CSE 341: Homework 1

due 01/12/01

All functions you are asked to implement and turn in should be typed and tested with the scheme interpreter. With each function turn in a hardcopy of the source code listing as well as sample output for a demonstrative set of tests. Do not forget to test *end cases*. Also, use the `turnin` program to turn in an electronic copy of your source code.

Reading:

- Sethi: Chapter 1 (handout).
- *SICP: Chapter 1* (on the web).
- R^4RS (handout). This will be very useful for completing this assignment.

Part I: (Not to be turned in.)

Try these questions out before section. First try to figure out the answers in your head. Then try typing them into the scheme interpreter.

1. Scheme variables and list constructs.

   Given the following definitions:

   ```
   (define x 10)
   (define y 44)
   (define foo '(1 2 3))
   (define bar '(a b c))
   ```

   what are the results of the following pieces of scheme code:

   (a) `x`
   (b) `(cons x foo)`
   (c) `(cons y bar)`
   (d) `(list x y)`
   (e) `(list x foo)`
   (f) `'(x y)`
   (g) `(car bar)`
   (h) `(cdr bar)`
   (i) `(+ x y)`

2. Scheme flow control.

   What are the results of the following pieces of code?

   (a) `(if #f 5 10)`
   (b) `(if (or #t #f)  5 10)`

(c)
```
(if #t
    (if #f 2 3)
    10
)
```

(d)
```
(define x 3)
(cond ((= x 1)     44)
      ((= x 2)     (+ 5 50))
      ((= x 3)     (* 6 11))
      (else        (- 80 3))
)
```

(e)
```
(let ((x 5)
      (y 10)
     )
     (+ x y)
)
```

3. Write predicate `char-hex-letter?` which returns true if its single argument is one of the letters that can be in a hexadecimal number, "a" through "f", in either upper or lower case). Write predicate `char-hex?` which returns true if its single argument is a hexadecimal character (letter or digit). For example:

```
(char-hex-letter? #\b)          => #t
(char-hex-letter? #\3)          => #f
(char-hex? #\B)                 => #t
(char-hex? #\3)                 => #t
```

Part II: (Not to be turned in.)

These are some sample problems that we will be discussing in section. Use the `error` function for handling errors.

1. Write a function `char-hex->number` that converts a hexadecimal digit into a number (do not use the primitive `string->number` function). For example:

```
(char-hex->number #\b)          => 11
```

2. Write a recursive function `string-hex->number` that takes a hexadecimal string and converts it into a number (do not use the primitive `string->number` function).

```
(string-hex->number "3f")       => 63
```

3. Recursively define a function `positive-only` that takes a list and returns a list with the positive elements in it. For example:

```
(positive-only '())             => ()
(positive-only '(1 2 -3 4))     => (1 2 4)
(positive-only '(-1 -2 -3))     => ()
```

Part III: (Due Friday, 01/12/01. Turn this in.)

Complete the following. You do not have to handle errors in input gracefully; however, if you want to you may use the `error` primitive function; e.g. `(error "error in input")`. Do not forget to turn in a print out of your source code as well as example executions of your code on test cases.

1. (5 points) Write a function, `string-hex->char`, that takes a hexadecimal string and returns the character that it represents. For example, the character "a" is represented as decimal 97 in ASCII which is 61 in hex.

   ```
   (string-hex->char "61")          => #\a
   ```

2. (5 points) Write a function, `decode-hex-string`, that takes a string and converts it as follows:

   - "+" gets converted to whitespace.
   - "%XX", where `XX` is a two digit hex number, gets converted to the character that it represents.
   - All other characters are unchanged.

   For example:

   ```
   (decode-hex-string "In+hex+5+%2b+5+is+%27a%27.")
         => "In hex 5 + 5 is 'a'."
   ```

3. (5 points) Write a function `split` that takes two arguments, a character `ch` and a string `str`. It breaks `str` up into a list of strings at each occurrence of `ch`. For example:

   ```
   (split #\space "this is a test")
          => ("this" "is" "a" "test")
   ```