THE UNIVERSITY of York

Degree Examinations 2011-12

DEPARTMENT OF COMPUTER SCIENCE

Principles of Programming Languages

Time allowed: **Two hours**

Candidates should answer question 1 in Section A, one question from Section B and one question from Section C.

Calculators may be used in this examination.

Do not use red ink.

Section A: You must answer this question

1 (20 marks)

(i) [2 marks] Some programming languages are *typeless*. Give brief arguments for and against this language-design decision.

(ii) [2 marks] Distinguish between *static typing*, and *strong typing*.

(iii) [2 marks] Which of the following argument-passing methods *do not* allow a procedure's body to modify the value of the argument in the caller?

- call-by-value
- call-by-reference
- call-by-name

(iv) [2 marks] Briefly describe the grammatical ambiguity in conventional 'if' statements.

(v) [2 marks] Distinguish between a *conditional expression* and a *conditional statement*, and give an example of a language that has *both* types of conditional construct.

(vi) [2 marks] What is the principal difference between a *function* in Haskell, and a *method* in Java?

(vii) [2 marks] What does it mean to say that the concurrency operator is transitive?

(viii) [2 marks] What does it mean for a system of tasks to be deadlocked?

(ix) [2 marks] What is a binary semaphore, and what is it used for?

(x) [2 marks] Why in a nested task structure must a parent task never terminate before a child task?

Section B: Answer one question from this section

2 (40 marks)

(i) [10 marks] Most languages use *static scoping* in preference to *dynamic scoping*. Explain these terms, and briefly say why static is the more popular design choice.

(ii) [12 marks] Consider the following skeleton of an Ada-like program:

```
procedure Main is
  X, Y, Z : Integer;
  procedure Sub1 is
    A, Y, X : Integer;
    begin
          -- Sub1 body
    end;
  procedure Sub2 is
    A, B, Z : Integer;
    begin
    -- Sub2 body
    end;
  procedure Sub3
    A, X, W : Integer;
    begin
      --- Sub3 body
    end;
  begin
    --- Main body
  end;
```

For *this* program, assume that the language uses *dynamic scoping* (unlike actual Ada programs). For each of the following calling sequences, state which variables are visible during the execution of the last procedure in the sequence. You must state the variable name, together with the name of the subprogram in which is is declared.

(a) [3 marks] `Main` calls `Sub1`; `Sub1` calls `Sub2`; `Sub2` calls `Sub3`.

(b) [3 marks] `Main` calls `Sub1`; `Sub1` calls `Sub3`.

(c) [3 marks] `Main` calls `Sub1`; `Sub1` calls `Sub3`; `Sub3` calls `Sub2`.

(d) [3 marks] `Main` calls `Sub3`; `Sub3` calls `Sub2`; `Sub2` calls `Sub1`.

(iii) [18 marks]

(a) [4 marks] Briefly explain what is meant by *type inference*.

(b) [14 marks] The following are two function definitions in Haskell. In each case deduce the most general type. You must describe the sequence of steps that you use to reach your result.

a) (6 marks)

```
map f [] = []
map f (x:xs) = f x : map f xs
```

b) (8 marks)

```
foldr f z []     =  z
foldr f z (x:xs) =  f x (foldr f z xs)
```

3 (40 marks)

(i) [8 marks] A language is said to have the property of *referential transparency* if replacing any expression by its value does not effect the meaning of the program. Many pure functional languages, such as Haskell, have this property.

(a) [4 marks] Why would referential transparency be useful for a language designed for writing concurrent programs?

(b) [4 marks] Explain, with an example, a feature of `scheme` that prevents it from being referentially transparent.

(ii) [12 marks] The specification for `scheme` states that all implementations must be 'properly tail-recursive'.

(a) [2 marks] Briefly explain what is meant by a *tail-recursive* procedure.

(b) [6 marks] Give actual programming examples (in a language of your own choice) of a procedure that is *not* tail-recursive, together with a version of a procedure with the same function that *is* tail-recursive. You *must* show the code for the whole procedure, but small errors in syntax will not lose marks.

