THE BRITISH COMPUTER SOCIETY
THE BCS PROFESSIONAL EXAMINATION
Professional Graduate Diploma

April 2002

Programming Paradigms

The pass rate was 92%. The highest mark was 75%. The general level of candidates' performance was therefore satisfactory although the mean of 51% is indicative of the fact that few candidates did really well.

Question 1

The expression *"Horses for Courses"* is a term used in the racing world to mean choosing the right racehorse for a particular racecourse. In the programming world, there exists a variety of programming languages that a programmer can use to implement a computer system. The same expression is used to mean choosing the right language for the right application.

Choose TWO different types of programming language (such as data-oriented, imperative, object-oriented), and compare and contrast them, discussing what characteristics they have and what type of applications they are most appropriate for. (25 marks)

Answer Pointers

The question is open to a variety of answers depending on which programming types have been chosen for the examples. The syllabus covers imperative languages (e.g., Fortran, COBOL, PL/1, Algol, C, etc.), data-oriented languages (APL, SETL), object-oriented languages (C++, Smalltalk, Java) and non-imperative languages (Lisp, ML, Prolog).

A description of the characteristics of the chosen languages was expected. It was also important that candidates discuss what types of applications the language is most suitable for and a good answer would relate the characteristics to the application. For example, imperative languages are more general-purpose languages and can be used for a variety of applications, whereas non-imperative languages include functional and logical languages and are targeted at specialised programming applications.

Illustrations of the characteristics and points discussed were expected to demonstrate the candidates' practical understanding of the theoretical points discussed.

Candidates offered few examples. Often they were good at one of the languages chosen but weak on the other - generally they chose OO and imperative. Some seemed to be guessing that their alternative language was just the opposite of the one they were "good" at.

Sometimes answers were very superficial, with little expansion on points made.

Quite a few candidates wrote very little, or nothing at all, on appropriate applications for the languages.

Question 2

a) **Interactive Development Environments (IDEs) are now commonplace for most programming languages. Describe briefly the sorts of features available in such an environment to support the programming development process.**

(10 marks)

b) **Evaluate the success of these tools in improving the productivity of programmers and the quality of the code they produce.**

(15 marks)

Answer Pointers

Candidates were expected to discuss the sort of tools available in an IDE for developing a program, from writing the code to testing it, plus debugging and configuration management utilities.

This part of the question was generally well answered.

(b) Evaluate the success of these tools in improving the productivity of programmers and the quality of the code they produce.

This question required candidates to reflect on the real usefulness of these tools. Within their discussion they should have indicated why they thought the tools were important or not. For example, they might have argued that they improve the speed and performance of the programmer, help with debugging, testing, etc. Disadvantages could include that the programmer may fall into bad programming practices by relying on the tool to correct mistakes, rather than checking and testing it thoroughly themselves.

For both parts, examples were expected of appropriate tools, for example, Microsoft Visual Studio, Borland C++, Borland Jbuilder, etc. They needed to identify what elements of the tool are useful to productivity and to say why they thought they were beneficial or not.

This was not as well answered as section (a). Some candidates gave a good list of advantages but little on disadvantages and some did the opposite. Some candidates just repeated what they had said in section (a) with little extra discussion. In general, answers were reasonably good at looking at how the programmer's productivity was speeded up, but not on reflecting on the quality of the code.

Question 3

a) **Discuss the role of constructors, pattern matching and recursion, when using compound types within an applicative (functional) programming language. Provide illustrative examples of their role within a functional programming language with which you are familiar.**

(15 marks)

b) **Lazy evaluation offers significant advantages over that of eager evaluation when performed within the context of functional programming. Distinguish between these types of evaluation, and describe what these advantages are.**

(10 marks)

The selection of two different aspects of functional programming was meant allow candidates to demonstrate their breadth of understanding of the issues involved.

