Thursday, 28th April 2011
9.30am – 11.00am
(Duration: 1 hour 30 minutes)

DEGREES OF MSci, MEng, BEng, BSc, MA and MA (Social Sciences)

COMPUTING SCIENCE 3Z: PROGRAMMING LANGUAGES 3

Answer all 4 questions.

This examination paper is worth a total of 60 marks.

You must not leave the examination room within the first half-hour or the last fifteen minutes of the examination.

1. (*Functional programming*)

(a) Given a list of integers, `ns`, write the following:

(i) a list comprehension that yields a list of Boolean values, in which each element is `True` if the corresponding element of `ns` is even, or `False` if the element is odd;

(ii) a list comprehension that yields a list of integers, in which each element is one less than the corresponding element of `ns`, except that non-positive elements of `ns` are discarded.

[2]

(b) Given a list of Boolean values, `bs`, define the following functions:

(i) "`all bs`" yields `True` if and only if `bs` is empty or all its elements are `True`;

(ii) "`any bs`" yields `True` if and only if `bs` is non-empty and at least one of its elements is `True`.

Make sure that your functions are efficient.

Your answer must explicitly declare the type of each function. *Do not use any list functions from the Haskell standard prelude*.

[4]

(c) Given an arbitrary list, `xs`, define the following polymorphic functions:

(i) "`map f xs`" yields a list in which each element is the result of applying function `f` to the corresponding elements of `xs`;

(ii) "`filter f xs`" yields a list containing just those elements of `xs` satisfied by function `f` (i.e., those elements `x` for which `f x` yields `True`).

Your answer must explicitly declare the type of each function. *Do not use any list functions from the Haskell standard prelude*.

[9]

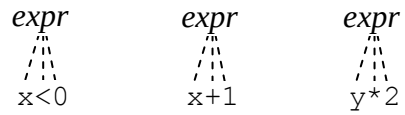
2. (*Syntax*)

Box 1 shows part of the BNF grammar of a fictional programming language, FPL. It shows only the syntax of commands and sequential commands. (The syntax of expressions is not needed in this question, and is not shown in Box 1.)

(a) By drawing two different syntax trees, show that the following FPL sequential command is ambiguous:

```
while x<0 do x = x+1; y = y*2;
```

You may assume that `x<0`, `x+1`, and `y*2` are expressions. Your syntax tree should show these expressions in outline:



[6]

(b) Suggest how the grammar might be modified to eliminate the ambiguity. Your modified grammar must still allow a sequential command within a loop body.

[4]

seq-command	::=	*command*
	|	*seq-command command*
command	::=	*ident* = *expr* ;
	|	`while` *expr* `do` *seq-command*
	|	…

Box 1 Part of the grammar of a fictional programming language FPL. (Here *expr* is an expression, and *ident* is an identifier.)

3. (*Implementation*)

(a) The Microsoft common-language run-time system (known as CLR or .NET) consists of a suite of front-end compilers translating high-level languages into CLR intermediate code, plus a suite of back-end compilers translating CLR code into native machine code. The front-end compilers run on the software developer's machine, but the back-end compilers run on the customer's machine.

Using this system, a software developer writes an application program in a high-level language, compiles it, and distributes it as CLR code.

Draw tombstone diagrams for (i) a C++ front-end compiler; (ii) a C# front-end compiler; (iii) a PPC back-end compiler; (iv) a P4 back-end compiler.

[4]

(b) Using tombstone diagrams, show how an application program *P* written in C# would be (i) compiled, and (ii) run on a customer's PPC machine.

Comment on your answer to (ii) – when exactly would PPC native code be generated?

[5]

(c) Compare and contrast the CLR system with the Java Development Kit. What are the strengths and weaknesses of each?

[6]

4. (*Concepts*)

(a) What is meant by an *abstract data type* (*ADT*)?

[2]

(b) Explain briefly how ADTs are supported by (i) Java and (ii) Haskell.

Illustrate your answer by outlining (i) a Java program-unit and (ii) a Haskell program-unit, each implementing an ADT whose values are sets of integers. It must be possible to construct an empty set, to add a given integer to a set, and to test whether a given integer is in a set. The data structure representing the set must be hidden.

Your outlines should show the headings of all operations, but should not show the data structure or how the operations are implemented.

[8]

(c) Show how you would generalize your Java and Haskell program-units of part (b) to supports sets of elements of an arbitrary type.

[6]

(d) Suppose that an additional requirement is now introduced: it must be possible to obtain the *least* value in a set. Show how your Java program-unit of part (c) would be modified.

[4]