Duration: **50 minutes**
Aids Allowed: **NONE** (in particular, no calculator)

Student Number: |_|_|_|_|_|_|_|_|_|

Last (Family) Name(s): SOLUTIONS

First (Given) Name(s): SAMPLE

---

*Do **not** turn this page until you have received the signal to start.*
(In the meantime, please fill out the identification section above, and read the instructions below *carefully*.)

This test consists of 5 questions on 5 pages (including this one), printed on one side of the paper. *When you receive the signal to start, please make sure that your copy of the test is complete and write your student number at the bottom of every page, where indicated.*

Answer the selected questions directly on the test paper, in the space provided, and use the reverse side of the pages for rough work. If you need more space for one of your solutions, use the reverse side of a page and *indicate clearly the part of your work that should be marked.*

In the programming questions, you do not have to write pre-conditions/post-conditions. Also, you do not have to comment your code but use meaningful variable names. You may use helper procedures (unless otherwise indicated). Use recursion where possible.

Marking Guide

# 1:	______/10
# 2:	______/ 6
# 3:	______/ 6
# 4:	______/ 5
# 5:	______/ 8
TOTAL:	______/35

Good Luck!

Question 1. [10 MARKS]

Short answer section: Give a short answer to each of the following questions. Marks will be deducted for overly long answers.

Part (a) [2 MARKS]

What is the difference between an effect and a side-effect of an expression?

SAMPLE SOLUTION

The effect is what the expression evaluates to, while a side-effect causes the state of the environment to change: changing the value of a memory cell, input or output, etc.

Part (b) [2 MARKS]

Give an example of an expression in Java or C that has a side-effect. What is the side-effect?

SAMPLE SOLUTION

Many different solutions. `i++;` is a common answer, changing the value of a variable.

Part (c) [2 MARKS]

Give an example of a Scheme expression that has a side-effect.

SAMPLE SOLUTION

Analagously, `(set!  i (+ 1 i))`. `(display "Hi!")` is also perfectly acceptable.

Part (d) [2 MARKS]

Give one advantage of compilation over interpretation.

SAMPLE SOLUTION

Speed of execution, compiler can optimize more aggressively, etc.

Part (e) [2 MARKS]

Give one advantage of interpretation over compilation.

SAMPLE SOLUTION

Better debugging/error messages, rapid prototyping, easy to “tweak” a program, etc.

Question 2. [6 MARKS]

Suppose we write the following statements in Scheme:

```
(define x 10)
(define y 20)
```

What do the following Scheme expressions evaluate to? Assume (for the purposes of side-effects) that they are executed in the given order.

		Value:
(a)	`(+ x 1)`	11
(b)	`(let ((x 2)) (+ x y))`	22
(c)	`(let ((x 1) (y x)) (+ x y))`	11
(d)	`(begin (set! x 5) (- y x))`	15
(e)	`((lambda (x a) (set! x a) x) x 4)`	4
(f)	`(+ x 1)`	6

Question 3. [6 MARKS]

`list-and` takes a single argument, a list, and returns the *logical and* of the elements of the list. That is, it returns `#f` if any of the items in the list are `#f`, and `#t` otherwise. For example,

```
(define l `(#t 1 #f ,(even? 2)))
(list-and l) ⟹ #f
(list-and `(1 ,(even? 2))) ⟹ #t
```

Note that the list has already been composed (and its elements evaluated), so we don't use short-circuit evaluation here.

Part (a) [1 MARK]

Can `list-and` be written as a procedure in Scheme? (yes or no) Yes

Part (b) [5 MARKS]

Write a Scheme procedure that performs `list-and` (without using `and`), or explain why it can't be done.

SAMPLE SOLUTION

```
(define list-and
  (lambda (l)
    (if (null? l) #t
        (if (car l) (list-and (cdr l))
            #f))))
```

Question 4. [5 MARKS]

The Scheme `not` procedure takes a single argument and returns the *logical negation* of the argument.

Part (a) [1 MARK]

Can `not` be written as a macro (syntactic extension) in Scheme (without using the built-in `not`)? (yes or no) Yes

Part (b) [4 MARKS]

Write a Scheme macro that performs `not` (without using the built-in `not` and without defining new procedures), or explain why it can't be done.

SAMPLE SOLUTION

```
(define-syntax my-not
  (syntax-rules ()
    ((_ a) (if a #f #t))))
```

Question 5. [8 marks]

A `repeat` ... `until` loop is common in many programming languages. We want to write code like the following in Scheme:

```
(let ((i 1))
  (repeat (display i)
          (newline)
          (set! i (+ i 1))
          (until (> i 5))))
```

Part (a) [4 marks]

Write the Scheme code that the following statement should expand to:

```
(repeat EXPR1 EXPR2 (until TEST))
```

Use only standard Scheme procedures and forms (such as `if`, `lambda`, the `let` forms, etc.).
Hint: Write a procedure to execute one iteration of the loop, and call it until the test is true.

Sample Solution

There are two general methods: letrec and named let.

```
(letrec ((p (lambda () EXPR1 EXPR2 (if (not TEST) (p))))) (p))

or

(let loop () EXPR1 EXPR2 (if (not TEST) (loop)))
```

Note that let does not work, since the procedure must recursively call itself.

Part (b) [4 marks]

Write a macro in Scheme that implements a `repeat` ... `until` loop as illustrated above. It should automate the expansion you gave in part (a). `until` should be treated as a keyword, and must appear last.

Sample Solution

```
(define-syntax repeat
  (syntax-rules (until)
    ((_ e ... (until test))
     (let loop () e ... (if (not test) (loop))))))
```