CMSC 10500-1: Homework 6

(due on Monday July 12th)

The Sieve Method

In a previous homework (and midterm), we saw how to check if a number was prime or not. This exercise, outlines a method to generate the list of all prime numbers lesser than a given number. This ancient method is also called the **The Sieve of Erosthenes**. The idea behind this method is very simple. Let `n` denote the number till which we want to find primes.

- Start with a list of all numbers from 2 to `n`.
- Mark 2 and cross out all multiples of 2.
- Mark 3 and cross out all multiples of 3.
- In general, mark the first uncrossed number you see and cross out all multiples of it.
- Eventually all numbers have been either marked or have been crossed out.
- The numbers which have been marked are precisely the prime numbers.

Lets start with an example. Take `n=22`. The following table shows the process. The **bold** numbers represent the marked numbers and the underlined numbers represent the crossed out numbers.

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
2	**3**	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
2	**3**	4	**5**	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
2	**3**	4	**5**	6	**7**	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
2	**3**	4	**5**	6	**7**	8	9	10	**11**	12	13	14	15	16	17	18	19	20	21	22
2	**3**	4	**5**	6	**7**	8	9	10	**11**	12	**13**	14	15	16	17	18	19	20	21	22
2	**3**	4	**5**	6	**7**	8	9	10	**11**	12	**13**	14	15	16	**17**	18	19	20	21	22
2	**3**	4	**5**	6	**7**	8	9	10	**11**	12	**13**	14	15	16	**17**	18	**19**	20	21	22

Hence we conclude that 2,3,5,7,11,13,17 and 19 are all the prime numbers lesser than or equal to 22.

Your task is to write a scheme function `primes` which consumes a number `n` and produces a list of all primes lesser than or equal to `n`. Accomplish this using the following outline.

1. **(4 points)**Write a function `interval` which consumes two integers `low` and `high` and returns a list containing all the numbers between (and inclusive) `low` and `high`.

```
> (interval 3 9)
(list 3 4 5 6 7 8 9)
```

2. **(4 points)** Write a function `sieve` which consumes a number `k` and a list of numbers `lon` and produces a list containing all the numbers in `lon` which are **not** a multiple of `k`.

```
> (sieve 3 (list 4 5 6 7 8 9))
(list 4 5 7 8)
```

3. **(5 points)** Write a function `sieve-helper` which consumes a list of potential prime numbers (`candidates`) and a list of known primes (`known`) and produces the list of all primes in `known` and `candidates`. You may assume that the first candidate (if any) is always a prime. Thus you enlarge the `known` list by adding the first candidate, and remove all multiples of the first candidate from the candidate list, and repeat this till there are no candidates, in which case you just return the `known` list as the answer. Use recursion to repeat, and make sure you check for any boundary cases.

```
> (sieve-helper (list 3 5 7 9) (list 2))
(list 2 3 5 7)
```

4. **(2 points)** Finally write a function `primes`, which just calls your helper function with the list of all numbers between 2 and n (as `candidates`) and the empty list (as `known`).

```
> (primes 39)
(list 2 3 5 7 11 13 17 19 23 29 31 37)
```

Note: Set your scheme language to "Beginning student with LIST abbreviations" (or higher) to save yourselves the bother of not having to read through a whole lot of `cons`'s.

Solution

The `interval` function is simple. As long as `low <= high`, keep adding `low` to the front of the list. This is used to generate the initial list of candidate primes (all numbers between 2 and `n`).

```
;; interval: number number -> list-of-number
;;    return a list of numbers between low and high inclusive
(define (interval low high)
  (cond
    [(> low high) empty]
    [else (cons low (interval (+ low 1) high))]
  )
)
```

Each round of processing is done by this function `sieve`. It just removes all multiples of its first argument from the given list of numbers. Just check if the first element is a multiple of `k` or not and add it to the output list if it is not. Exercise: (don't turn it in) implement `sieve` using `filter` and `local` or `lambda`.

```
;; sieve: number list-of-numbers -> list-of-numbers
;;   remove all multiples of numbers from list
(define (sieve k lon)
  (cond
    [(empty? lon) lon]
    [(= (remainder (first lon) k) 0) (sieve k (rest lon))]
    [else (cons (first lon) (sieve k (rest lon)))]
  )
)
```

The `sieve-helper` is the key to whole algorithm, it keeps track of all the numbers. Crossed out numbers are just removed from the list. Marked number are moved over to the `known` list, and unprocessed numbers (i.e. unmarked numbers not crossed out) are stored in candidates. Also, we assume that the first unprocessed number is always a prime. Exercise: (don't turn in) why is this true?. Hence, we just need to move the first number of `candidates` over to `known` and remove all multiples of `(first candidates)` from `candidates` (using `sieve`). When we no longer have any numbers to process, we just return the `known` list.

```
;; sieve-helper: list-of-numbers list-of-numbers -> list-of-numbers
;;   return list of primes in candidates or known.
;;   known has numbers known to be primes.
;;   candidates has a list of potential primes,
;;        and the first one is a prime.
(define (sieve-helper candidates known)
  (cond
    [(empty? candidates) known]
    [else (sieve-helper
            (sieve (first candidates) (rest candidates))
            (cons (first candidates) known)
```

```
          )
    ]
  )
)
```

Finally, putting it all together to get the list of primes till `n`, we start with all numbers from `2` to `n` and an empty list for `known`. Even though `sieve-helper` finds the primes in increasing order, it keeps adding it to the beginning of the list. Hence to get the list of primes in increasing order, we need to reverse the output of `sieve-helper`.

```
;; primes: number -> list-of-numbers
;;  list all primes less or equal to given number
(define (primes n)
  (cond
    [(<= n 1) empty]
    [else (reverse (sieve-helper (interval 2 n) empty))]
  )
)
```