Examination for
Bachelor of Computer Science, Bachelor of Mathematical and Computer Science, Bachelor of Business Information Technology, Bachelor of Engineering, Graduate Diploma in Computer Science, Masters of Information Technology, Masters of Computer Science, Masters of Software Engineering.

Semester 2, November 2010

9811	**Advanced Programming Paradigms COMPSCI 3009, 7031**

Official Reading Time: 10 mins
Writing Time: 120 mins
Total Duration: 130 mins

Questions	Time	Marks
Answer all 8 questions	120 mins	120 marks
		120 Total

Instructions

- Begin each answer on a new page
- Examination material must not be removed from the examination room
- No Calculators Allowed
- Dictionaries for translation only

Materials

- 1 Blue book

DO NOT COMMENCE WRITING UNTIL INSTRUCTED TO DO SO

Question 1

(a) The exponent function can be defined as:

```
exp(x, n) = x * exp(x, n-1)     , if n > 0
          = 1                   , if n = 0
```

i. Write a Scheme function which implements exp as defined above. Your function must generate a *Recursive process*.

[4 marks]

ii. Write another version of exp which generates an *Iterative* process.

[5 marks]

(b) Write a Scheme procedure, called `enum-downfrom`, that takes a positive integer `n` and returns a list of integers from `n` down to zero inclusive. So, for example:

```
(enum-downfrom 4)
```

returns the list `(4 3 2 1 0)`.

[3 marks]

(c) Write a Scheme procedure, called `drop`, which takes a positive integer parameter `n` and a list parameter `ls` and returns a list consisting of all but the first `n` elements of `ls`. If `ls` is less than `n` elements long then `drop` should just return the empty list. So, for example:


```
(drop 3 (list 3 4 5 6 7 8))
```


will return the list `(6 7 8)` and:

```
(drop 10 (list 1 2 3))
```

will return the empty list: `()`

[4 marks]

(d) Write a Scheme procedure called `sfunc` which takes in two procedural values: `f` and `g` and returns a procedure which takes a value `x` and returns: *f(x)* + *g(x)*. So, for example, given the following definitions:

```
(define (square x) (* x x))
(define (inc x) (+ x 1))
(define si (sfunc square inc))
```

The call: `(si 4)` = (+ 16 5) = 21

[4 marks]

(e) Consider the following definition of `sum`, a proccedure which computes the sum of all numbers in the range `a` to `b`.

```
(define (sum a b)
   (if (> a b)
       0
       (+ a
          (sum (+ 1 a) b))))
```

An analogous procedure can also be written for `product`:

```
(define (product a b)
   (if (> a b)
       1
       (* a
          (product (+ 1 a) b))))
```

Write a higher-order Scheme procedure, called `accumulate`, that can generalize these types of series by taking in an `operator` and `identity` value thus:

```
(accumulate operator identity a b)
```


So that we can re-define the above series as:

```
(define (sum a b)     (accumulate + 0 a b))
(define (product a b) (accumulate * 1 a b))
```


[5 marks]

[Total for Question 1: 25 marks]

Question 2

Appendix 1 shows a Scheme definition for a simple `Item` object which could be used in an inventory system. An item object consists of a *name* and *stock* which tracks how many of that particular item are in the inventory.

(a) Write down a Scheme expression to create an item, `B1` with a name of "Ale", and an inital stock of 10. Please note that you can represent the name using either a String or a quoted-literal. That is, as "Ale" or 'Ale

[2 marks]

(b) To increase the stock of item `B1` by five, the following expression is needed:

```
( (B1 'addItems) 5)
```

Many people find this notation confusing: write a scheme procedure, `increaseStock`, which accepts two arguments: an item name and an amount. It can then be used thus:

```
(increaseStock B1 5)
```

[3 marks]

(c) i. Draw an environment diagram which shows the state of the system after `B1` from part (a) has been defined.

[4 marks]

ii. Now extend your diagram from part (i) to depict the evaluation of `(define adder (B1 'addItems))`.

[3 marks]

iii. Finally extend your diagram to depict the evaluation of `(adder 5)` which will increase the stock of item `B1` by five.

[2 marks]

[Total for Question 2: 14 marks]

Question 3

(a) Examine each of the following four expressions, and state whether or not it is a syntactically valid lambda-expression:

- $(\lambda x.x)$
- λx
- $x\ y$
- $\lambda a.(\lambda a.a)$

[4 marks]

(b) Reduce, using normal order evaluation, the following lambda expression. Show your working, identifying the binding variable, argument and substitution made at each step, and stating the lambda reduction rule used.

$(((\lambda x.\lambda y.x)z)((\lambda r.r\ r)(\lambda s.s\ s)))$

[4 marks]

[Total for Question 3: 8 marks]

Question 4

(a) State, and briefly explain, the three reduction rules of the lambda calculus. Use examples in your answer.

[6 marks]

(b) Consider these Scheme definitions, each of a function of one argument that always produces a constant value:

```
(define (consta x) 'a)
(define (constb x) 'b)
```

Now consider the Scheme expression

```
( (compose consta constb) 'c))
```

Write this expression in λ-notation, and use the rules of the λ-calculus to illustrate how evaluation of the expression takes place. To do this, you will need to first define `compose` in the λ-calculus: the composition of two one-argument functions f and g is defined as the function that maps an argument x to $f(g(x))$.

[7 marks]

[Total for Question 4: 13 marks]

Question 3

(a) What is the most important characteristic of streams in Scheme, that distinguishes them from lists?

[2 marks]

(b) Explain, with a simple example, why it is not possible to implement streams in Scheme using the pre-defined operations *cons*, *car* and *cdr*.

[4 marks]

(c) Write a Scheme procedure, called `even-stream`, that takes an infinite stream of integers, and produces a stream containing only the even elements of the stream. In your answer, you must define any stream-processing procedures that you use. You may assume that the predicate `even?` has been defined.

[6 marks]

(d) The Fibonacci numbers may be defined in Scheme as:

```
(define fib (cons-stream 1 f2))
(define f2 (cons-stream 1 f3))
(define f3 (addL fib f2))
```


Draw a process network corresponding to this definition. On your diagram, label the arcs corresponding to `fib`, `f1`, and `f2`.

[4 marks]

[Total for Question 3: 16 marks]

Appendix 1 - Scheme definition of item object

```
(define (make-item name stock)

  (define (addItems n)
    (set! stock (+ stock n)))

  (define (removeItems n)
    (set! stock (- stock n)))

  (define (dispatch m)
    (cond  ( (eq? m 'name) name )
           ( (eq? m 'stock) stock)
           ( (eq? m 'addItems) addItems)
           ( (eq? m 'removeItems) removeItems)
           (else (error "Unknown request" m))))

   dispatch)
```


End of exam