VICTORIA UNIVERSITY OF WELLINGTON

Te Whare Wananga o te Upoko o te Ika a Maui

EXAMINATIONS — 2001

END OF YEAR

COMP 304

Programming Languages

Time Allowed: 3 Hours

Instructions: There are 5 (five) questions, each worth 40 (forty) marks. You should attempt all of the questions.
You are not allowed to use electronic foreign language dictionaries.

Question 1. [40 marks]

(a) Give an example of an inductively defined type in Haskell, and explain how the structure of the type definition helps us to write functions over the type. [7 marks]

(b) Explain what we mean by the term *parametric polymorphism* and explain what advantages, if any, *parametric polymorphism* provides for the functional programmer. [11 marks]

(c) Explain what we mean by the term *higher-order function* and explain what advantages, if any, *higher-order functions* provide for the functional programmer. [11 marks]

(d) Explain what we mean by the term *lazy evaluation* and explain what advantages, if any, *lazy evaluation* provides for the functional programmer. [11 marks]

Question 2. [40 marks]

(a) Prolog programs are often described as having both a declarative semantics and a procedural semantics. Explain what we mean by the terms *declarative semantics* and *procedural semantics*.

[6 marks]

(b) Give examples of two different situations were Prolog predicates whose declarative semantics are unclear are used. [4 marks]

Polish notation allows us to represent arithmetic expressions in a compact form without using brackets. We write all expressions in prefix form, with the principal operator written at the front. For example:

$$\begin{array}{rll} 2+3 & \text{is written as} & +\ 2\ 3 \\ 4+3*5 & \text{is written as} & +\ 4\ *\ 3\ 5 \\ (4+3)*5 & \text{is written as} & *\ +\ 4\ 3\ 5 \\ -2+3-4 & \text{is written as} & -\ +\ -\ 2\ 3\ 4 \end{array}$$

We can represent such expressions in Prolog using lists, e.g.:

```
[add, 2, 3]
[add, 4, times, 3, 5]
[times, add, 4, 3, 5]
[sub, add, neg, 2, 3, 4]
```

Given an expression written in Polish notation we may want to evaluate it. We might write the following code:

```
eval([N], N) :- number(N).

eval([add | L], N) :-
    append(A, B, L),
    eval(A, An),
    eval(B, Bn),
    N is An + Bn.

eval([times | L], N) :-
    append(A, B, L),
    eval(A, An),
    eval(B, Bn),
    N is An * Bn.

eval([sub | L], N) :-
    append(A, B, L),
    eval(A, An),
    eval(B, Bn),
    N is An - Bn.

eval([neg | L], -N) :-
    eval(L, N).
```

The predicate `number/1` has already been defined for us. A call to `number(N)` will succeed if `N` has been instantiated to a number. So `number(6)` will succeed, and `number(X)` and `number(helen)` will fail.

(c) Implement `append/3`, such that a call to `append(L1, L2, L3)` will succeed if the list `L3` is `L2` appended to `L1`. [4 marks]

(d) Explain why the definition of `eval/2` is rather inefficient. [10 marks]

(e) Re-implement `eval/2`, avoiding this inefficiency, by implementing `eval/3`, such that:

```
eval(L, V) :- eval(L, [], V).
```

[14 marks]

(f) Describe the **minor** change you would make to your code to allow it to translate between Polish notation and the more common infix notation. [2 marks]

Question 3. [40 marks]

(a) Explain what benefits there are for each of the following people in having a formal semantics for a programming language:

1. language designers; [4 marks]
2. compiler writers; [4 marks]
3. programmers. [4 marks]

(b) Explain how we present a *denotational semantics* for a programming language. [8 marks]

(c) Present a semantic algebra for the natural numbers, and use this to give a denotational semantics for binary numerals. [8 marks]

(d) Explain how we present an *axiomatic semantics* for a programming language. [8 marks]

(e) Explain how can use the formalism of axiomatic semantics to reason about the correctness of programs. [4 marks]

Question 4. [40 marks]

(a) Describe the control structures which you would expect every typical, modern, imperative programming language to have. [10 marks]

(b) Carefully describe the actual control structures provided by an imperative programming language with which you are familiar. [8 marks]

(c) Explain why the designer or designers of the language whose control structures you have described in **b** chose to provide those control structures. [6 marks]

(d) Explain what the following terms mean:

1. pass-by-value; [4 marks]
2. pass-by-name; [4 marks]
3. pass-by-reference. [4 marks]

(e) Explain why pass-by-need would not be appropriate for an imperative programming language. [4 marks]

Question 5. [40 marks]

Write short notes on each of the following topics:

1. The support for abstraction provided by *either* Java *or* C++; [8 marks]
2. Why there are so many programming languages; [8 marks]
3. How to compare programming languages; [8 marks]
4. What object-oriented programming is good for, and why; [8 marks]
5. What logic programming is good for, and why. [8 marks]

****