TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

EXAMINATIONS — 2005

MID-YEAR

COMP 304

Programming Languages

Time Allowed: 3 Hours

Instructions: There are 5 (five) questions.
Questions 1, 2, 3 and 4 are each worth 30 (thirty) marks.
Question 5 is worth 20 (twenty) marks.
There are 140 (one hundred and forty) marks in total.
You should attempt all the questions.

Question 1.

(a) You are designing two different imperative programming languages:

- **TeachingLang** – to be used to teach imperative programming to beginning students;
- **IndustrialLang** – to be used in industry to implement large programs by expert programmers.

(i) Describe the control structures you would provide for **TeachingLang**, and explain why this is a good choice. [8 marks]

(ii) Describe the control structures you would provide for **IndustrialLang**, and explain why this is a good choice. [8 marks]

(iii) State which parameter passing mechanism or mechanisms you would provide for **TeachingLang**, and explain why this is a good choice. [6 marks]

(b) In many artificial intelligence applications a 'generate and test' strategy is used. This strategy works by generating possible solutions, and then testing them to see if they are actual solutions. For example, you could find someone's password by generating every possible sequence of characters and testing them all. Game playing programs often choose the next move to make by generating all possible outcomes from the current state of the game and choosing the move which give the best outcome.

Briefly describe lazy evaluation, and how it can help in implementing a 'generate and test' strategy. [8 marks]

Question 2.

(a) State three benefits of providing a formal semantics for a programming language. [6 marks]

(b) State one advantage that operational semantics has over axiomatic semantics. [6 marks]

(c) State one advantage that axiomatic semantics has over operational semantics. [2 marks]

(d) You are defining a programming language and have defined the following abstract syntax:

$$\begin{array}{rcl} P & \in & \mathit{Program} \\ C & \in & \mathit{Command} \\ E & \in & \mathit{Expression} \\ B & \in & \mathit{BooleanExpr} \\ I & \in & \mathit{Identifier} \\ N & \in & \mathit{Numeral} \end{array}$$

$$\begin{array}{rcl} P & ::= & C. \\ C & ::= & C1\ ;\ C2 \\ & | & \texttt{if}\ B\ \texttt{then}\ C \\ & | & \texttt{if}\ B\ \texttt{then}\ C1\ \texttt{else}\ C2 \\ & | & I := E \\ E & ::= & E1 + E2 \mid E1 - E2 \mid E1 * E2 \mid E1\ /\ E2 \mid I \mid N \\ B & ::= & E1 == E2 \mid \neg B \mid B1 \wedge B2 \end{array}$$

(i) You have decided that the store will be a mapping from identifiers to numbers. Hence the semantic algebra for stores look like:

$$\begin{array}{l} \mathit{Domain\ Store} = \mathit{Id} \rightarrow \mathit{Nat} \\ \mathit{Operations} \\ \texttt{Operation defintions} \end{array}$$

Define the operations that this semantic algebra will provide. [6 marks]

(ii) The valuation function for commands will be

$$\mathbf{C} : \mathit{Command} \rightarrow \mathit{Store}_{\perp} \rightarrow \mathit{Store}_{\perp}$$

Give a suitable valuation function for *Command*s, assuming that you have suitable valuation functions for *Program*, *Expression*, *BooleanExpr*, *Identifier*, *Numeral*. [10 marks]

Question 3.

(a) Describe a model for imperative programming. [6 marks]

(b) For an imperative language with which you are familiar, describe three similarities to the model and describe three differences from the model. [6 marks]

(c) The λ-calculus provides a model for functional programming. For a functional language with which you are familiar, describe three similarities to the λ-calculus and describe three differences from the λ-calculus. [6 marks]

(d) Unification is closely related to equation solving: unifying two terms t_1 and t_2 can be thought of as solving the equation $t_1 = t_2$ for the variables which occur in the terms. For each of the following equations, either solve it, or show that no solution exists. (x, y, z are variables and f, g, h are constants):

(i). $f(x) = f(h)$

(ii). $f(x) = g(y)$

(iii). $f(x, y) = f(y, g)$

(iv). $f(x, y) = f(g(y), x)$

[4 marks]

(e) Describe how unification is used in Prolog. [8 marks]

Question 4.