(c) [4 marks] In what way is the tail-recursive version of a `scheme` procedure more efficient than a non tail-recursive procedure?

(iii) [20 marks] The following two `scheme` procedures, `(expt b n)` and `(fast-expt b n)` calculate the value of b^n, for integer n.

```
(define (expt b n)
  (if (= n 0)
      1
      (* b (expt b (- n 1)))))

(define (fast-expt b n)
  (cond ((= n 0) 1)
        ((even? n) (square (fast-expt b (/ n 2))))
        (else (* b (fast-expt b (- n 1))))))
```

where `(even? n)` returns `#t` if `n` is an even number, and `(square x)` returns x^2.

(a) [10 marks] Explain how each of these procedures calculate the value of b^n

(b) [10 marks] A user tested the speed of these by evaluating the following

expressions:

1. `(expt 2 12345 )`
2. `(fast-expt 2 12345)`
3. `(expt 2 123450)`
4. `(fast-expt 2 123450)`

and found that evaluation 1 was $23\times$ slower than evaluation 2, and evaluation 3 was $487\times$ slower than evaluation 4

Explain why `fast-expt` is the faster, and why its speed improves relative to `expt` for larger values of n.

Section C: Answer one question from this section

4 (40 marks)

(i) [6 marks] What are monitors used for in concurrent programming languages?

(ii) [10 marks] Explain how Ada and Java support monitors. What are the main differences between the approaches of the two languages.

(iii) [12 marks] The following Ada code implements a broadcast facility. Explain in detail the behaviour of this code as it is used by client tasks.

```
protected type Broadcast is
  entry Await;
  procedure Release(N : positive);
private
  Number : integer := 0;
end Broadcast;

protected body Broadcast is

  entry Await when Number > 10 is
  begin
    if Await'count = 0 then
      Number := 0;
    end if;
  end Await;

  procedure Release(N : positive) is
  begin
    Number := Number + N;
  end Release;

end Broadcast;
```

(iv) [12 marks] Write a Java class that corresponds to this facility.

5 (40 marks)

(i) [10 marks] A message-passing facility within a concurrent programming language can be defined in terms of the following characteristics: level of synchronisation, naming and type of message. Explain, briefly, each of these three terms.

(ii) [10 marks] Using the above terms describe the Erlang model of message passing. (Your solution should include brief descriptions of the language features.)

(iii) [10 marks] Define the properties of the Ada select statement as it is used in conjunction with accept statements.

(iv) [10 marks] A server task controls membership of a club. Other tasks can Join, Leave, Suspend or Un-suspend their membership of the club. At most there can be only 100 members (i.e. members that are active or have suspended their membership). Give the outline of an Ada task that implements this membership scheme using message passing. The task should terminate if there are no members and no tasks exist who could Join.

COM000051

THE UNIVERSITY of York

DEPARTMENT OF COMPUTER SCIENCE

Degree Examinations 2011-12

Principles of Programming Languages

Marking Notes

Marking Notes

Answers for Section A: You must answer this question

Question 1 (20 marks)

Part (i) [2 marks]

- For: easy to implement interpreter.
- Against: type errors not detected until run-time.
- . . . etc.

Part (ii) [2 marks]

Static typing implies that all type checking is done before run-time . . . but some type errors can still occur at run-time.

Strong typing implies that all type errors are detectable, whether statically or dynamically.

Part (iii) [2 marks]

by-name; by-value

Part (iv) [2 marks]

```
if _ then if _ then _ else _
```

...`else` could be associated with inner or outer `if`

Part (v) [2 marks]

- **expression** returns the value of the consequent or alternative

  statement doesn't!

- Java

Part (vi) [2 marks]

Methods may contain side-effects, such as assignment. Haskell functions are 'pure'.

