VICTORIA UNIVERSITY OF WELLINGTON

Te Whare Wananga o te Upoko o te Ika a Maui

EXAMINATIONS — 2003

END-YEAR

COMP 304

Programming Languages

Time Allowed: 3 Hours

Instructions: There are 5 (five) questions, each worth 40 (forty) marks. You should attempt all the questions.

Question 1.

(a) Explain how we present an operational semantics for a programming language. [9 marks]

(b) Give operational semantic rules suitable to describe the behaviour of the following constructs in a typical imperative programming language:

(i) binary conditional (`if...then...else...`) [5 marks]

(ii) pre-test loop [6 marks]

(c) Explain how we present an axiomatic semantics for a programming language. [9 marks]

(d) Give axiomatic semantic rules suitable to describe the behaviour of the following constructs in a typical imperative programming language:

(i) simple conditional (`if...then...`) [5 marks]

(ii) pre-test loop [6 marks]

Question 2.

(a) Explain what the following terms mean:

(i) pass-by-value [6 marks]

(ii) pass-by-name [6 marks]

(iii) pass-by-reference [6 marks]

(b) What will the output of the following program be if the parameter passing mechanism used in the language is:

(i) pass-by-value [5 marks]

(ii) pass-by-reference [5 marks]

(iii) pass-by-name [5 marks]

```
int i;
int a[3];

void swap( int x, int y )
{ x = x + y;
  y = x - y;
  x = x - y;
}

main()
{ i = 0;
  a[0] = 2;
  a[1] = 1;
  a[2] = 0;
  swap(i, a[i]);
  printf("%d %d %d %d", i, a[0], a[1], a[2]);
  return(0);
}
```

(c) Explain why 'pass-by-need' is an appropriate choice for a functional programming language, and an inappropriate choice for an imperative language. [7 marks]

Question 3.

(a) Explain the difference between the procedural and the declarative semantics of a Prolog program. [14 marks]

(b) The predicate `replaced/2` tells us who replaced who as New Zealand Prime Minister, since Rob Muldoon replaced Bill Rowling on Friday, 12 December 1975. The predicate `later_than/2` tells us which Prime Minister was later than which other Prime Minister.

```
% replaced(New, Old) holds if New replaced Old as PM.

replaced(muldoon, rowling).
replaced(lange, muldoon).
replaced(palmer, lange).
replaced(moore, palmer).
replaced(bolger, moore).
replaced(shipley, bolger).
replaced(clark, shipley).

% later_than(New, Old) holds if New was a later PM than Old.

later_than(A, B) :-
   replaced(A, C),
   later_than(C, B).

later_than(A, B) :-
   replaced(A, B).
```

Explain what response the following queries will get:

(i)

```
?- later_than(X, savage).
```

[3 marks]

(ii)

```
?- later_than(clark, Y), write(Y), nl, fail.
```

[5 marks]

(Question 3 continued on next page)

(Question 3 continued)

(iii) Give a definition of `later_than/3` such that `later_than(New, Old, Tween)` holds if `New` was later than `Old`, and `Tween` is the list of Prime Ministers between them. For example:

```
?- later_than(clark, shipley, []).
yes

?- later_than(clark, palmer, PMs).

PMs = [shipley,bolger,moore]
yes
```

[6 marks]

(c) You have a friend who is doing another paper at University where he has to implement a program in Prolog to parse very simple sentences in English. He has written the following code:

```
det([the]). % 'the' is a determiner.
n([cat]).   % 'cat' is a noun.
n([mouse]). % 'mouse' is a noun.
iv([ate]).  % 'ate' is an intransitive verb.
tv([ate]).  % 'ate' is also a transitive verb.

s(L) :-     %  Rule: S ->  NP, VP
   append(NP, VP, L),
   np(NP),
   vp(VP).

np(NP) :-    %  Rule: NP ->  Det, N
   append(Det, N, NP),
   det(Det),
   n(N).

vp(VP) :-    %  Rule: VP ->  TV, NP
   append(V, NP, VP),
   tv(V),
   np(NP).

vp(VP) :-    %  Rule: VP ->  IV
   iv(VP).
```

(Question 3 continued on next page)

(Question 3 continued)

He is pleased that this code works, but is disappointed at how slowly it executes.

Explain why this code is inefficient, and re-implement it to remove the inefficiency.

[12 marks]

Question 4.

(a) Explain the criteria which we might use to assess a programming language, and use these to evaluate a programming language with which you are familiar. [10 marks]

(b) Describe the model of computation which underlies imperative programming. [10 marks]

(c) Logic, functional and imperative programming languages all have variables. Carefully explain how the notion of variable differs between them. [10 marks]

(d) Explain what benefits there are, if any, of providing a formal semantics for a programming language. [10 marks]

Question 5.

(a) Haskell provides higher-order functions. State what it means for a function to be a higher-order function. [2 marks]

(b) In Haskell the function `(+)` can be given the type `Int -> Int -> Int`. State whether `(+)` is, or is not, a higher-order function, and explain why. [4 marks]

(c) Explain what benefits, if any, higher-order functions provide for the programmer. [10 marks]

(d) Describe the facilities that Haskell provides which allow the programmer to define new types. [10 marks]

(e) The function `map` may be defined in Haskell as:

```
map _ [] = []
map f (h:t) = (f h) : map f t
```

The function `fold` may be defined in Haskell as:

```
fold d _ [] = d
fold d e (h:t) = e h (fold d e t)
```

(i) State a suitable type for `fold` and a suitable type for `map`. [4 marks]

(ii) Show how `map` can be defined using `fold`. [5 marks]

(f) The type of binary trees may be defined as:

```
data Tree a = Leaf
            | Node a (Tree a) (Tree a)
```

The function `treemap` allows us to map a function over a tree, just as `map` allows us to map a function over a list.

Define `treemap`. [5 marks]

****