(a) Briefly describe the following terms, as they are understood by a Prolog programmer:

(i). fact;

(ii). rule;

(iii). query.

[6 marks]

(b) In the English feudal system each man belongs to one of the following ranks, listed in descending order:

- King
- Duke
- Lord
- Knight
- freeman
- serf

You have a friend who is studying Computer Science and History. Your friend has attempted to capture this ranking using the following Prolog code.

```
/*
above(Upper, Lower) holds if Upper is immediately above  Lower in the ranking.
*/

above(king, duke).
above(duke, lord).
above(lord, knight).
above(knight, freeman).
above(freeman, serf).

/*
outranks(Upper, Lower) holds if Upper is higher than  Lower in the ranking.
*/

outranks(Upper, Lower) :-
    outranks(Upper, Mid),
    above(Mid, Lower).
outranks(Upper, Lower) :-
    above(Upper, Lower).
```

(Question 4 continued on next page)

(Question 4 continued)

(i) Explain why this code does not give the intended solution for the query:

```
outranks(king, Duke).
```

and give a correct definition for `outranks/2`. [8 marks]

(ii) Your friend has defined `not/1` in the usual way, and defines `nobetterthan/2` as follows:

```
nobetterthan(One, Other) :- not(outranks(One, Other)).
```

State the results of the following queries:

- `nobetterthan(serf, lord).`
- `nobetterthan(serf, Lord).`
- `nobetterthan(knight, lord).`
- `nobetterthan(knight, Lord).`

[8 marks]

(c) The predicate `clause/2` is defined so that `clause(Head, Body)` will succeed if there is a rule `Head :- Body` in the database.

Show how to use `clause/2` to write a predicate `explain/2`, which takes a goal and will succed if the goal does, and will give an explanation of why the goal succeeded. For example:

```
| ?- explain(outranks(king, lord), Ans).

Ans = outranks(king,lord)
       because above(duke,lord) and
              (outranks(king,duke) because above(king,duke))
```

You may assume that `and` and `becuase` have been defined already. [8 marks]

Question 5.

(a) State what it means for a function to be a *higher-order function* [4 marks]

(b) The type of binary trees can be defined in Haskell as:

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

(i) State the constructors of the type `Tree a`. [2 marks]

(ii) Give a higher-order function, `treefun`, which captures the typical pattern of computation on such binary trees. State the type of `treefun`. [4 marks]

(iii) Write a function, `treesum`, which computes the sum of the values in a binary tree. State the type of `treesum`. [4 marks]

e.g.

```
treesum (Node (Node Leaf 2 Leaf) 1 (Node Leaf 3 Leaf))
```

will evaluate to

```
0
```

(iv) Write a function, `prune :: Int -> Tree a -> Tree a`, which prunes a tree to a given depth. The pruned tree should have a maximum depth equal to the integer supplied, and should be the same as the input tree up to this depth.

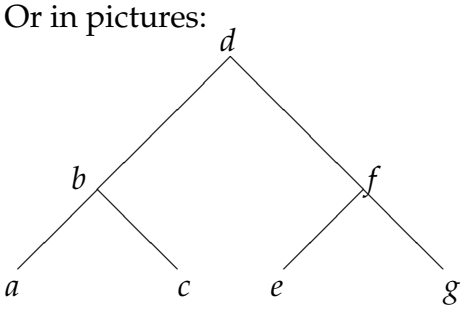
e.g.

```
prune 1 (Node (Node (Node Leaf 'a' Leaf) 'b' (Node Leaf 'c' Leaf)) 'd'
         (Node (Node Leaf 'e' Leaf) 'f' (Node Leaf 'g' Leaf)))
```

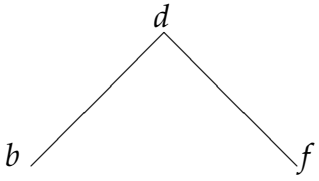
will evaluate to

```
Node (Node Leaf 'b' Leaf) 'd' (Node Leaf 'f' Leaf)
```

Or in pictures:



pruned to depth 1 is



[6 marks]

****