Part (vii) [2 marks]

P || Q and Q || R -> P || R

Part (viii) [2 marks]

Tasks are blocked and can never become unblocked - no progress possible for these tasks. A cycle of dependencies exists for blocked the tasks.

Part (ix) [2 marks]

A semaphore that only has the values 0 and 1, used for mutual exclusion only. Wait on 0 blocks.

Marking Notes

Part (x) [2 marks]

The child task may access variables (memory) from the parent task.

Answers for Section B: Answer one question from this section

Marking Notes

Question 2 (40 marks)

Part (i) [10 marks]

Static scope is where the appropriate binding is determined *lexically*, and thus can be determined at compile-time . . . it is the 'closest' binding textually.

Dynamic scope is where the binding is determined by the call-sequence of program blocks . . . it is the 'latest' binding made a run-time.

Part (ii) [12 marks]

Sub-part (a) [3 marks]

Sub1: A, X, W

Sub2: B, Z

Sub3: Y

Sub-part (b) [3 marks]

Sub3: A, X, W

Sub1: Y

Main: Z

Sub-part (c) [3 marks]

Sub2: A, B, Z

Sub3: X, W

Sub1: Y

Sub-part (d) [3 marks]

Sub1: A, Y, X

Sub2: B, Z

Sub3: W

Part (iii) [18 marks]

Sub-part (a) [4 marks]

Types of primitives are known, so the type of an expression can be inferred from the types of its components. This means that, in such a system, it is not necessary for a programmer to explicitly state the types of every name.

Sub-part (b) [14 marks]

[Deduction seen in the lectures]

- `map :: s -> [r] -> [q]`, since it is a function of two arguments one of which is a list, and the result is a list (from the first case).
- Second case ⇒ `s` is a function, `p -> t`
- From use of `f x` in the second case, `p` = `r` (element of the `[r]` argument), so `map :: (r -> t) -> [r] -> [q]`. But `f x` is an element of the result list ⇒ `t` = `q`.
- So, final type is `map :: (r -> q) -> [r] -> [q]`.

[Unseen]

a) `foldr :: a -> b -> [c] -> b` from first case

b) `a` = `(p -> q -> r)` from application in second case

c) `r` = `b` — final result of `foldr` (from second case)

d) `q` = `b` — result of `foldr` (from second case)

e) `q` = `c` — element of third argument, from application of `f x` in second case.

f) Thus `foldr :: (c -> b -> b) -> b -> [c] -> b`

Marking Notes

Question 3 (40 marks)

Part (i) [8 marks]

Sub-part (a) [4 marks]

Referential transparency implies that expressions' values only depend on the value of their sub-expressions. So non-dependent sub-expressions could be evaluated concurrently – there are no side-effects.

Sub-part (b) [4 marks]

Side-effects such as assignment — `(set! ...)` — prevent referential transparency.

e.g.
```
(define x 10)
(* x x)
(set! x 20)
(* x x)
```

The two `(* x x)` expressions have different values – boring, but true!

Other examples are acceptable.

Part (ii) [12 marks]

Sub-part (a) [2 marks]

A tail recursive definition is one in which the only recursive cases are tail-calls. A tail-call is where the recursive call *is* the value of the function.

Sub-part (b) [6 marks]

Non-tail call (a linear recursive definition):

```
(define (factorial n)
      (if (= n 1) 1 (* n (factorial (- n 1)))))
```

A tail recursive version (iterative):

```
(define (fact-iter product counter max-count)
   (if (> counter max-count)
       product
       (fact-iter (* counter product)
                  (+ counter 1)
                  max-count)))
```

where `(factorial n)` $\equiv$ `(fact-iter 1 1 n)`

Sub-part (c) [4 marks]

Tail recursion is equivalent to iteration in that it requires constant control-stack space, whereas a (linear) recursive version requires linearly-increases control stack space. This 'optimisation' is required in `scheme` and is available in some compilers for other languages e.g. gcc.

