1. (10 points)

(a) (2 points) Define the terms syntax and semantics in regard to a language.
Syntax is the set of legal words in a language and the legal ordering of those words.
Semantics refers to the meaning of words and expressions used in a language.

(b) (2 points) Suppose a and b are defined as #f, and c and d are defined as #t. What would be the result of evaluating the following expressions (true or false)?

i. (and (not a) (or b c d))
   true

ii. (or a (not (not b)) (not (and c d)))
   false

(c) (6 points) Use the method shown in class to convert the following infix expressions to prefix notation.
Just convert the expressions to prefix notation, do not simplify or change the order of operands:

i. \((x - 32) \times (\frac{5}{9})\)
   = \((x - 32) \times (5 \div 9)\) (fully parenthesized)
   = \((-x 32)(5 9)\) (prefix notation)

ii. \((x + 3 \times y) \times (x - y)\)
   = \(((x + (3 \times y)) \times (x - y))\) (fully parenthesized)
   = \((+x(*3y)(-xy))\) (prefix notation)

iii. \(a + ((b^c - d) \div e) \times f\)
   Hint: you may use expt
   = \((+ a ((expt b c) - d) \div e) \times f)\) (fully parenthesized)
   = \((+ a (* (/ (- (expt b c) d) e) f))\) (prefix notation)

2. (10 points) Consider the define statements and representation of the Global Environment given below:

(define A 3)
(define B 2)
(define C (* A B))
(define X (+ C 3))
(define fun (lambda (x y) (* x (+ y 4))))

(a) (5 points) Show, in the table below, the additions to the global environment created by the 5 definitions above when they are executed in DrRacket (assume entries are added from the top of the table downward).

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>6</td>
</tr>
<tr>
<td>X</td>
<td>9</td>
</tr>
<tr>
<td>fun</td>
<td>(λ (x y)(* x (+ y 4)))</td>
</tr>
</tbody>
</table>

(b) (5 points) Using the Global Environment from part (a), show the result of evaluating each of the following expressions, keeping in mind that the evaluation may cause an error message (the > is the Interactions Window prompt):
> (expt C 2)
36
> (* B 1000)
2000
> (string-append A B)
# error: operands are not strings
> (string? X)
#f
> (fun A X)
(lamba (3 9) (* 3 (+ y 4))) = 39

3. (15 points) Define a function, called name-card, that consumes an integer in the range 1...13, and a quoted symbol 'H, 'D, 'C, or 'S. The function should use the integer input to calculate the output depicted in the table below:

<table>
<thead>
<tr>
<th>First Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&quot;Ace&quot;</td>
</tr>
<tr>
<td>2...10</td>
<td>convert given number to a string</td>
</tr>
<tr>
<td>11</td>
<td>&quot;Jack&quot;</td>
</tr>
<tr>
<td>12</td>
<td>&quot;Queen&quot;</td>
</tr>
<tr>
<td>13</td>
<td>&quot;King&quot;</td>
</tr>
</tbody>
</table>

Your function should append one of the string outputs shown in the table above to the string generated for the quoted symbol parameter: " of Hearts" (for input 'H), " of Diamonds" (for input 'D), " of Clubs" (for input 'C), and " of Spades" (for input 'S). Read the 9 calls to this function in the interactions window output below to better understand how this function should work (the > is the interactions window prompt).

> (name-card 11 'D) > (name-card 10 'H) > (name-card 4 'C)
"Jack of Diamonds" "10 of Hearts" "4 of Clubs"
> (name-card 1 'H) > (name-card 13 'D) > (name-card 1 'S)
"Ace of Hearts" "King of Diamonds" "Ace of Spades"
> (name-card 3 'C) > (name-card 12 'S) > (name-card 6 'H)
"3 of Clubs" "Queen of Spades" "6 of Hearts"

On the lines indicated below, write a contract, header, (brief) purpose and at least 3 check-expect statements to demonstrate your understanding of how the function should work. Write the function definition for name-card on the facing page (page 5). Choose descriptive parameter names for the input parameters (e.g., single letter parameter names are not descriptive of what the parameters represent). There is no need to define constants for the literal values you use in the function and you do not have to account for erroneous input.

