Mid-Term Quiz
Set A

Time allowed: 50 minutes

Matriculation No:

Instructions (please read carefully):
1. Write down your matriculation number on the question paper. DO NOT WRITE YOUR NAME ON THE QUESTION SET!
2. This is a closed-book quiz. You are not allowed to use any books, notes, sheets, memos, etc., except for the question sheet provided. The use of laptops and calculators is prohibited.
3. Each question contains an appendix with code from the text book that you may need to refer to in your answer.
4. This paper comprises 3 questions and 12 pages. The time allowed for solving this quiz is 50 minutes.
5. The maximum score of this quiz is 20 marks. The weight of each question is given in square brackets beside the question number.
6. All questions must be answered correctly for the maximum score to be attained.
7. All questions must be answered in the space provided in the answer sheet; no extra sheets will be accepted as answers.
8. The back-sides of the sheets and the pages marked “scratch paper” in the question set may be used as scratch paper; you will not be allowed to use any other kind of scratch paper.

GOOD LUCK!

For examiner’s use:

\[
\begin{array}{|c|}
\hline
\text{Q1} \\
\hline
\text{Q2} \\
\hline
\text{Q3} \\
\hline
\text{Total} \\
\hline
\end{array}
\]
Question 1 [5 marks]

Assume the following expression has just been evaluated:

```
(define (fact-iter n m k)
  (if (> n m)
      1
      (fact-iter n (- m 1) (* k m))))
```

a) Show the value of the `fact-iter` symbol in the symbol table [1 mark]

Answer:

b) Using the Revised Substitution Model, show how the following expression is evaluated: [4 marks]

```
(fact-iter 1 1 1)
```

Answer:
Question 2 [11 marks]

Consider the following procedure:

\[
\text{(define (count-even list-of-numbers)} \\
\quad \text{(if (null? list-of-numbers)} \\
\quad \quad \text{0)} \\
\quad \quad (+ \text{(if (even? (car list-of-numbers))} \\
\quad \quad \quad \text{1)} \\
\quad \quad \text{0)} \\
\quad \text{(count-even (cdr list-of-numbers))}))
\]

a) What kind of procedure is this? Iterative or recursive? [1 mark]

\[
\text{Answer:}
\]

b) Rewrite this procedure using the well-known procedures accumulate and map (please refer to the appendix on page 6 for the code of these 2 procedures). Use the following model (you will have to fill in the \(<...>\) brackets): [3 marks]

\[
\text{(define (count-even list-of-numbers)} \\
\quad \text{(accumulate \(<...> \<...> \text{(map \(<...> \text{list-of-numbers)})})}
\]

\[
\text{Answer:}
\]
c) Rewrite this procedure using filter and length (refer to the appendix on page 6 for the code of these procedures). Use the following template (you need to fill in the <...> brackets): [2 marks]

(define (count-even list-of-numbers)
  (length (filter <...> list-of-numbers)))

Answer:

(d) If you answered “recursive” to subquestion 2a, then write an iterative version of this procedure. If your answer was “iterative”, write a recursive version for this procedure [3 marks].

Answer:
e) A sublist of a list $L$ (or a list segment of $L$) is another (shorter) list whose elements appear in the same sequence in $L$. For example, if $L$ is $(1, 3, 4, 5, 7, 8, 9)$, then $(1, 3, 4), (4, 5, 7, 8), (3, 4, 5, 7, 8, 9)$ are sublists of $L$. Write a procedure `sublists-of-length` that takes a number and a list as arguments, and returns the list of all sublists whose length is equal to the number. [2 marks]

Example calls:

```
(sublists-of-length 2 (list 1 2 3 4 5))
;Value: ((1 2) (2 3) (3 4) (4 5))

(sublists-of-length 3 (list 1 2 3 4 5))
;Value: ((1 2 3) (2 3 4) (3 4 5))

(sublists-of-length 5 (list 1 2 3 4 5))
;Value: ((1 2 3 4 5))

(sublists-of-length 6 (list 1 2 3 4 5))
;Value: ()
```
(define (accumulate op initial sequence)
  (if (null? sequence)
      initial
      (op (car sequence)
           (accumulate op initial (cdr sequence))))

(define (map proc items)
  (if (null? items)
      nil
      (cons (proc (car items))
            (map proc (cdr items)))))

(define (length items)
  (if (null? items)
      0
      (+ 1 (length (cdr items)))))

(define (filter predicate sequence)
  (cond ((null? sequence) nil)
        ((predicate (car sequence))
         (cons (car sequence)
               (filter predicate (cdr sequence))))
        (else (filter predicate (cdr sequence))))

Appendix for Question 2
Question 3 [4 marks]

Using the procedures flip-vert, rotate90, beside and below defined for the picture language, (refer to the appendix on page 9 for the code of these procedures) define a procedure that performs the following transformation of a painter:

Answer:
Appendix for Question 3

(define (transform-painter painter origin corner1 corner2)
  (lambda (frame)
    (let ((m (frame-coord-map frame)))
      (let ((new-origin (m origin)))
        (painter
         (make-frame new-origin
                     (sub-vect (m corner1) new-origin)
                     (sub-vect (m corner2) new-origin)))))))

(define (flip-vert painter)
  (transform-painter painter
                       (make-vect 0.0 1.0) ; new origin
                       (make-vect 1.0 1.0) ; new end of edge1
                       (make-vect 0.0 0.0))) ; new end of edge2

(define (rotate90 painter)
  (transform-painter painter
                       (make-vect 1.0 0.0)
                       (make-vect 1.0 1.0)
                       (make-vect 0.0 0.0)))

(define (beside painter1 painter2)
  (let ((split-point (make-vect 0.5 0.0)))
    (let ((paint-left
            (transform-painter painter1
                         (make-vect 0.0 0.0) ; new origin
                         split-point
                         (make-vect 0.0 1.0)))
           (paint-right
            (transform-painter painter2
                         split-point
                         (make-vect 1.0 0.0)
                         (make-vect 0.0 0.0))))
     (lambda (frame)
      (paint-left frame)
      (paint-right frame)))))))

(define (below painter1 painter2)
  (let ((split-point (make-vect 0.0 0.5)))
    (let ((paint-top
            (transform-painter painter1
                         split-point
                         (make-vect 1.0 0.5)
                         (make-vect 0.0 1.0))
           (paint-bottom
            (transform-painter painter2
                         (make-vect 0.0 0.5)
                         (make-vect 1.0 1.0)
                         (make-vect 0.0 1.0))))
     (lambda (frame)
      (paint-top frame)
      (paint-bottom frame)))))))

9
(lambda (frame)
  (paint-top frame)
  (paint-bottom frame))))