Part (iii) [20 marks]

Sub-part (a) [10 marks]

expt works by using the identity $x^n = x \times x^{n-1}$.

fast-expt uses the identity $x^n = (x^{n/2})^2$ for even n otherwise as `expt`.

Sub-part (b) [10 marks]

Answer will include:

- Half the number of multiplies are required for *even* exponents.
- Every odd exponent is followed by a call with an even exponent (since $n-1$ is even for odd n).
- Hence, for `fast-expt` x^{2n} requires *one* additional multiplication compared with x^n, whereas it requires n more for `expt`. This is the reason for the speed difference.
- Thus the time increases *linearly* for`expt`, but *logarithmically* for `fast-expt`.

NB Marks will be given for a less precise description than the above, providing the fundamental concepts are covered.

Marking Notes

Answers for Section C: Answer one question from this section

Question 4 (40 marks)

Part (i) [6 marks]

Bookwork. Main use is to provide mutual exclusion. Also supports encapsulation.

Part (ii) [10 marks]

Bookwork.

Ada supports protected object (PO) that support mutual exclusion by definition. Java has synchronized methods. A class in Java can have synchronized and non-synchronized methods. In Ada functions have only a read lock on the PO and so concurrent reads are possible.

Condition synchronization in Ada is via barriers to PO entries. In Java it is via notify and notifyall methods.

Part (iii) [12 marks]

Unseen problem; testing the ability to read relatively straightforward code.

Initially the barrier on the Await entry is false and hence any call on Await will suspend. Calls to the procedure Release will result in the value of 'Number' increasing (Note type of parameter is positive and hence N must be at least 1.

All callers to Await will block until Number get a value greater then 10.

Part (iv) [12 marks]

```
package monn;

public class Doer {
    private int value, count;

    public Doer() {
        value = 0;
        count = 0;
    }

    public synchronized void await(){
        try{
        count = count + 1;
        while (value < 10) wait();
        }
        catch (InterruptedException e){
            System.out.println("exception raised");
        }
        count = count - 1;
        if (count == 0) value = 0;
    }

    public synchronized void release(int n){
```

Marking Notes

```
            value = value + n;
            notifyAll();
        }
    }
```

Exception only gives 2 marks - solution could notifyAll only when value goes above 10.

Question 5 (40 marks)

Part (i) [10 marks]

Bookwork. Syn always has message received after sent. If no constraint on sender then asynchronous model. If wait for message to be received then synchronous model. If wait for reply to be returned then remote invocation. (4 marks)

Naming is either direct, or via an intermediary such as a mailbox or channel. (2 marks). It can be symmetric or asymmetric. (2 marks)

Type of message is either a predefined, usually scalar type, or any supported type in the language. (2 marks)

Part (ii) [10 marks]

Erlang uses asynchronous, asymmetric, direct naming and any type including functions. Buffers cannot become full (4 marks). Each related language feature shoudl be briefly described for 6 marks (Bookwork).

Part (iii) [10 marks]

Bookwork. The select statement is used to non-deterministically (1 mark) choose between different accept statements (2 marks). Each could have a guard (2 marks). An execution of the select statement means that one open alternative with a waiting task is chosen for execution (2 marks). If no such task then task is suspended (1 mark). The select statement can also have a terminate alternative (1 mark) and delay or else parts (1 mark). Also note that guards are not reevaluated when calls come in (2 marks).

Part (iv) [10 marks]

Unseen problem.

The server task must define three entries:

```
task Membership is
  entry Join;
  entry Leave;
  entry Suspend;
  entry Un_Suspend;
end Membership;
```

The body of this task needs a loop and a select statement:

```
task body Membership is
  Members : integer := 0;
begin
  loop
    select
      when Members < 100 do
        accept Join;
```

```
            Members := Members + 1;
        or
          accept Suspend;
        or
          accept Un_Suspend;
        or
          accept Leave;
          Members := Member - 1;
          -- perhaps check is members goes negative
          -- or put a guard on this entry
        or
          when Members = 0 do
          terminate;
        end select;
      end loop;
    end Membership;
```