**Built-in functions that may be useful in your function:** number->string, symbol=?, and string-append.

; Contract: (name-card num sym) -> string
; Inputs: num=number between 1 and 13 (inclusive), sym:symbol
; Purpose: Return a string representation of given card number and symbol.
; Pre-function tests:
(check-expect (name-card 10 'H) "10 of Hearts")
(check-expect (name-card 1 'C) "Ace of Clubs")
(check-expect (name-card 11 'D) "Jack of Diamonds")
; Function definition:
(define name-card
  (lambda (num sym)
    (string-append
      (cond
        [(= num 1) "Ace"]
        [(<= num 2 10) (number->string num)]
        [(= num 11) "Jack"]
        [(= num 12) "Queen"]
        [(= num 13) "King"]
      )
      (cond
        [(symbol=? sym 'H) " of Hearts"]
        [(symbol=? sym 'D) " of Diamonds"]
        [(symbol=? sym 'S) " of Spades"]
        [(symbol=? sym 'C) " of Clubs"]
      )
    ))
)

4. (10 points) Given the following character sequences (words) (a through e below), indicate whether the word is a special form, a built-in function, or neither. If you answer special form, indicate how the form is used and give the return type (or void if a side-effect is the only thing returned).

For example:

    if  SPECIAL FORM used to make decisions. if has 3 arguments. The first argument is a boolean expression, bool. If bool is true, the second argument is evaluated, otherwise the third argument is evaluated. The value of either the second or the third argument is returned.

If you answer built-in function, give an example of calling the function on valid arguments and indicate the output of the function call. For example:

    +  A built-in function that takes numeric arguments and returns a number, the sum of its arguments. Example:  (+ 5 7 4 8 2) ==> 26

If you think the given word is an undefined name, write “neither”. For example:

    my-func  neither

(a) boolean?
    built-in function that takes any Racket type as input and returns a boolean. Ex: (boolean? 5) = #f
(b) λ
    special form used to create functions. Ex: (lambda (x) (add1 x))
(c) >=
    built in function that consumes numbers and returns a boolean. Ex: (>= 8 5) = #t
(d) or
    special form that consumes any number of booleans and that returns #f only if all arguments are false. Ex: (or false false false true) = true
5. (5 points) This question involves a recursive function.

(a) (1 point) Suppose you were asked to write a regular recursive function that produces the sum of the cube of each number between 0 and n (no negative numbers allowed). This sum could be expressed as

\[0^3 + 1^3 + 2^3 + 3^3 + \ldots + n^3]\]

Would the following function compute this sum? Answer “yes” or “no”. No

\[
\text{(define sum-cube
  (lambda (n)
    (cond
      [(= n 1) 0]
      [else
        (+ (* n n) (sum-cube (- n 2)))])
  ))}
\]

(b) (4 points) Rewrite the sum-cube function below, correcting any errors in the function (if you found any errors), and adding appropriate comments on each clause.

\[
\text{(define sum-cube}
  \text{ (lambda (n))}
  \text{ (cond}
    \text{ [(= n 0) 0]}
    \text{ [else}
      \text{ (+ (* n n n) (sum-cube (- n 1)))])})\)
\]

6. (5 points) Consider the count-down function, given in the space below on the left. Assume the line numbers and the text to the right are not part of the function.

The count-down function is supposed to produce the output shown on the right when run in the interactions window. Unfortunately, in its current state, it would fail to do so.

1. (define count-down
2.  (lambda (n) 5
3.    (cond 4
4.      [(n = 0) 3
5.        (begin 2
6.          (printf "~a~%" n) 1
7.            (printf "BLAST-OFF")))) 0
8.      [else BLAST-OFF
9.        (begin >
10.          (printf "~a~%" n)
11.            (count-down (n - 1)))]))

(a) (2 points) Indicate which lines in this function would need to be changed in order for the function to produce the desired output when run in the interactions window. After writing each line number that needs to be changed, re-write the line in its correct form.
Line 4: [(n = 0) should be [(= n 0)]
Line 11: count-down (n - 1) should be count-down (- n 1)

(b) (3 points) Write a contract, input, and purpose for the count-down function.

; Contract: (count-down n) -> void; only side-effect printing
; Input: n=non-negative integer (>= 0)
; Purpose: Print a sequence of numbers from n down to 0 and then print BLAST-OFF.