In this section, candidates are expected to describe how, for example, lists can be created using constructors which in turn can be used when defining functions by pattern matching. The importance of recursion in these functions should have been discussed, as, the absence of "statements," it is the only way of creating loops in expression evaluation. Candidates were expected to provide illustrative examples of the respective roles that could describe, for example, how to define a (recursive) type for trees whose nodes are labelled with integers, and, in turn, how this type can be used to write functions that process the tree. Any such example would have been favorably received.

A few good points were made but answers did not address the question properly. Constructors and pattern matching were not addressed.

(b) Lazy evaluation offers significant advantages over that of eager evaluation when performed within the context of functional programming. Distinguish between these types of evaluation, and describe what these advantages are.

In answer to section (b), candidates were expected to define the terms used, and provide suitable illustrative examples for clarification. The importance of the implications (and hence advantages) of each evaluative mechanism should have been demonstrated in real terms, which might relate to storage, or time-based outcomes. Credit would have been given for any arguments correctly put.

This section was badly answered. No illustrative examples were given, and no evaluation mechanism was demonstrated in real terms of storage or real-time outcomes.

Question 4

a) **Real logic programming languages require "extra-logical" features. Discuss this statement and comment upon any compromises that these features might create.**

(12 marks)

b) **It has been proposed that logic programming languages should be integrated with other programming paradigms such as functional programming. Present arguments for and against this integration.**

(13 marks)

Answer Pointers

The selection of two different aspects of logic programming should have allowed candidates to demonstrate their breadth of understanding of the issues involved.

In section (a) of the question, candidates were expected to demonstrate their knowledge of the basic concepts of pure logic programming. Based upon their familiarity with a real logic programming language, candidates were expected to understand that 'extra-logical' features are required, which are not in keeping with a pure logical approach. The compromise discussion might refer to logical or non-

logical aspects, such as interface design or time dependency versus simplicity and consistency. No reasonable approach was rejected.

Not all students properly understood the term “extra-logical”. Some students even considered that a logic program that was capable of artificial intelligence solving capabilities was “extra-logical”. Nevertheless, there were some good answers from those students who did understand the term.

(b) It has been proposed that logic programming languages should be integrated with other programming paradigms such as functional programming. Present arguments for and against this integration.

In answer to this section, candidates are expected to present ‘for’ arguments that might include the very close relationship between the mathematics of functions (lambda calculus) and logic. The ‘against’ arguments were expected to be greater in number, and might have included differences in unification, use of inference engine for logic programming, etc. Credit was given for any arguments correctly put.

Generally the responses to this section were poor. Some students did not attempt to answer this section, or, if they did, were unable to formulate arguments ‘for and against’ integration.

Question 5

a) **Describe what is meant by the term *concurrency* when used within the context of multi-programming operating systems, multi-tasking operating systems and embedded systems. Illustrate your answer with examples for each system. (10 marks)**

b) **Explain the mutual exclusion problem for concurrent programs. Using simple examples from programming language examples that support concurrency, discuss the advantages, or otherwise, of the possible solutions to this problem. (15 marks)**

Answer Pointers

a) Candidates were expected to demonstrate some breadth of knowledge of concurrent systems, and to illustrate the application of this type of computing using appropriate examples, drawn from their own experience, and/or background reading. They were expected to define concurrency within the context of the three systems, and thereby introduce and describe additional terms such as time-sharing, task scheduling, real-time, and processes.

A number of students failed to describe the term ‘concurrency’ within the context asked for. They either ignored the systems mentioned, or failed to define them correctly.

b) Explain the mutual exclusion problem for concurrent programs. Using simple examples from programming language examples that support concurrency, discuss the advantages, or otherwise, of the possible solutions to this problem.

Candidates are expected to introduce and define terms such as interleaving and atomic instructions, and to use these to explain the mutual exclusion problem. Concepts such as critical and non-critical code sections and Dekker’s algorithm were expected to be introduced, together with solutions such as semaphores and monitors. Credit was given for any clear illustration as to the advantages or otherwise of these solutions.

Generally there were some very good answers to this section. A number of students were able to introduce examples using real concurrency supporting programming languages. These answers attracted greater credit. No student mentioned Dekker's algorithm.