```
> (double 5)
illegal argument
> (double '())
()
> (double '(a))
(a a)
> (double '(1 2 3))
(1 1 2 2 3 3)
> (double '((1) 2 (3 4)))
((1) (1) 2 2 (3 4) (3 4))
> (double (double '(1 2 3)))
(1 1 1 1 2 2 2 2 3 3 3 3)
```

3. Write a function (`fibonacci? A`) which returns `#t` if the list `A` is a valid Fibonacci sequence, i.e. if $a_i = a_{i-2} + a_{i-1}$, for all $i > 3$. Print an error message if the list has fewer than three elements. The function should have the following behavior:

[10 points]

```
> (fibonacci? '(1 1))
the argument list should have at least three elements
> (fibonacci? '(1 1 2))
#t
> (fibonacci? '(2 2 4))
#t
> (fibonacci? '(3 4 7 11 18))
#t
> (fibonacci? '(2 2 4 5))
#f
```

4. Write a function (`palindrome? A B`) which returns `#t` if reversing the list `A` yields the list `B`. You can assume that the lists contain only symbols and numbers. The function should have the following behavior:

[10 points]

```
> (palindrome? '() '())
#t
> (palindrome? '(a) '(a))
#t
> (palindrome? '(a b c) '(c b a))
#t
> (palindrome? '(a b c) '(c b d))
#f
> (palindrome? '(a b c) '(c b))
#f
> (palindrome? 'a '(c b))
illegal argument
```

5. Write a function `zip` which takes two lists

$$a = (a_1, a_2, \dots, a_n)$$

and

$$b = (b_1, b_2, \dots, b_n)$$

of equal lengths and produces a new list of pairs c where the elements of the i :th pair is (a_i, b_i) :

$$c = ((a_1, b_1), (a_2, b_2), \dots, (a_n, b_n)).$$

An error message should be printed if the arguments are not lists or are lists of unequal length. The function should have the following behavior: [10 points]

```
> (zip '() '())
()
> (zip '(a) '(1))
((a 1))
> (zip '(a b) '(1 2))
((a 1) (b 2))
> (zip '((a) (b)) '(1 2))
(((a) 1) ((b) 2))
> (zip 1 '(a b))
arguments must be lists
> (zip '(1) '(a b))
lists must be of equal length
```

3 A Set Library

Write a library of set-manipulation functions. A set is represented as a list such that

1. the *order* of elements in the list does not matter;
2. the list contains no nested lists;
3. the list contains no duplicate elements.

You should provide the functions `set?`, `set-make`, `set-union`, and `set-intersection` as defined below.

1. The `set?` function should return `#t` if its argument is a valid set, according to the definition above. The function should have the following behavior: [10 points]

```
> (set? '())
#t
> (set? '(a))
#t
> (set? '(a b))
#t
> (set? '(b a))
#t
```

```

> (set? '(1 2 a))
#t
> (set? '(1 2 (a)))
#f
> (set? '(1 2 1))
#f

```

2. The function `set-make` takes a nested list (possibly with duplicate elements) and constructs a set by flattening the input list and removing duplicates. The function should have the following behavior:[20 points]

```

> (set-make '())
()
> (set-make '(a))
(a)
> (set-make '(a a))
(a)
> (set-make '(a b))
(a b)
> (set-make '(()))
()
> (set-make '((a)))
(a)
> (set-make '((a) (b (c))))
(a b c)
> (set-make '((a) (b (a))))
(a b)

```

3. The `set-union` function computes the union of two sets. If either argument is not a valid set the function should fail with an error message. The function should have the following behavior:[10 points]

```

> (set-union '() '())
()
> (set-union '(a) '())
(a)
> (set-union '(a) '(b))
(a b)
> (set-union '(a) '(a))
(a)
> (set-union '(a b c) '(1 2))
(a b c 1 2)
> (set-union '(a b c) '(1 a))
(a b c 1)
> (set-union '(x 1) (set-union '(a b c) '(1 a)))
(x a b c 1)
> (set-union '(a) '(b b))
illegal argument: (b b) is not a set

```

4. The `set-intersection` function computes the intersection of two sets. If either argument is not a valid set the function should fail with an error message. The function should have the following behavior: [10 points]

```

> (set-intersection '() '())

```

```
()
> (set-intersection '(a) '())
()
> (set-intersection '(a) '(b))
()
> (set-intersection '(a) '(a))
(a)
> (set-intersection '(a b c) '(1 b c))
(b c)
> (set-intersection '(b) (set-intersection '(a b c) '(1 b c)))
(b)
> (set-intersection '(a) '(b (c)))
illegal argument: (b (c)) is not a set
```

4 Submission and Assessment

The deadline for this assignment is noon, Wed Sep 15. You should submit the assignment (a text-file containing the function definitions) electronically using the Unix command `turnin cs372.2 <files>`. This assignment is worth 3% of your final grade.

Don't show your code to anyone, don't read anyone else's code, don't discuss the details of your code with anyone. If you need help with the assignment see the instructor.
--