Principles of Programming, Spring 2009
Practice 5
Managing Hierarchical List and Iteration using Map - Application

Woosuk Lee, Suwon Jang and Seongkeun Cho
Programming Research Lab.@SNU

Oct 5, 2009

1. (SICP ex. 2.42) The "eight-queens puzzle" asks how to place eight queens on a chessboard so that no queen is in check from any other (i.e., no two queens are in the same row, column, or diagonal). One possible solution is shown in figure 1. One way to solve the puzzle is to work across the board, placing a queen in each column. Once we have placed $k-1$ queens, we must place the kth queen in a position where it does not check any of the queens already on the board. We can formulate this approach recursively: Assume that we have already generated the sequence of all possible ways to place $k-1$ queens in the first $k-1$ columns of the board. For each of these ways, generate an extended set of positions by placing a queen in each row of the kth column. Now filter these, keeping only the positions for which the queen in the kth column is safe with respect to the other queens. This produces the sequence of all ways to place k queens in the first k columns. By continuing this process, we will produce not only one solution, but all solutions to the puzzle.

   We implement this solution as a procedure `queens`, which returns a sequence of all solutions to the problem of placing n queens on an $n \times n$ chessboard. `queens` has an internal procedure `queen-cols` that returns the sequence of all ways to place queens in the first k columns of the board.

```
(define (queens bs)
  (define (queen-cols k)
    (if (< k 0)
        (list empty-b)
        (filter
         (lambda (p) (safe? p))
         (accumulate append
                     null
```

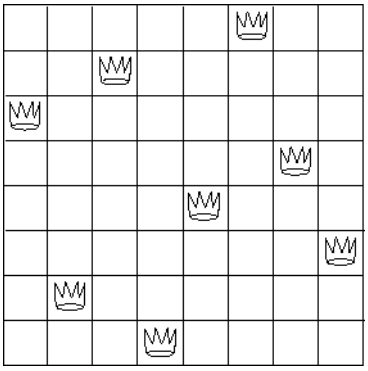

Figure 1: a solution to the eight-queens puzzle

```
                    (map <??>
                         (queen-cols (- k 1)))))))
  (queen-cols (- bs 1)))
```

In this procedure `empty-b` represents an empty set of positions. The procedure `safe?` determines for a set of positions, whether the queen in the kth column is safe with respect to the others. (Note that we need only check whether the new queen is safe – the other queens are already guaranteed safe with respect to each other.) Fill in the missing expressions in the above definition.

2. (http://mitpress.mit.edu/sicp/psets/ps3/readme.html) The Rivest-Shamir-Adleman (RSA) public key encryption algorithm works as follows: two very large prime numbers p and q are chosen and kept secret. Let $n = p*q$ (the idea is that it's hard to figure out what q and p are given n). Two more numbers, d and e, are picked so that $(e*d)$ mod $((p-1)*(q-1)) = 1$. The pair (n, e) is called the public key or encryption key, and is used by everybody to encrypt messages. The private key consists of the number d, which is kept secret. A message can be represented as a number m between 0 and n-1. To encrypt the message, compute m to the eth power mod n, in Scheme this would be written (remainder (expt m e) n). To decrypt a "cipher" message c, we use (remainder (expt c d) n). That is, if c = (remainder (expt m e) n), then (remainder (expt c d) n) will give us the original message m.

   For our excercise, the prime numbers are kept small (61 and 53), and e = 17, d = 2753. We encode text messages character by character using their ascii values. For example, "hello" is encoded with the list of ascii values:

   ```
   '(104 101 108 108 111)
   ```

   If we apply the encryption algorithm to each number in the list, we get the "ciphertext"

```
(2170 1313 745 745 2185)
```

Decrypting it will give us back the original list.

Scheme have no problems computing (expt c 157), which would be a bit more difficult for languages using only 32 and 64 bit integers.

You are to implement two functions: encrypt-list and decrypt-list that will encrypt and decrypt lists of values. I suggest you first define functions to encrypt/decrypt individual values.

```
(define p 61)
(define q 53)
(define n (* p q))
(define e 17)
(define d 2753)

(define pub_key (cons n e))
(define priv_key d)

; encrypt : integer * pub_key -> integer(ciphered)
(define (encrypt m pub_key)
   (<??>))

; decrypt : integer(ciphered) * priv_key -> integer(decrypted)
(define (decrypt c priv_key)
   (<??>))

; encrypt-list : integer list -> integer list
(define (encrypt-list l)
   (<??>))

; decrypt-list : integer list -> integer list
(define (decrypt-list l)
   (<??>))
```

I'm giving you the following function so that you can print a list of ascii values as a string:

```
; printasciis : integer list -> void
; convert integer list => character list, and print it
(define (printasciis l)
   (begin
    (map (lambda (x) (display (integer->char x))) l)
    (display "\n"))
     )
```

for example,

```
1 ]=> (printasciis '(104 101 108 108 111))
hello
```

Don't try to print the ciphertext.

Use your decryption program to decrypt the following top secret message:

```
(printasciis
 (decrypt-list
  '(3123 1632 690 1313 1992 3179 884 1992 2170 1632 612 612 1313 2235 1853)))
```

Swallow the message after you read it. You are sworn to secrecy!

Use your encryption algorithm to encrypt a message for other people in the class to decode. You can use the following function to transform a string to a list of ascii values:

```
; makeasciis : string -> integer list
; convert string => character list => integer list
(define (makeasciis S)
   (map (lambda (x) (char->integer x)) (string->list S)))
```

for example,

```
(makeasciis "hello")
```

returns

```
(104 101 108 108 111)
```