**Name:**

CSCI 305 Concepts of Programming Languages

**Participation Test 5**

**Instructions:** Write your name above. Relax and attempt the problems above. This is NOT a quiz and participation credit will be given for any sincere attempt. (Later, solutions will be posted on the course webpage.) Turn in the sheet at the end of the class to receive your participation credit.

**Exercise A:** Consider the construction of lists in Scheme and fill out the table below. The first column shows a Scheme list \( X \). In the second column, show how you would build \( X \) using only `cons` and the empty list `()` and integers. In the third column, show how you would return the value 2 from list \( X \) using only `car` and `cdr`. The first line has been completed for you as an example.

<table>
<thead>
<tr>
<th>( X )</th>
<th>How to construct ( X )</th>
<th>How to return 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>( (1 \ 2) )</td>
<td><code>(cons 1 (cons 2 '()))</code></td>
<td><code>(car (cdr X))</code></td>
</tr>
<tr>
<td>( (0 \ 1 \ 2) )</td>
<td><code>(cons 0 (cons 1 (cons 2 '())))</code></td>
<td><code>(car (cdr (cdr X)))</code></td>
</tr>
<tr>
<td>( ((1) \ (2)) )</td>
<td><code>(cons (cons 1 '())) (cons (cons 2 '())())</code></td>
<td><code>(car (car (cdr X)))</code></td>
</tr>
<tr>
<td>( ((1 \ 2)) )</td>
<td><code>(cons (cons 1 (cons 2 '()))) (car X)</code></td>
<td><code>(car (car (cdr X)))</code></td>
</tr>
<tr>
<td>( (1 \ . \ 2) )</td>
<td><code>(cons (1 2))</code></td>
<td><code>(car X)</code></td>
</tr>
<tr>
<td>( (((2) \ 1) ) )</td>
<td><code>(cons (cons (cons 2 '()) (car X)) (cons (car X)))</code></td>
<td><code>(car (car (car (cdr X)))</code></td>
</tr>
<tr>
<td>( (1 \ (((2)))) )</td>
<td><code>(cons 1 (cons (cons (cons 2 '()) (car X)) (cons (car X))))</code></td>
<td><code>(car (car (car (cdr X)))</code></td>
</tr>
</tbody>
</table>
Exercise B: Define a Scheme function, odds, that takes a list and returns every other element, starting with the first. Use any of the primitive functions if, null?, cond, car, cdr and cons. Then define a Scheme function evens that takes a list and return the elements at even positions, starting with the second.

```
(define (odds list)
  (if (null? list)
      '()
      (cons (car list)
            (if (null? (cdr list))
                '()
                (odds (cdr (cdr list))))))
)

(define (evens list)
  (if (null? list)
      '()
      (odds (cdr list))
)
```

When running your code, the output should be like:
(odds '()) → ()
(odds '(a b)) → (a)
(odds '(a b c d e)) → (a c e)
(evens '(a)) → ()
(evens '(a b)) → (b)
(evens '(a b c d e)) → (b d)