Problem 1 (4 points, 5 minutes)

Part a

Fill in the blanks below to complete the next-higher procedure. Given a grade A, B, C, D, or F, next-higher returns the next higher grade; the next higher grade for A is A.

(define (next-higher grade)
  (cadr
    (assoc
      grade
      ____________________________________________)))

Part b

A grading policy is a procedure that takes as argument a list of scores and returns a letter grade. Define a procedure named generous that, given a grading policy as an argument, returns a grading policy that awards a grade one higher than the argument policy would give. Use the next-higher procedure from part a.

Problem 2 (4 points, 7 minutes)

Consider the following procedure.

(define (exam a)
  (let ((b 9))
    (lambda (c)
      (let ((d 11))
        (set! a (+ a 1))
        (set! b (+ b 2))
        (set! c (+ c 3))
        (set! d (+ d 4))
        (list a b c d) ) ) )

Fill in the blank with the output that stk would produce.

STk> (define f (exam 7))
f
Problem 3 (6 points, 7 minutes)

Suppose the second and third arguments to the call to lookup-variable-value in eval-1 are accidentally exchanged as follows, with no other changes to the program:

```
((eq? kind 'symbol)
 (lookup-variable-value exp outer-env inner-env))
```

Give a sequence of expressions whose effect in the Scheme-1 interpreter would differ in the modified version from its effect in the original version. Also indicated how the Scheme-1 interpreter would handle the expressions you provide, and explain how and why their behavior in the modified code would differ.

Problem 4 (7 points, 10 minutes)

Consider a procedure named find that’s given two arguments,

- an x (which may be of any type), and
- a table whose elements have the form (list pred_k val_k), where pred_k is a one-argument predicate and val_k is any value.

Find searches the table for the first pred_k for which (pred_k x) is true, and then returns val_k. If (pred_k x) is false for all pred_k in the table, find returns #f. Three examples appear below.

```
STk> (define tb1 (list (list integer? ’a) (list symbol? ’b)))
tbl
STk> (find 17 tbl)
a
STk> (find ’mike tbl)
b
STk> (find ’(a b) tbl)
#f
```

Implement find using a single call to accumulate or reduce. Use a lambda expression rather than a named procedure for the argument to accumulated or reduce, and use descriptive names for its parameters.
(define (find x table)
  (____________________ ; accumulate or reduce goes here
  ; arguments go here)

---

**Problem 5 (7 points, 12 minutes)**

An integer range represents a sequence of consecutive integers. It is represented by a two-element list whose first element is the first integer in the sequence and whose second element (a non-negative integer) is the number of integers that follow in the sequence. Some examples:

<table>
<thead>
<tr>
<th>integer range</th>
<th>sequence represented</th>
</tr>
</thead>
<tbody>
<tr>
<td>(9 4)</td>
<td>9, 10, 11, 12</td>
</tr>
<tr>
<td>(-3 5)</td>
<td>-3, -2, -1, 0, 1</td>
</tr>
<tr>
<td>(1 1)</td>
<td>1</td>
</tr>
<tr>
<td>(4 0)</td>
<td>empty sequence</td>
</tr>
</tbody>
</table>

Define a procedure named expanded that, given a possibly infinite stream of integer ranges as argument, returns the stream of integers that results from expanding all the integer ranges into the sequences they represent. This stream may contain duplicate values; for example, if int-range-stream is defined to be the stream containing the ranges (1 1), (-3 5), (4 0), and (9 4)), then (expanded int-range-stream) should return the stream containing the integers 1, -3, -2, -1, 0, 1, 9, 10, 11, 12. You may use auxiliary procedures.
Problem 6 (10 points, 18 minutes)

One often sees novice Scheme programmers code a test as (if \textit{expr} #t #f) where merely saying expr would suffice. The fixed procedure below is intended to replace all occurrences of (if expr #t #f) by \textit{expr}; fill in the blanks to complete the procedure. Assume that the argument expression is anything recognized by the Scheme-0 interpreter.

Your solution shouldn't do any evaluating. For example, you shouldn't simplify the expression (if (> a 0) (quote #t) #f). Your solution must, however, handle nested expressions. For instance, it should return #t when given the argument

\begin{verbatim}
'(if (if #t #t #f) (if #t #t #f) (if #f #t #f))
\end{verbatim}

(define (fixed expr)
  (cond
    ; base cases
    
    ((eq? (car expr) 'if)
      
        
    )

((eq? (car expr) 'if)
  
)
Problem 7 (1 point extra credit)

According to the Belgians, playing what musical instrument requires "a strong back, a weak mind, and freedom from gout"?