UNIVERSITY OF SWAZILAND

FINAL EXAMINATION 2005

Title of paper: PROGRAMMING LANGUAGES

Course number: CS343

Time allowed: Three (3) hours

Instructions: Answer any five (5) of the seven (7) questions.

This examination paper should not be opened until permission has been granted by the invigilator.

Question 1

(a) Describe in detail the main phases of a compiler.

[14]

(b) Describe the reference-counting method of automatic storage reclamation.

[6]

Question 2

(a) Explain the main points made in favour of each side in the following debates:

(i) Imperative vs declarative programming.

(ii) Static vs dynamic typing.

[8]

(b) *Modularity*, *interface/implementation separation* and *separate compilation* are features that support programming-in-the-large. Explain each feature.

[12]

Question 3

(a) (i) Distinguish between the 4 kinds of polymorphism: overloading-, conversion-, inclusion- and parametric-polymorphism.

[8]

(ii) Give short examples in C++ to demonstrate any 2 kinds of routine polymorphism.

[4]

(b) What is *dynamic dispatch*?

Explain how dynamic dispatch is implemented by C++ though *vtables*.

[8]

Question 4

(a) Define the following terms as they relate to functional programming:

(i) Type class.

(ii) Higher-order function.

(iii) Lazy evaluation.

(iv) Referential transparency.

[8]

(b) (i) Describe the procedure for reducing a λ–calculus expression to normal form.

[4]

(ii) Show how the following λ–calculus expression is reduced to normal form:

$$(((\lambda x.x)\ (\lambda y.y*y))\ ((\lambda z.z+1)\ 2))$$

[8]

77

Question 5

(a) State 2 examples of extra-logical features of Prolog.

[2]

(b) Give a recursive definition of Prolog *structures*.

[4]

(c) Distinguish between the 2 following equality predicates of Prolog:

```
=:=          =
```

[4]

(d) Assuming that the program given below has already been entered into Prolog, draw the search tree for the following query:

```
% This is the query:
pass(test1, Student), pass(test2, Student).

% Program follows:
marks(test1, joe, 90).
marks(test1, sam, 45).
marks(test2, joe, 75).
marks(test2, sam, 100).
pass(Test, Student) :-
     marks(Test, Student, M),
     M >= 50.
```

[10]

78

Question 6

Define a function in Haskell that accepts 2 integer arguments, multiples them using the Russian Peasant's algorithm (described below) and returns the product.

[20]

Russian Peasant's algorithm

This algorithm takes 2 positive integers, X and Y, and computes their product.

A series of pairs is constructed, such that the initial pair is comprised of the original factors, for example with X on the left and Y on the right: (X, Y). The second member of the series is a pair whose left element is 2X and whose right element is $\lfloor Y/2 \rfloor$.[1] In general if the *i*-th member of the series is (X_i, Y_i), then the *(i+1)*-th member is $(2X_i, \lfloor Y_i/2 \rfloor)$.

The series terminates with the pair whose right element is 1.

The final product is the *sum of the left elements* of each pair in the series whose *right elements are odd numbers.*

Example 1: to multiply 23 by 11, the following series is constructed:

[(23, 11), (46, 5), (92, 2), (184, 1)]

... and the result is 23+46+184, which equals 253. 92 is not added because it is paired with an even number on the right.

Example 2: to multiply 100 by 10, we construct [(100, 10), (200, 5), (400, 2), (800, 1)], and obtain the product by adding 200 and 800. 100 and 400 are ignored because they are paired with even numbers on the right.

[1] The notation $\lfloor Y/2 \rfloor$ denotes the quotient (integer part) of Y/2. E.g. $\lfloor 7/2 \rfloor$ is 3 and $\lfloor 11/2 \rfloor$ is 5.

Question 7

(a) Assume that results of football matches have already been entered into Prolog, under a predicate named `match(Team1, Team2, Goals1, Goals2)`, where `Team1` and `Team2` are symbols denoting the names of the 2 opposing teams and `Goals1` and `Goals2` are integers denoting the number of goals scored during the match by `Team1` and `Team2`, respectively.

(i) One of the teams is named `aces`. Write a query to determine the names of teams that have played against `aces`. Note that the symbol `aces` may appear in either of the `Team1` or `Team2` positions within the `match` predicate.

[3]

(ii) Write a query to create a list of all matches that were drawn (i.e. where both teams scored the name number of goals.)

[6]

(b) Define a recursive Prolog predicate `listlen(List, Len)` that succeeds when the `List` argument (which has already been bound to a list) has length `Len`.

[4]

(c) Define a recursive Prolog predicate `minpositive(Nums, Min)` that succeeds when `Min` is the smallest positive number inside the `Nums` list argument (which has already been bound to a list of numbers.) If the `Nums` list is empty, or contains no positives, the predicate should bind `Min` to zero.

[7]